ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное
образовательное бюджетное учреждение
высшего профессионального образования
«САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ
им. проф. М. А. БОНЧ-БРУЕВИЧА»

М. П. Белов, Ф. В. Филиппов

ИНТЕЛЛЕКТУАЛИЗАЦИЯ ИНФОКОММУНИКАЦИОННЫХ СИСТЕМ

Часть 2

УЧЕБНОЕ ПОСОБИЕ

СПб ГУТ)))

САНКТ-ПЕТЕРБУРГ
2014

УДК 004.032(075.8)
ББК 32.973-018я73
Б43

Рецензенты:
доктор технических наук, профессор кафедры системного анализа и управления Национального минерально-сырьевого университета «Горный» *Д. А. Первухин*,

доктор технических наук, профессор кафедры автоматизированного электропривода и электротехники Санкт-Петербургского государственного университета растительных полимеров *В. Д. Кулик*

Утверждено редакционно-издательским советом СПбГУТ в качестве учебного пособия

Белов, М. П.
Б43 Интеллектуализация инфокоммуникационных систем : учебное пособие. Часть 2 / М. П. Белов, Ф. В. Филиппов ; СПбГУТ. – СПб., 2014. – 80 с.

Приводятся основы теории интеллектуализации инфокоммуникационных систем, рассмотрено применение нейронных сетей, генетических алгоритмов, Интернет-технологии, смысловая разметка контента, набор данных для поисковых машин, универсальный способ представления знаний, автоматизация интерпритации контента, язык SPARQL.

Предназначено для студентов, обучающихся по направлению подготовки 09.03.02, а также для аспирантов и специалистов в области телекоммуникаций. Возможно использование при выполнении курсовых и дипломных работ.

УДК 004.032(075.8)
ББК 32.973-018я73

СОДЕРЖАНИЕ

3. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ДЛЯ РЕАЛИЗАЦИИ И МОДЕЛИРОВАНИЯ НЕЙРОННЫХ СЕТЕЙ

3.1. Определение и классификация нейропроцессоров

Разработку нейрочипов и на их основе нейрокомпьютеров осуществляют во многих странах.

В настоящее время сложился ряд отечественных школ, занимающихся исследованиями в области нейронных сетей и имеющих значительный научно-технический потенциал:

– НЦН РАН (А. И. Галушкин, Москва);

– группа «НейроКомп» СО РАН (А. Н. Горбань, Новосибирск);

– НТЦ «Модуль» и МАК «Вымпел» (М. Ф. Яфраков, Москва);

– лаборатория нейроинформатики СПИИРАН (А. В. Тимофеев, Санкт-Петербург);

– отделение «Нейронные сети, нейроматематика, нейрокомпьютеры» Международной академии информатизации (В. Л. Дунин-Барковский, Москва) и др.

Нейропроцессор – это кристалл, который обеспечивает выполнение нейросетевых алгоритмов в реальном масштабе времени.

Нейропроцессоры, используемые в системах управления электромеханическими объектами, реализуются на основе разновидностей кристаллов:

– заказные кристаллы (ASIC);

– специализированные нейрочипы;

– перепрограммируемые логические интегральные схемы (FPGA, ПЛИС);

– процессоры цифровой обработки сигналов (ПЦОС);

– встраиваемые микроконтроллеры (mC);

– процессоры общего назначения (GPP);

– транспьютеры.

Специализированные нейрочипы часто реализуются на основе процессорных матриц (систолических процессоров). Такие нейрочипы похожи по характеристикам на обычные RISC-процессоры, объединяют в своем составе некоторое число процессорных элементов, а управляющая и дополнительная логика, как правило, строится на базе дополнительных схем.

Различают также *нейросигнальные процессоры*, ядро которых представляет собой типовой ПЦОС, а реализованная на кристалле дополнительная логика обеспечивает выполнение характерных нейросетевых операций (например, дополнительный векторный процессор и т. п.).

3.2. Параметры нейропроцессоров

Для оценки производительности нейропроцессоров и нейрокомпьютеров применяются специальные показатели (параметры):

– MMAC – миллионов умножений с накоплением в секунду;

– CUPS (Connections Update per Second) – число измененных значений весов в секунду (оценивает скорость обучения);

– CPS (Connections per Second) – число соединений (умножений с накоплением) в секунду (оценивает производительность);

– CPSPW = CPS/Nw, где Nw – число синапсов в нейроне;

– CPPS – число соединений примитивов в секунду:

$$CPPS = CPS \cdot Bw \cdot Bs,$$

где Bw, Bs – разрядность чисел, отведенных под веса и синапсы.

Для оценки же производительности устройств (реализованных на основе ПЦОС и ПЛИС), применяемых для решения задач управления и прогнозирования, контролируется время выполнения типовых операций ЦОС и др.

Проведенный анализ различных источников по НС позволил получить следующие обобщенные оценки современных возможностей аппаратной реализации НС:

– число моделируемых нейронов – до 5 млн;

– число моделируемых связей – до 5 млн;

– скорость моделирования – до 500 млн переключений связей/с.

3.3. Специализированные нейрочипы

Основное отличие нейрочипов от других процессоров – это обеспечение высокого параллелизма вычислений за счет применения специализированного нейросетевого логического базиса или конкретных архитектурных решений, что и обеспечивает резкое увеличение производительности нейрочипов.

Приведем классификацию существующих нейропроцессоров:

– *по типу логики* нейропроцессоры разделяют на цифровые, аналоговые и гибридные;

– *по типу реализации нейросетевых алгоритмов*: с полностью аппаратной реализаций и с программно-аппаратной реализацией;

– *по характеру реализации нелинейных преобразований*: нейропроцессоры с жесткой структурой нейронов (аппаратно реализованные) и нейрокристаллы с настраиваемой структурой нейронов (перепрограммируемые);

– *по гибкости структуры нейронных сетей*: нейропроцессоры с жесткой и переменной нейросетевой структурой.

Полная классификация нейрочипов приведена на рис. 3.1.

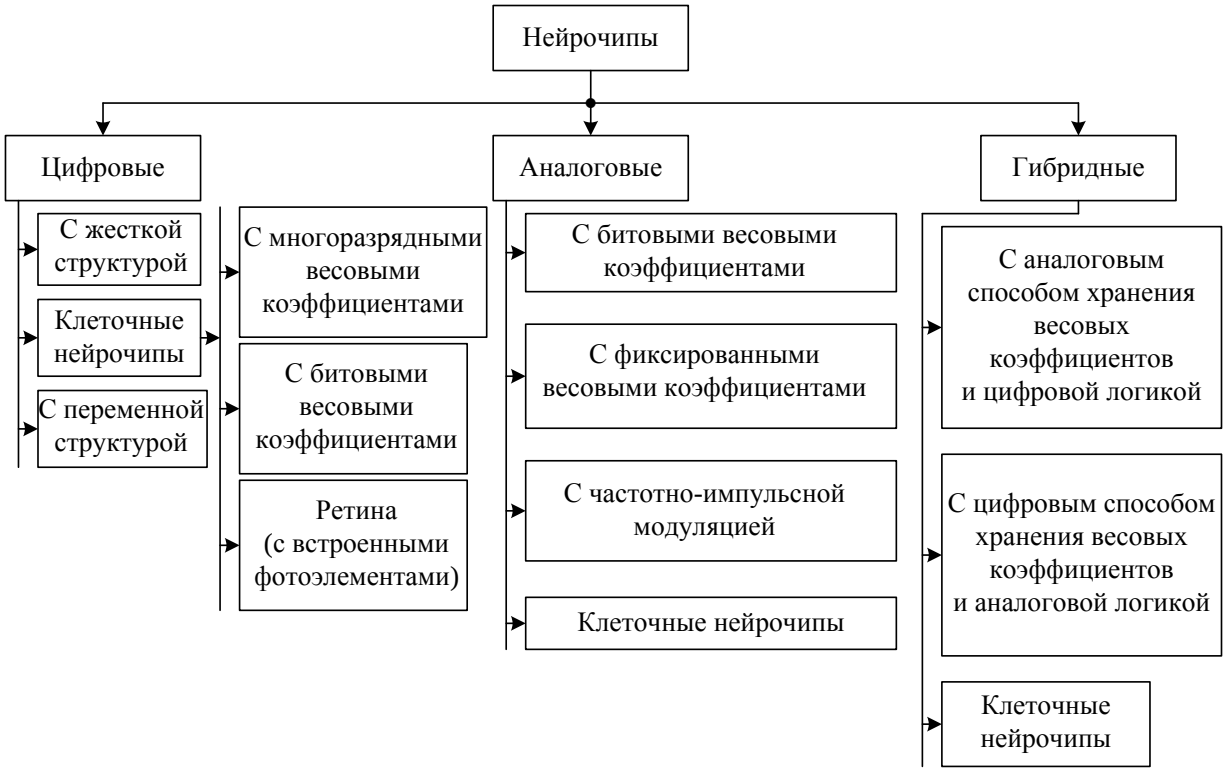

Рис. 3.1. Полная классификация нейрочипов

Производство специализированных нейрочипов ведется во многих странах мира. Большинство из них ориентируются на закрытое использование (так как создаются для конкретных прикладных систем), однако среди нейрочипов достаточно и универсальных кристаллов (табл. 3.1) [18], [24].

Примечания: * – максимальное число синапсов определяет размер внутрикристалльной памяти весов; ** – максимальное число слоев определяется числом операций умножения с накоплением, выполняемых за один такт для операндов длиной 8 бит.

Для аппаратной реализации НС в настоящее время используются следующие технологии:

– распараллеливание нейровычислений на низком уровне;

– ориентирование операционных элементов на выполнение базовой процедуры нейровычислений (умножение чисел с накоплением);

– эффективное использование локальной памяти для хранения параметров НС;

– эффективное использование каналов межпроцессорной связи при передаче сигналов по моделируемой сети;

– балансирование быстродействия операционных элементов, каналов связи и локальной памяти для уменьшения простоев и исключения перегрузки выполнения нейровычислений;

– обеспечивание простоты наращиваемости возможностей по моделированию НС.

Таблица 3.1

Наименование	Фирма изготовитель	Разрядность, бит	Максимальное количество синапсов*/ максимальное число слоев**	Примечание
MA16	Siemens	48 (умножители и сумматоры)	–	400 MMAC
NNP (Neural Networks Processor)	Accurate Automation	N×16		MIMD, N – число процессоров
CNAPS-1064	Adaptive Solutions	16	128 Кбайт/64	–
100 NAP Chip	HNC	32	512 Кбайт/4	4 процессорных элемента
Neuro Matrix NM6403, Такт. частота 50 МГц	Модуль, Россия	64 (вект. процессор), 32 RISC ядро	4096 шт./24	Совместим с портами TMS320C4x
Neuro Matrix NM6404, Такт. частота 133 МГц	Модуль, Россия	64 (вект. процессор), 32 RISC ядро	4096 шт./ ~48	
CLNN 32 CLNN 64	Bellcore	32 64	496/32 нейрона 1024/32 нейрона	10^8 перекл./с 2×10^8 перекл./с
NC 3001	NeuriGam	16	4096 шт./32	
ZISC 036 (Zero Instruction Set Computer)	IBM	64 разр. входного вектора	–/36 нейронов	Частота 20 МГц, векторно-прототипный нейрочип
ETANN 80170NW	Intel	64 входа	Два банка весов 64×80/64 нейрона в слое, 3 слоя	Аналоговая
MD-1220	Micro Devices	16	64 шт./8	8 нейронов
MT 19003 – Neural Instruction Set Processor	Micro Circuit Engineering (MCE)	16 разр. умножитель 35 разр. сумматор	–/1	RISC МП с 7 специальными командами
Neuro Fuzzu	National Semiconductor	–	–	–
NI 1000	Nestor	5–16 (одного нейрона)	–/1024 прототипных 256 мерных векторов	Векторно-прототипный нейрочип

Окончание табл. 3.1

Наименование	Фирма изготовитель	Разрядность, бит	Максимальное количество синапсов*/ максимальное число слоев**	Примечание
NLX420 (NLX 110, 230)	Adaptive Logic	16	1 Мбайт/16	16 процессорных элементов
OBL Chip	Oxford Computer	16	16 Мбайт	–
L-Neuro 1.0 L-Neuro 2.3	Philips		1536/16 нейронов 192 (12×16)	26 МГц 60 МГц
RSC (Speech Recognition Chip) – 164	Sensory Circuits	–	–	–
ORC 110xx (Object Recognizer Chip)	Synaptics			
Pram-256 Chip	UCLi Ltd.	8 (одного нейрона)	–/256 нейронов	33МГц
SAND	Datafactory	16	–/4	200 MCPS
ACC	–		–	–
Геркулес	Россия		1 Мбайт/64	
Neuro Classifier	Университет Твента, DESY	70 вх. нейронов	–/6 (внутр) 1 вх., 1 вых.	2 × 1010 перекл./с
ANNA	AT&T	Число нейронов 16–256	4096 весов	Число входов у нейрона 256-16.
WSC (Wafer Scale Integration)	Hitachi	–	64 связи на нейрон/ 576 нейронов	–
SASLM2	Mitsubishi	2 (одного нейрона)	–/4096(64×64) нейронов	50 МГц
TOTEM	Kent (Univer UK), di Trento (Italy)	16 (одного нейрона)	–/64 нейрона	30 МГц
Neuron 3120, Neurom 3150	Echelon (США)	8 бит (шина данных)	–	Наличие параллельных, последовательных и коммуникационных портов

При этом основными трудностями реализации НС являются:

– полиномиальный рост числа синаптических связей при линейном росте числа нейронов в моделируемой сети;

– большая размерность НС, которые необходимы для решения практически значимых задач обработки сигналов и изображений.

3.4. Программы для моделирования НС

Различают универсальные и прикладные программные продукты для моделирования нейронных сетей (Neural Network Simulators).

Универсальные, или *объектно-инвариантные, среды* позволяют синтезировать оптимальные нейронные сети, применяемые для решения широкого класса задач, с предложением различных парадигм и алгоритмов обучения.

Прикладные среды моделирования ориентированы для синтеза нейронных сетей, применяемых в той или иной отрасли, прикладной области или специфичной задаче.

Среди важнейших свойств нейросетевых симуляторов – способность синтезировать код программы результирующей нейронной сети на алгоритмическом языке высокого уровня (чаще всего – Си и Паскаль). Такой код впоследствии легко интегрировать в пользовательскую программу.

Для реализации НС с помощью современных вычислительных машин применяются программные средства, которые называются нейропакетами. Нейропакет является программной реализацией алгоритмов решения различных задач, в том числе и задач по разработке нейросетевых регуляторов для управления электромеханическими объектами. На рис. 3.2 приведена общая структура нейропакета.

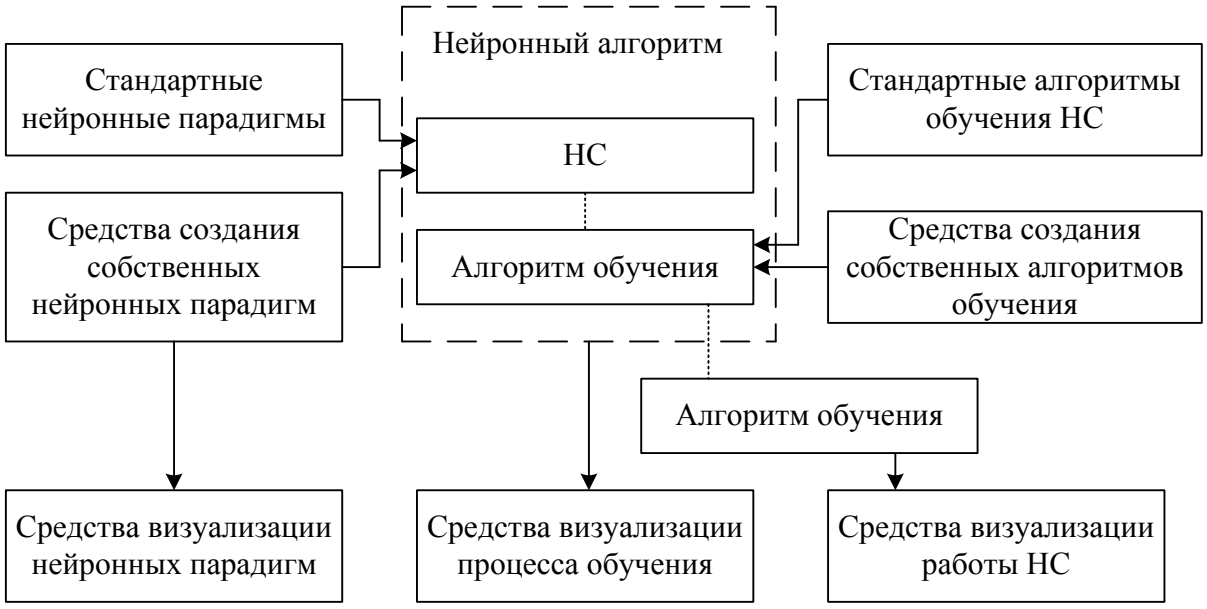

Рис. 3.2. Общая структура нейропакета

Среди них можно отметить средства разработки (библиотеки запрограммированных нейронных парадигм и алгоритмов обучения), универсальные нейропакеты, предоставляющие пользователю работу с несколькими видами нейронных сетей и имеющие средства для создания собственных разработок, специализированные нейропакеты и т. д.

Таким образом, с помощью нейропакетов создаются виртуальные НС, для реализации которых могут быть использованы различные архитектуры параллельных процессоров: матричные процессоры, архитектуры типа «гиперкуб» и вычислители, создаваемые на базе современных транспьютеров и цифровых сигнальных процессоров.

В табл. П1 [18] приложения представлены наиболее распространенные *универсальные* программные среды для моделирования нейронных сетей.

3.5. Использование Simulink при построении нейронных сетей

Пакет Neural Network Toolbox содержит ряд блоков, которые могут быть либо непосредственно использованы для построения нейронных сетей в среде Simulink, либо применяться вместе с рассмотренной функцией gensim.

Для вызова этого набора блоков в командной строке необходимо набрать команду neural, после выполнения которой появляется окно вида рис. 3.3.

Каждый из представленных на рис. 3.3 блоков, в свою очередь, является набором (библиотекой) некоторых блоков.

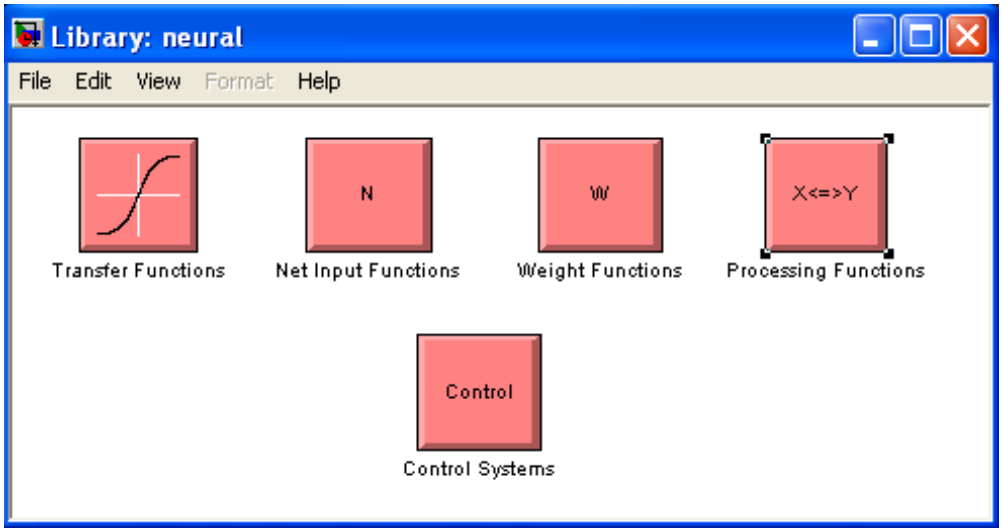

Рис. 3.3. Основные нейросетевые блоки Simulink

1. *Блоки функций активации* (Transfer Functions). Двойной щелчок на блоке Transfer Functions приводит к появлению библиотеки функций активации (рис. 3.4).

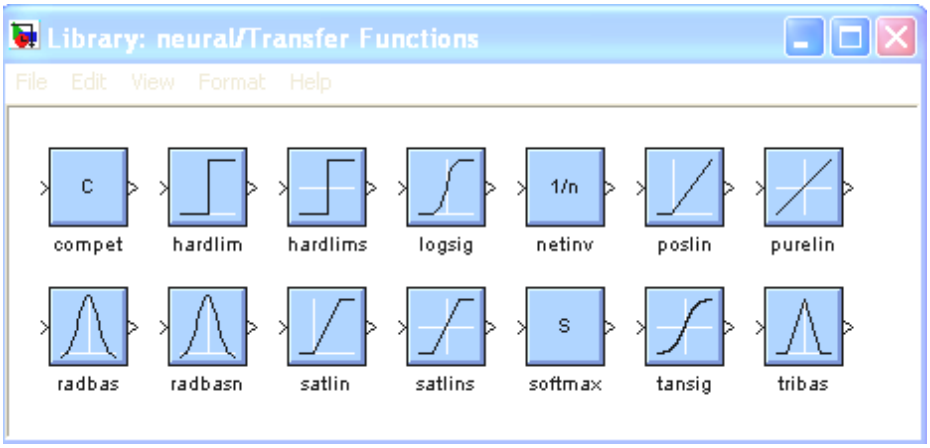

Рис. 3.4. Библиотека функций активации

Каждый из блоков данной библиотеки преобразует подаваемый на него вектор в соответствующий вектор той же размерности.

2. *Блоки преобразования входов сети*. Проведение аналогичной рассмотренной операции, но с блоком Net Input Functions, приведет к появлению блоков, показанных на рис. 3.5.

Блоки данной библиотеки реализуют рассмотренные ранее функции преобразования входов сети.

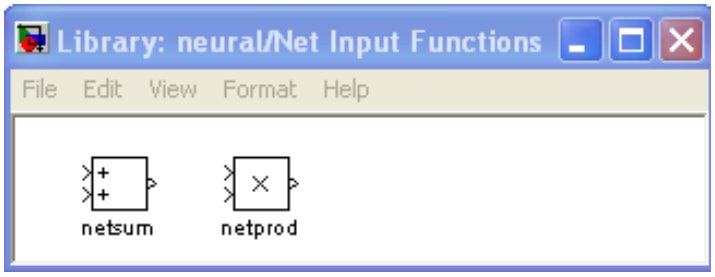

Рис. 3.5. Библиотека блоков
преобразований сигналов

3. *Блоки весовых коэффициентов*. Двойной щелчок на блоке Weight Functions приводит к появлению библиотеки блоков (рис. 3.6), реализующих некоторые функции весов и расстояний.

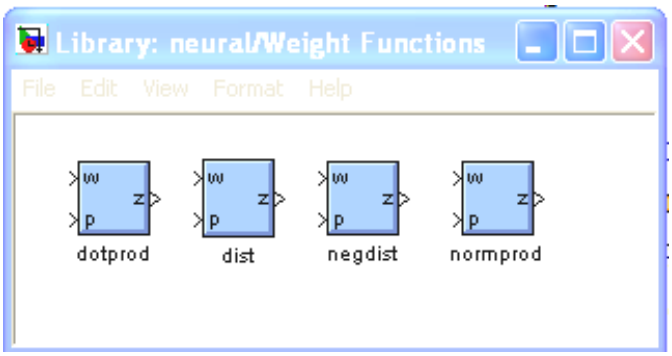

Рис. 3.6. Библиотека блоков весовых коэффициентов

4. *Блоки нейросетевых контроллеров*, применяемых в системах управления. Двойной щелчок на блоке Control Systems приводит к появлению библиотеки блоков (рис. 3.7).

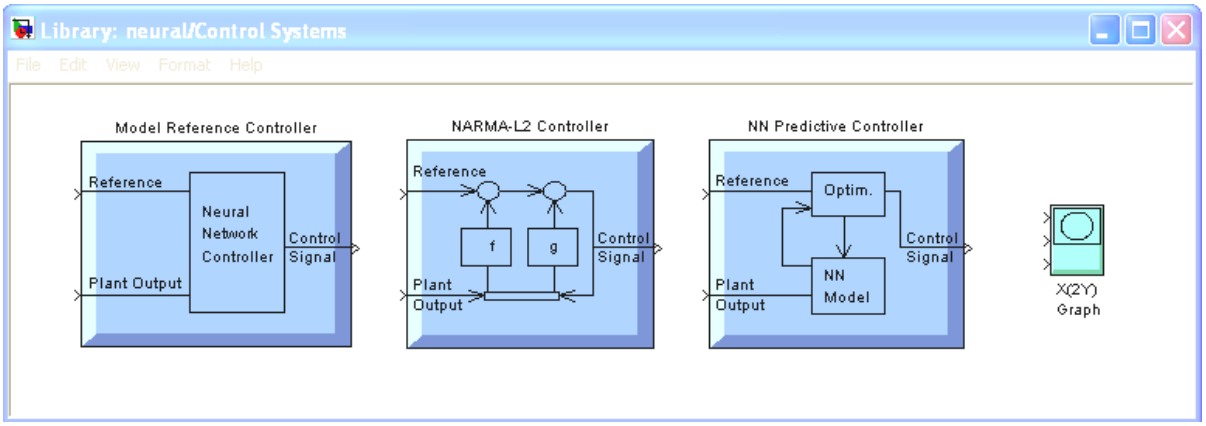

Рис. 3.7. Библиотека блоков Control Systems

Отметим, что в процессе работы со всеми приведенными блоками при задании конкретных числовых значений ввиду особенностей Simulink векторы необходимо представлять как столбцы, а не как строки (как это было до сих пор).

4. СМЫСЛОВАЯ РАЗМЕТКА КОНТЕНТА

Обычно HTML-теги указывают браузеру, как отображать информацию, заключенную в тег. Так, тег *<h2>Кин-дза-дза</h2>* означает, что строку «Кин-дза-дза» следует отображать в формате заголовка второго уровня. Однако HTML-тег не предоставляет никакой информации о смысловом значении этой строки. Это усложняет поисковым системам задачу нахождения информации, адекватной запросу пользователя.

Общедоступный словарь schema.org, с помощью которого вебмастер может размечать страницы, позволяет сделать их понятными самым распространенным поисковым системам: Яндексу, Google, Microsoft и Yahoo.

Словарь *schema.org* применяется вместе с микроданными (формат *microdata*), именно о них и пойдет речь в данном разделе.

4.1. Разметка контента с помощью микроданных

Содержание страниц сайта понятно читающим их людям, однако поисковым системам сложно определить, о чем идет речь. Добавляя специальные теги к HTML-коду страниц, можно «подсказать» поисковой системе, что в конкретном контенте описывается такой-то фильм (место, человек, событие). Тем самым можно обеспечить поисковым системам и другим приложениям лучше понимать контент и отображать его подходящим образом. Именно для этой цели в HTML5 были предложены микроданные – специальный набор тегов.

Тэги itemscope и itemtype – сущность и ее тип

Рассмотрим пример. Пусть у нас есть страница о фильме «Кин-дза-дза» — со ссылкой на трейлер, информацией о режиссере и т. п. HTML-код может выглядеть примерно так:

```
<div>
 <h2>Кин-дза-дза</h2>
 <span>Режиссер: Георгий Данелия (род. 25 августа 1930, Тбилиси)</span>
 <span>Фантастика</span>
 <a href="http://kukindzadza.com/">Трейлер</a>
</div>
```

В первую очередь необходимо указать, какая часть страницы посвящена непосредственно фильму «Кин-дза-дза». Для этого добавим атрибут itemscope к HTML-тегу, в который заключена эта информация:

```
<div itemscope>
  <h1>Кин-дза-дза</h1>
  <span>Режиссер: Георгий Данелия (род. 25 августа 1930, Тбилиси) </span>
  <span>Фантастика</span>
  <a href="http://kukindzadza.com/">Трейлер</a>
</div>
```

Добавление itemscope, означает, что HTML-код, содержащийся в блоке ... , описывает некоторую сущность. Чтобы указать тип сущности, добавим атрибут itemtype сразу после itemscope:

```
<div itemscope itemtype="http://schema.org/Movie">
  <h1>Кин-дза-дза</h1>
  <span>Режиссер: Георгий Данелия (род. 25 августа 1930, Тбилиси)</span>
  <span>Фантастика</span>
  <a href="http://kukindzadza.com/">Трейлер</a>
</div>
```

Тем самым мы уточняем, что сущность, описание которой заключено в теге * *, представляет собой фильм (тип *Movie* в иерархии типов *schema.org*). Названия типов имеют вид URL, в нашем случае *http://schema.org/Movie*.

Тэг itemprop – свойства сущности

О фильме можно сообщить множество интересных сведений: актерский состав, режиссер, рейтинг. Чтобы отметить свойства сущности, используется атрибут *itemprop*. Например, чтобы указать режиссера фильма, добавим атрибут *itemprop=«director»* к HTML-тегу, содержащему имя режиссера. Полный список свойств, которые можно задать для фильма, приведен в [39]:

```
<div itemscope itemtype="http://schema.org/Movie">
  <h1 itemprop="name">Кин-дза-дза</h1>
  <span>Режиссер: <span itemprop="director">Георгий Данелия</span> (род. 25 августа 1930, Тбилиси)</span>
  <span itemprop="genre">Фантастика</span>
  <a href="http://kukindzadza.com/" itemprop="trailer">Трейлер</a>
</div>
```

Обратите внимание, что мы добавили дополнительный тег *... *, чтобы привязать атрибут *itemprop* к соответствующему тексту на странице. Тег ** не влияет на отображение страницы в браузере, поэтому его удобно использовать вместе с itemprop.

Теперь поисковые системы смогут понять не только то, что *http://kukindzadza.com/* – это ссылка, но и то, что это ссылка на трейлер фантастического фильма «Кин-дза-дза» режиссера Георгия Данелия.

Вложенные сущности

Иногда значение свойства может само являться сущностью, с собственным набором свойств. Например, режиссер фильма может быть описан как сущность с типом *Person*, у которой есть свойства *name* (имя), *birthDate* (дата рождения) и *birthPlace* (место рождения). Чтобы указать, что значение свойства представляет собой сущность, необходимо добавить атрибут *itemscope* сразу после соответствующего *itemprop*:

```
<div itemscope itemtype="http://schema.org/Movie">
  <h1 itemprop="name">Кин-дза-дза</h1>
      <div itemprop="director" itemscope itemtype="http://schema.org/Person">
  Режиссер: <span itemprop="name"> Георгий Данелия </span> (род.
<span itemprop="birthDate">16 августа 1954 г.</span>
<span itemprop="birthPlace">Тбилиси </span>)
  </div>
  <span itemprop="genre">Фантастика</span>
  <a href="http://kukindzadza.com/" itemprop="trailer">Трейлер</a>
</div>
```

В последнем примере жирным шрифтом выделены все свойства, значения которых определяют дополнительные сведения о фильме, доступные поисковым системам.

Типы и свойства schema.org

Кроме типов *Movie* и *Person*, schema.org описывает множество разнообразных типов сущностей, для каждого из которых определен набор свойств.

Наиболее обобщенный тип сущности – это *Thing* (нечто), у которого есть четыре свойства: *name* (название), *description* (описание), *url* (ссылка) и *image* (изображение). Более специализированные, частные типы имеют общие свойства с более универсальными. Например, *Place* (место) – частный случай *Thing*, а *LocalBusiness* (местная фирма) – частный случай *Place*. Частные типы наследуют свойства родительского типа (более того, тип *LocalBusiness* является и частным случаем *Place*, и частным случаем *Organization*, поэтому наследует свойства обоих родительских типов.)

Ниже приведен список некоторых популярных типов сущностей:

– творческие произведения: *CreativeWork* (творческое произведение), *Book* (книга), *Movie* (фильм), *MusicRecording* (музыкальная запись), *Recipe* (рецепт), *TVSeries* (телесериал);

– встроенные нетекстовые объекты: *AudioObject* (аудио), *ImageObject* (изображение), *VideoObject* (видео);

– *Event* (событие);

– *Organization* (организация);

– *Person* (человек);

– *Place* (место), *LocalBusiness* (местная фирма), *Restaurant* (ресторан);

– *Product* (продукт), *Offer* (предложение), *AggregateOffer* (сводное предложение);

– *Review* (отзыв), *AggregateRating* (сводный рейтинг).

Полный список сущностей можно найти в [6], [39].

Разметка и валидаторы

При размеке страницы с помощью *schema.org*, обычно придерживаются следующих рекомендаций:

– чем больше контента размечено, тем лучше. Однако, как правило, следует размечать только контент, видимый посетителям сайта, но не содержимое скрытых тегов * * и других скрытых элементов страницы;

– просматривая типы *schema.org*, можно заметить, что у многих свойств есть так называемые ожидаемые типы. Это значит, что значение свойства может быть вложенной сущностью. Однако добавлять вложенную сущность не обязательно: приемлемо использовать просто текст или *URL*. Также вместо ожидаемого типа можно использовать дочерний тип. Например, если для свойства указан ожидаемый тип *Place*, можно добавить вложенную сущность с типом *LocalBusiness*;

– многие страницы посвящены целому ряду сущностей. Например, на сайте компании есть список сотрудников, со ссылками на профиль каждого из них. Для таких агрегирующих страниц нужно разметить отдельно каждую сущность (в этом случае получится последовательность сущностей с типом *Person*) и добавить свойство *url* в ссылку на соответствующую страницу для каждой сущности, например:

```
<div itemscope itemtype="http://schema.org/Person">
  <a href="/alice.html" itemprop="url">Элис Джонс</a>
</div>
<div itemscope itemtype="http://schema.org/Person">
  <a href="/bob.html" itemprop="url">Боб Смит</a>
</div>
```

Подобно тому, как браузер необходим для проверки изменений в верстке веб-страницы, а компилятор — для тестирования кода, разметку *schema.org* также следует тестировать.

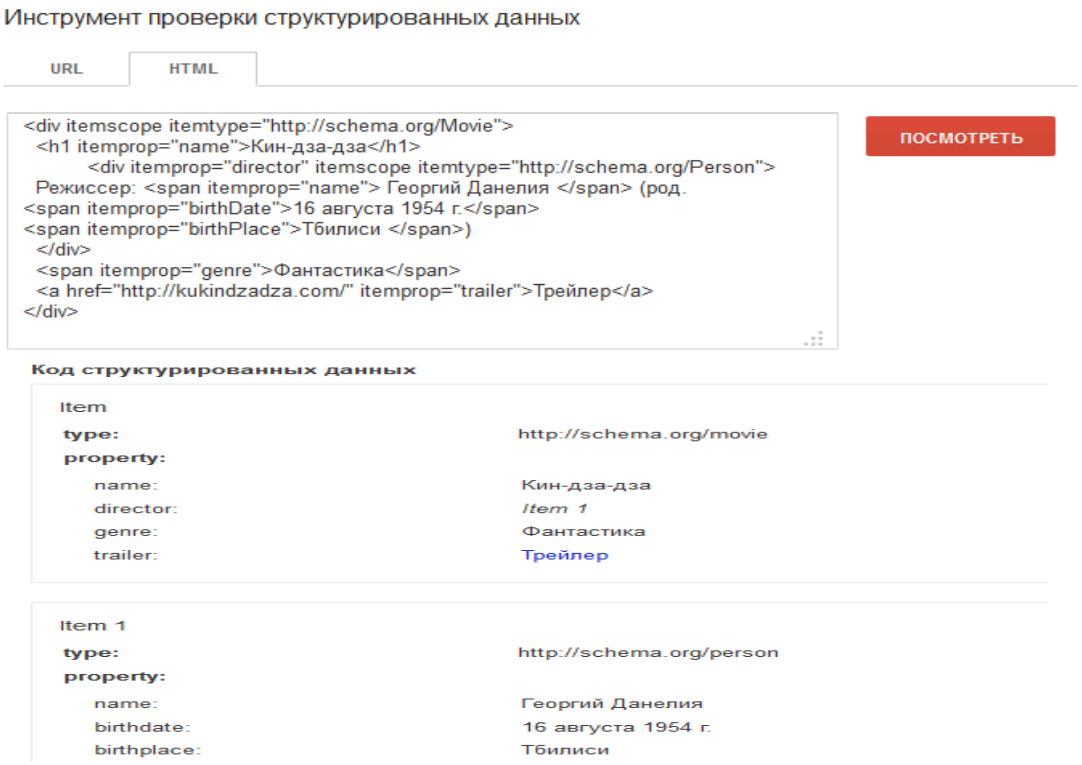

Рис. 4.1 Пример работы валидатора Google

В Интернете существует несколько инструментов-валидаторов, с помощью которых можно проверить корректность разметки и выявить ошибки, например, от Яндекса [9] и от Google [10] (рис. 4.1).

4.2. Машиночитаемая версия информации

Многие страницы можно разметить, используя только атрибуты *itemscope*, *itemtype* и *itemprop* вместе с типами и свойствами *schema.org* (описанными выше). Однако в некоторых случаях роботу сложно однозначно интерпретировать свойство сущности. Этот раздел описывает, как добавить машиночитаемую версию информации в разметку страниц.

Даты и время

Не всегда роботам удается правильно понять дату и время. Например, «04/01/11» – это 11 января 2004 г., 4 января 2011 г. или 1 апреля 2011 г.? Чтобы однозначно задать дату, используйте тег <time> вместе с атрибутом datetime. Значение атрибута datetime – дата в формате YYYY-MM-DD. Следующий HTML-код точно определяет дату как 1 апреля 2011 г.:

```
<time datetime="2011-04-01">04/01/11</time>
```

Время суток задается в формате hh:mm либо hh:mm:ss, с префиксом T. Время может быть указано вместе с датой:

```
<time datetime="2011-05-08T19:30">8 мая, 19:30</time>
```

Рассмотрим пример в контексте: HTML-код с информацией о концерте, который состоится 8 мая 2011 г. Разметка типа Event включает название события, описание и дату.

```
<div itemscope itemtype="http://schema.org/Event">
  <div  itemprop="name">Spinal Tap</div>
  <span  itemprop="description">Одна из самых громких музыкальных групп всех времен воссоединится для незабываемого двухдневного шоу.</span> Концерт состоится
  <time itemprop="startDate" datetime="2011-05-08T19:30">8 мая в  19:30</time>
</div>
```

Длительность описывается аналогичным образом, с помощью тега <time> и атрибута datetime. Значение атрибута предваряется префиксом P (от англ. period). Например, указать в рецепте, что время приготовления составляет полтора часа, можно следующим образом:

```
<time itemprop="cookTime" datetime="PT1H30M">1 ½ часа</time>,
```

где H обозначает количество часов, а M — количество минут. Форматы даты, времени и длительности соответствуют стандарту ISO 8601.

Перечисления

Некоторые свойства имеют ограниченный набор возможных значений. Программисты называют их перечислениями. Например, интернет-магазин может использовать тип сущности *Offer* для подробного описания товарного предложения. Свойство *availability* (наличие) обычно имеет одно из нескольких возможных значений: *In stock* (в наличии), *Out of stock* (отсутствует), *Pre-order* (на заказ) и т. д. Значения перечисляемых свойств можно указывать в виде URL, аналогично типам сущностей *schema.org*.

Вот товарное предложение, размеченное как сущность типа *Offer*, с соответствующими свойствами:

```
<div itemscope itemtype="http://schema.org/Offer">
  <span itemprop="name">Blend-O-Matic</span>
  <span itemprop="price">$19.95</span>
  <span itemprop="availability">Уже в продаже!</span>
</div>
```

А вот то же самое предложение, размеченное с использованием <link> и href, что позволяет однозначно указать одно из допустимых значений свойства availability:

```
<div itemscope itemtype="http://schema.org/Offer">
  <span itemprop="name">Blend-O-Matic</span>
  <span itemprop="price">$19.95</span>
  <link itemprop="availability" href="http://ruschema.org/InStock"/>Уже в продаже!</span>
</div>
```

Schema.org предоставляет список возможных значений для небольшого количества свойств – в тех случаях, когда у свойства есть ограниченный набор типичных значений. Так, возможные значения для свойства *availability* перечислены в [39].

Ссылки на канонические страницы

Обычно ссылки создаются с помощью тега <a>. Например, следующая гиперссылка на страницу в Википедии для книги «Над пропастью во ржи»:

```
<div itemscope itemtype="http://schema.org/Book">
  <span itemprop="name">Над пропастью во ржи</span>
  Автор <span itemprop="author">Джером Сэлинджер</a>
  <a itemprop="url" href="http://en.wikipedia.org/wiki/The_Catcher_in_the_Rye">Страница в Википедии</a>
</div>
```

Как видите, атрибут itemprop=«url» можно использовать для ссылки на страницу другого сайта (в этом случае, Википедии), которая описывает ту же самую сущность. Ссылки на сторонние сайты помогают поисковым системам лучше понимать, о чем речь на вашей странице.

Если вы не желаете добавлять ссылку, видимую посетителям, используйте тег <link>, как показано ниже:

```
<div itemscope itemtype="http://schema.org/Book">
  <span itemprop="name">Над пропастью во ржи</span>
  <link itemprop="url" href="http://en.wikipedia.org/wiki/The_Catcher_in_the_Rye" />
  Автор <span itemprop="author">Джером Сэлинджер</span>
</div>
```

Неявная информация

Иногда важная информация не может быть размечена из-за способа ее отображения на странице. Например, информация представлена на рисунке (изображение рейтинга 4 звезды из 5) или во Flash-объекте (например, длительность видеоролика) либо она подразумевается, но не указана на странице в явном виде (например, валюта цены).

В таких случаях можно использовать тег <meta> с атрибутом content. В следующем примере картинка иллюстрирует рейтинг 4 звезды из 5:

```
<div itemscope itemtype="http://schema.org/Offer">
  <span itemprop="name">Blend-O-Matic</span>
  <span itemprop="price">$19.95</span>
  <img src="/four-stars.jpg" />
  25 оценок
</div>
```

А вот тот же пример с размеченной информацией о рейтинге.

```
<div itemscope itemtype="http://schema.org/Offer">
  <span itemprop="name">Blend-O-Matic</span>
  <span itemprop="price">$19.95</span>
  <div itemprop="reviews" itemscope itemtype="http://schema.org/AggregateRating">
    <img src="/four-stars.jpg" />
    <meta itemprop="ratingValue" content="4" />
    <meta itemprop="bestRating" content="5" />
    <span itemprop="ratingCount">25</span> оценок
  </div>
</div>
```

Этим приемом не следует злоупотреблять. Используйте тег <meta> с атрибутом content только для той информации, которую невозможно разметить иным способом.

Расширение *schema.org*

Большинству сайтов и организаций не потребуется расширять *schema.org*. Однако ясно, что разнообразие данных слишком велико, чтобы какая-либо одна схема (типа *schema.org*) могла удовлетворить всем потребностям в структуризации этих данных. В этой связи schema.org предусматривает возможность добавлять свойства и дочерние типы для уже имеющихся типов сущностей. Все подобные расширения, которые будут приняты большим числом заинтересованных лиц, могут быть добавлены в ядро *schema.org*, увеличивая тем самым его словарный запас и позволяя обеспечить поисковым системам большую функциональность, основанную на еще более лучшем понимании структурированных данных.

При расширении *schema.org* следует придерживаться определенных правил именования всех свойств и дочерних типов [http://ruschema.org/docs/extension]. Причем названия типов и расширений должны начинаться с заглавной буквы (например, KindleBook), а названия свойств с маленькой буквы (например, leadVocalist) – те и другие пишутся слитно в стиле *CamelCase*. Для ввода расширения используется косая черта, после которой следует имя расширения. Так, для расширения типа используется запись в виде:

Person/Engineer

Для образования нового типа из уже существующего можно продолжить эту запись, в виде:

Person/Engineer/ElectricalEngineer,

а так можно расширить свойства:

musicGroupMember/leadVocalist
musicGroupMember/leadGuitar1
musicGroupMember/leadGuitar2.

Если содержание разрабатываемого домена вообще не связано с типами сущностей из *schema.org*, то, конечно, придется прибегнуть к созданию новых схем. Естественно, если эти новые схемы окажутся полезными, то поисковые системы также смогут начать их использовать.

5. RDFA – НАБОР ДАННЫХ ДЛЯ ПОИСКОВЫХ МАШИН

За последние несколько лет нам довелось стать свидетелями увлекательной эволюции. Всемирная паутина однозначно была создана в целях удовлетворения нужд человечества, но веб-контент также обрабатывается значительным количеством машин, которые ожидают от него наличия более структурированных данных. Поисковые системы научились предоставлять более детальные результаты поиска на основе расширенных данных содержащихся на страницах обрабатываемых роботом.

Ключевыми технологиями сделавшими реальными эти новшества являются HTML и RDFa (Resource Description Framework in attributes). Совокупность этих технологий сделала возможным оснастить веб-страницы дополнительными данными для улучшенной восприимчивости машинами. В этом разделе рассматриваются методы использования формата RDFa для выделения на HTML-страницах данных, важных для пользователей, чтобы они стали доступными и для машин, т. е. чтобы машины готовили их для пользователей.

5.1. Назначение RDFa

Современный web полон связанных между собой хранилищ информации. До недавнего времени организация подобная этой предусматривала исключительно удобство пользования человеком. На типичной web-странице разработчик HTML мог указать заголовок, подзаголовок, маленький блок с курсивным начертанием текста, несколько параграфов произвольного размера и, наконец, несколько ссылок. Конечно браузеры будут следовать этой разметке беспрекословно, однако, только человеческий разум способен понять, что шапка статьи размеченной подобным образом обозначает заголовок поста блога. Содержимое под заголовком описывает автора (рис. 5.1), курсивный текст дату публикации заметки, а несколько ссылок отражают рубрики, к которым относится содержимое.

Рис. 5.1. Слева – это видит браузер, справа – это видит человек

К сожалению, компьютеры лишены возможности различать нюансы в контексте контента; разница между человеческим и машинным восприятием огромная.

RDFa позволяет разработчикам HTML дополнить данные для людей машиночитаемыми индикаторами для браузеров и других программ для понимания. Веб-страницы могут содержать разметку такую простую, как заголовок статьи или такую сложную, как комплексная информация для социальных сетей.

Исторически RDFa специфицирован только для XHTML. Несмотря на это, он одинаково хорошо подходит для HTML4 и HTML5. Для простоты будет использован термин «HTML» для обозначения всех языков семейства HTML.

RDFa основывается на атрибутах. Хотя HTML-атрибуты (например, href, rel) будут использованы для новых целей, некоторые в спецификации RDFa являются новыми. Это имеет значение поскольку некоторые из интерпретаторов HTML или XHTML не смогут идентифицировать HTML-код после его обновления для совместимости с новшествами RDFa. Это не является проблемой, поскольку большинство браузеров просто игнорируют незнакомые атрибуты. Ни один из обозначенных в RDFa атрибутов не дает побочных эффектов на визуальное представление.

Рассмотрим способы добавления машиночитаемых подсказок на web-страницы на следующем примере.

Пусть Алиса – блоггер, публикующий смешанные типы содержимого, например, заметки персонального и профессионального характера, на страничках по адресу http://example.com/alice. Рассмотрим примеры разметки, иллюстрирующие, как Алиса может применить RDFa.

Подсказки для сайтов социальных сетей

Алиса установила на свой блог плагин предоставляющий возможность поместить под каждой публикацией кнопку «Like». Эта кнопка должна позволить информации о характере содержимого на блоге Алисы (заголовки, превью, типы содержимого) стать машино-читаемым и легко переносимым в социальную сеть. Для того, чтобы это стало возможным она использует Open Graph Protocol (OGP) [46] для разметки своего контента. OGP, по своей сути, является обычным RDFa:

```
<html prefix="og: http://ogp.me/ns#">
 <head>
  <title>Проблемы с Бобом</title>
  <meta property="og:title" content="Проблемы с Бобом" />
  <meta property="og:type" content="text" />
  <meta property="og:image" content="http://example.com/alice/bob-ugly.jpg" />
</head>
```

Все «подсказки» для поисковых машин, сделанные с помощью атрибутов RDFa, выделены курсивом. Первое утверждение, *prefix=«og:http://*

ogp.me/ns#», определяет базовый словарь OGP. Этот словарь используется для атрибутов property чтобы указать на свойства самого документа или содержащихся в этих атрибутах данных. Один документ может содержать несколько таких словарей. Атрибут *content* применяется для определения значения свойства.

Обозначение заголовка и автора

Как только Алиса добавила соответствующие мета-данные формата OGP, она обнаружила, что заголовок ее странички уже есть в видимом содержимом разметки:

```
<div>
  <h2>Проблемы с Бобом</h2>
  <h3>Алиса</h3>
</div>
```

Алиса также может использовать RDFa атрибут *property* в элементе HTML h2 чтобы пометить отображаемый браузером текст машиночитаемым параметром указывающим на то, что содержимое тега является заголовком страницы:

```
<div prefix="og: http://ogp.me/ns#">
  <h2 property="og:title">Проблемы с Бобом</h2>
  <h3>Алиса</h3>
</div>
```

Отметим, что в приведенном примере у Алисы нет нужды использовать атрибут *content*, так как он будет повторять уже существующую информацию.

Алиса также обращает внимание на то, что неплохо было бы разметить информацию об авторе. Словарь OGP не содержит способов описания авторства, но это позволяет сделать словарь дублинского ядра (Dublin Core) [37]. Применение RDFa подразумевает легкое смешение различных словарей при разметке содержимого:

```
<div prefix="og: http://ogp.me/ns# dc: http://purl.org/dc/terms/">
  <h2 property="og:title">Проблемы с Бобом</h2>
  <h3 property="dc:creator">Алиса</h3>
</div>
```

Обратим внимание на то, что RDFa, являясь базирующейся на RDF технологией, использует URL для определения фактически всего что угодно. Именно поэтому для обозначения свойств наподобие *creator* и *title*, используются dc:*creator* и og:*title*. Таким образом, каждый префикс содержит связку с URL своего словаря – http://purl.org/dc/terms/creator и http://ogp.me/ns#title соответственно.

Причина такого дизайнерского решения уходит корнями в стремление сделать данные портативными, совместимыми и легко распространяемыми.

Таким образом проектировщики RDF желали устранить возможные неясности в терминологии RDFa. Без обеспечения атрибутам уникальности один и тот же термин мог бы означать различные по своей сути понятия. Например, «title» можно было бы интерпретировать как «название работы», «должность» или «акт покупки недвижимости». Когда каждый термин словаря связан с URL, подробное разъяснение для каждого термина доступно в пределах одного клика мышью. Это позволяет и человеку, и машине проследовать по ссылке и узнать, что именно подразумевает данный термин словаря.

Используя URL-адрес для обозначения конкретного типа заголовка, к примеру http://ogp.me/ns#title, большая часть людей и машин сможет понять, что URL однозначно указывает на заголовок документа, такого как web-страница. URL http://purl.org/dc/terms/creator индицирует о том, что термин словаря указывает на «на сущность ответственную за создание ресурса».

Использование URL-адресов в качестве идентификаторов в RDFa гарантирует избежание неясностей и двузначностей терминов словарей.

В итоге у Алисы получилось:

1) импортировать словари Dublin Core и OGP, используя атрибут prefix, где префиксы dc и og указывают на соответствующие словари;

2) использовать dc:creator и og:title являющиеся краткими записями соответствующих терминов словарей http://purl.org/dc/creator/creator и http://ogp.me/ns#title.

Существует один удобный способ для визуализации структурных данных (рис. 5.2).

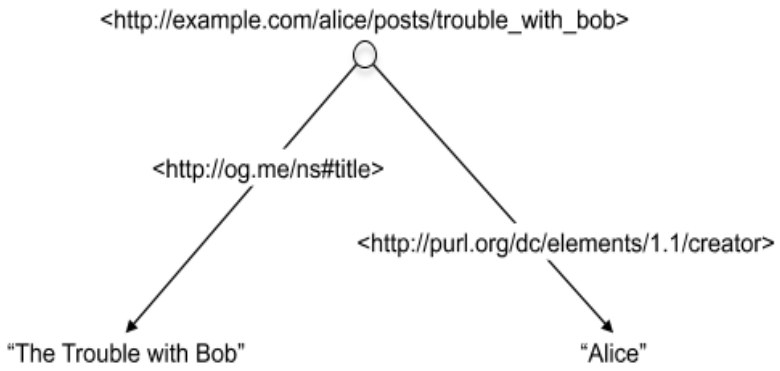

Рис. 5.2. Визуализация структурных данных
для HTML блога с заголовком «Проблемы с Бобом»
и автором назвавшим себя «Алиса»

Реляционные ссылки

В конце своего блога Алиса пожелала указать, что ее контент может свободно использоваться в соответствии с лицензией Creative Commons [33]:

```
Вся информация на этом сайте попадает под действие
  <a href="http://creativecommons.org/licenses/by/3.0/">
    лицензии Creative Commons
  </a>.
```

Любой человек ясно осознает смысл этих строк. Но, к сожалению, когда Боб посещает блог Алисы его браузер видит лишь обычную ссылку, которая ничем не отличается от ссылки на блог одного из друзей Алисы. Чтобы браузер Боба смог осмыслить ссылку как нормативный индикатор, Алиса должна добавить некоторые *отношения* к ссылке. Она может указать эти намеки используя атрибут rel определяющий *отношения* текущей страницы к странице, связанной ссылкой. В данном случае значение атрибута *rel* license, это ключевое слово специально зарезервированное в HTML:

```
Вся информация на этом сайте попадает под действие
  <a rel="license" href="http://creativecommons.org/licenses/by/B.0/">
    лицензии Creative Commons
  </a>.
```

При помощи этого маленького дополнения браузер Боба теперь сможет понять, что отношение ссылки указывает на нормативную лицензию относящуюся к содержимому блога Алисы. Отметим, что эти зарезервированные ключевые слова имеют URI сноски определенные также как в RDFa в словаре http://www.w3.org/1999/xhtml/vocab# (рис. 5.3).

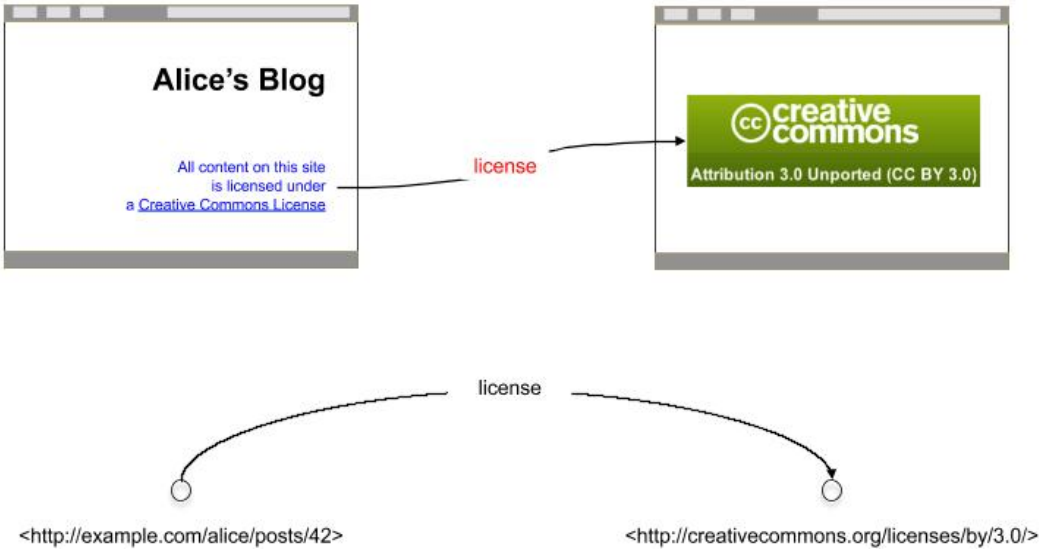

Рис. 5.3. Ссылка индицирует лицензию,
распространяющуюся на данную web-страницу

Для того, чтобы с помощью OGP обозначить превью для изображений RDFa использует тот же атрибут rel *в пределах* image элемента:

```
<div rel="og:thumbnail"> <img src="alice.jpg" /> </div>
```

Таким образом Алиса смогла добавить к содержимому своего блога мета-разметку на основе RDFa ни разу не повторив содержимое текста и ни разу не продублировав URI активных ссылок.

Повторяющихся блоки данных на страницах

Блог Алисы содержит множественные записи. Иногда на блог Алисы заглядывает сестра Ева. На главной странице блога представлены 10 последних записей, каждая из которых имеет собственное название, автора и вступительный абзац. Как в таком случае Алисе разметить название каждой записи, даже если они отображаются на одной и той же Web-странице? Для этих целей в RDFa есть атрибут about. Данный атрибут позволяет задать конкретный URL, к которому применяется разметка вложенного RDFa:

```
<div prefix="dc: http://purl.org/dc/elements/1.1/ og: http://ogp.me/ns#">
    <div about="/alice/posts/trouble_with_bob">
    <h2 property="og:title">Проблема с Бобом</h2>
    <h3 property="dc:creator">Алиса</h3>
  </div>
   <div about="/alice/posts/jos_barbecue">
    <h2 property="og:title">Барбекю у Джо</h2>
    <h3 property="dc:creator">Ева</h3>
  </div>
</div>
```

Можно выразить это в виде схемы соединения URL адресов со свойствами, представленными на рис. 5.4.

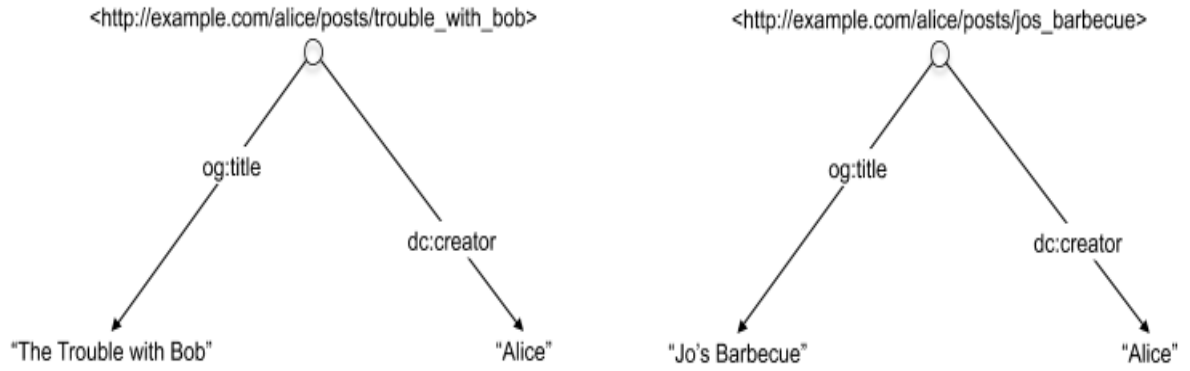

Рис. 5.4. Повторяющихся блоки данных на страницах

Алиса вполне может использовать тот же способ при публикации фотографий ее друга Боба:

```
<div about="/alice/posts/trouble_with_bob">
   <h2 property="og:title">Проблема с Бобом</h2>
   Самая главная проблема с Бобом в том, что он фотографирует гораздо лучше, чем я:
       <div about="http://example.com/bob/photos/drugs_and_bird.jpg">
    <img src="http://example.com/bob/photos/drugs_and_bird.jpg" />
    <span property="og:title">Упоротый пингвин</span>
    by <span property="dc:creator">Боб</span>.
   </div>
 </div>
```

Обратите внимание на вложенное значение about, соответствующее ссылке http://example.com/bob/photos/drugs_and_bird.jpg. Оно «перекрывает» внешнее значение /alice/posts/trouble_with_bob при разметке внутри вложенного тега div. На рис. 5.5 приведена схема, на которой абстрактно показаны данные, лежащие в основе этого нового фрагмента разметки.

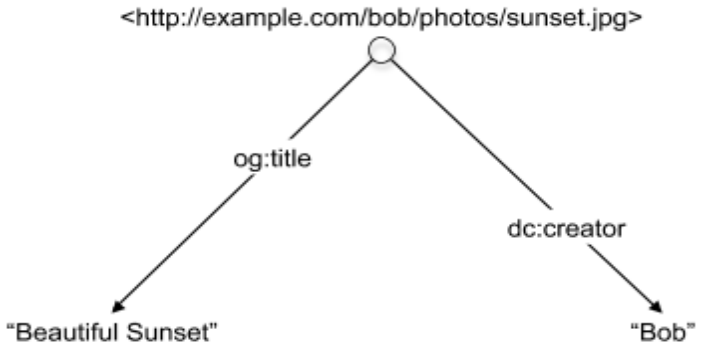

Рис.5.5. Описание фотографии

5.2. Упрощение разметки контента

RDFa располагает рядом методов позволяющих значительно упростить разметку контента. Рассмотрим три метода, упрощающие разметку при использовании нескольких словарей.

Установка пометки о словаре по умолчанию

Часто в случае необходимости использовать, например OGP, HTML редакторы будут использовать только одиночный словарь. Вместо того, чтобы каждый раз дублировать префикс og: в каждом атрибуте элемента, RDFa позволяет определить атрибут vocab для определения префикса атрибута группе однотипных данных в пределах одного несущего HTML элемента. Таким образом, вместо того, чтобы писать:

```
<html prefix="og: http://ogp.me/ns#">
 <head>
   <title> Проблема с Бобом </title>
   <meta property="og:title" con Проблема с Бобом tent=" " />
   <meta property="og:type" content="text" />
   <meta property="og:image" content="http://example.com/alice/bob-ugly.jpg" />
 </head>
```

имеется возможность поступить так:

```
<html>
 <head vocab="http://ogp.me/ns#">
   <title> Проблема с Бобом </title>
   <meta property="title" content=" Проблема с Бобом " />
   <meta property="type" content="text" />
   <meta property="image" content="http://example.com/alice/bob-ugly.jpg" />
</head>
```

Гораздо удобнее размечать содержимое в теле HTML и повторно использовать уже однажды определенные словари. В RDFa использование атрибута vocab допускается как для head-элемента, так и для body-элемента HTML даже при условии отсутствия атрибута content.

Связки словарей

В процессе разметки своего контента Алиса хочет пометь еще больше данных с помощью RDFa, она может использовать все больше параметров из таких словарей, как Dublic Core [38] и FOAF [35]. Алисе хочется иметь возможность упростить записи для сопоставления различных словарей. Хотя, она хотела бы также избежать употребления префиксов, но атрибут vocab не рассчитан на использование с множественными словарями. Чтобы решить проблемы простоты и согласованности в RDFa применяется идея *комплектаций словарей* в одном профиле. Эта техническая идея позволяет с редакторам с легкостью совместить использование множественных словарей при помощи простых short-hand именований.

Для реализации подобной идеи потребуется опубликовать *профиль RDFa документа*, описывающий карту кратких имен, таких как name указывающих на полные URI такие как http://xmlns.com/foaf/0.1/name. При этом, сама разметка будет выглядеть весьма просто:

```
<div profile="http://example.org/profiles/alice">
   <span property="title"> < Проблема с Бобом /span>
   <span property="name">Алиса</span>
</div>
```

В приведенном примере используется созданный Алисой словарь содержащий для атрибутов «title» и «name» префиксы и связи словарей. Такая разметка подразумевает описание данных с помощью словарей Dunlin Core и FOAF соответственно.

Профили документов могут также определять префиксы. Таким образом автор может использовать профиль документа описывающий карту объявлений префиксов словарей. Это эквивалентно комплексу prefix объявлений в разметке поста блога. Для примера рассмотрим пример:

```
<div prefix="og: http://ogp.me/ns# dc: http://purl.org/dc/terms/">
  <h2 property="og:title"> Проблема с Бобом </h2>
  <h3 property="dc:creator">Алиса</h3>
 </div>
```

В данном случае можно заменить атрибут prefix профилем:

```
<div profile="http://example.org/profiles/prefixes">
  <h2 property="og:title"> Проблема с Бобом </h2>
  <h3 property="dc:creator">Алиса</h3>
</div>
```

Здесь подразумевается, что профиль http://example.org/profiles/prefixes содержит префикс для определения og: и dc:. При замене двух префиксов одним профилем достигается незначительная минификация разметки, но в более сложных примерах данный метод поможет авторам значительно уменьшить результирующий код.

Стандартные профили

Профили позволяют укомплектовать наборы префиксов в одном месте. Однако это не избавляет от потребности явно ссылаться на эти профили непосредственно из документа. Существует достаточно большое число словарей с префиксами активно используемыми web-сообществом. Хорошим примером является словарь дублинского ядра (Dublin Core). Такие словари как правило определяются несколько раз в пределах одного документа, но иногда авторы забывают объявить префиксы.

Для облегчения этой задачи RDFa располагает концептуальной идеей *базовых профилей*. Эти профили, разработанные W3C, *всегда* косвенно ссылаются на любое содержимое RDFa. В первую очередь RDFa процессор загружает именно эти словари для каждой обрабатываемой страницы. Объявления профилей и префиксов пользователем в документе всегда являются приоритетными над базовыми профилями. Если редактор вдруг забудет объявить такие словари как DC или FOAF, RDFa процессор сможет взять значения по умолчанию из базового профиля.

В HTML определены два таких базовых профиля.

1. Базовый профиль для *любого* XML содержимого (например, SVG, Atom и т. д.). URI для такого профиля определено как http://www.w3.org/profile/rdfa-1.1.

2. Базовый профиль для отдельных версий HTML (XHTML, HTML, и т. д.), который загружает *после* базового профиля XML. Его URI определено как http://www.w3.org/profile/html-rdfa-1.1.

Редакторы могут обратиться к профилям за информацией о том, какие префиксы и термины подключаются к документу автоматически. Базовые профили по умолчанию выступают в роли предохранителя позволяющего избежать ошибок, если автор забыл задать основные префиксы. Авторы могут полагаться на то, что в RDFa 1.1 базовые профили будут доступны всегда, но стоит обратить внимание на то, что префиксы могут меняться в течение последующих 5–10 лет. Лучшую уверенность в работоспособности документов может гарантировать только применение соответствующих атрибутов prefix повсеместно.

К примеру, в примере ниже не объявлен префикс для dc: и не задан какой либо профиль с помощью атрибута profile:

```
<div>
  <h2 property="dc:date">2011-03-19</h2>
  <h3 property="dc:creator">Alice</h3>
</div>
```

Однако RDFa процессор будет по прежнему иметь возможность идентифицировать dc:date и dc:creator и связать их с соответствующими URI словарей. RDFa процессор в состоянии сделать это самостоятельно поскольку dc префикс является частью базового профиля http://www.w3.org/profile/rdfa-1.1.

Данная практика является лишь вариантом защиты от случайной ошибки редактора и направлена на помощь начинающим авторам. Частое пренебрежение объявлением словарей крайне не приветствуется.

5.3. Обеспечение доступности информации

Алисе хотелось бы сделать свою персональную информацию, такую как email и номер телефона, более доступной ее друзьям и программному обеспечению. Вместо описания свойств web-страницы Алиса собирается описать свойства субъекта, т. е. себя. Для воплощения задуманного Алисе понадобится более сложная структура разметки, которая соединит множественные элементы.

Контактная информация

На блоге Алисы уже имеется отображаемая контактная информация.

```
<div>
  <p>Алиса Иванова</p>
   <p>Email: <a href="mailto:alice@example.com">alice@example.com</a></p>
   <p>Phone: <a href="tel:">+1 617.555.7332</a></p>
 </div>
```

Словарь дублинского ядра не предоставляет каких либо свойств для описания контактной информации, однако они есть в словаре Friend-of-a-Friend [35]. RDFa позволяет относительно просто комбинировать данные описываемые различными словарями на одной страницу. Алиса импортирует словарь FOAF и определяет foaf:Person. Для этого она использует атрибут RDFa typeof, позволяющий указать новый элемент данных указанного типа:

```
<div prefix="foaf: http://xmlns.com/foaf/0.1/" typeof="foaf:Person">
```

Алиса понимает, что только собирается употребить словарь FOAF в данном месте, поэтому сразу же определяет элементу содержащему RDFa разметку соответствующий атрибут:

```
<div vocab="http://xmlns.com/foaf/0.1/" typeof="Person">
```

Затем Алиса размечает участок содержимого страницы, содержащий информацию о ее полном имени, email и номере телефона:

```
<div vocab="http://xmlns.com/foaf/0.1/" typeof="Person">
  <p property="name">Алиса Иванова</p>
```

```
  <p>Email: <a rel="mbox" href="mailto:alice@example.com">alice@example.com</a></p>
  <p>Phone: <a rel="phone" href="tel:+1-617-555-7332">+1 617.555.7332</a></p>
</div>
```

В данном примере Алиса не определяла атрибут about, как при добавлении метаданных к записям блога. В сущности typeof расположенный внутри открывающего тега div, неявно задает тип свойств, которые размечены внутри этого div. Имя, адрес электронной почты и номер телефона связаны с новым узлом типа foaf:Person. Этот узел не имеет четко определенного URI и называется чистым узлом (blank node).

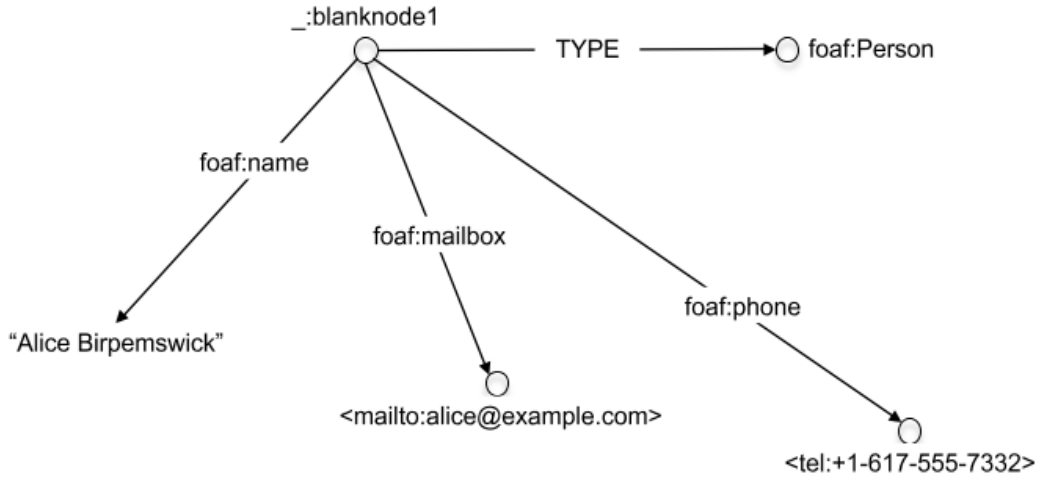

Рис. 5.6. Чистый узел: в blank node URI не определен.

Вместо URI чистые узлы содержат атрибут @typeof, который устанавливает тип представляемых данных. Такой подход (отсутствие имени при наличии типа) особенно полезен при перечислении набора элементов на странице, например, событий в календаре, авторов статьи, приятелей внутри социальной сети и т. д.

Описание социальных сетей

Алиса хочет добавить информацию о своих приятелях, включив хотя бы их имена и адреса главных страниц. Для начала у Алисы есть простой базовый HTML следующего вида:

```
<div>
  <ul><li><a href="http://example.com/bob/">Боб</a></li>
        <li><a href="http://example.com/eve/">Ева</a></li>
        <li><a href="http://example.com/manu/">Маша</a></li>
  </ul>
</div>
```

Сначала Алиса задает тип foaf:Person в атрибуте typeof:

```
<div vocab="http://xmlns.com/foaf/0.1/">
  <ul><li typeof="Person"><a href="http://example.com/bob/">Боб</a></li>
```

```
        <li typeof="Person"><a href="http://example.com/eve/">Ева</a></li>
        <li typeof="Person"><a href="http://example.com/manu/">Маша</a></li>
    </ul>
  </div>
```

После описания типа данных, с которым нам предстоит работать, каждый typeof создает новый чистый узел с индивидуальными свойствами. При этом идентификаторы URI задавать не нужно. Таким образом, Алиса может разметить главные страницы своих приятелей:

```
<div vocab="http://xmlns.com/foaf/0.1/">
  <ul><li typeof="Person"><a rel="homepage" href="http://example.com/bob/">Боб</a></li>
        <li typeof="Person"><a rel="homepage" href="http://example.com/eve/">Ева</a></li>
        <li typeof="Person"><a rel="homepage" href="http://example.com/manu/">Маша</a></li>
  </ul>
</div>,
```

и, конечно, имена приятелей:

```
<div vocab="http://xmlns.com/foaf/0.1/">
  <ul>
    <li typeof="Person">
      <a rel="homepage" href="http://example.com/bob/" property="name">Боб</a>
    </li>
    <li typeof="Person">
      <a rel="homepage" href="http://example.com/eve/" property="name">Ева</a>
    </li>
    <li typeof="Person">
      <a rel="homepage" href="http://example.com/manu/" property="name">Маша</a>
    </li>
  </ul>
</div>
```

С помощью property Алиса определяет, что связанный текст («Боб», «Ева» и «Маша»), по сути, обозначает имена ее приятелей. Атрибут rel позволяет указать, что ссылки позволяют перейти на их главные страницы. Алиса очень довольна. Ведь совсем небольшой объем дополнительной разметки позволил ей полностью описать приятную человеческому взору страницу и одновременно машиночитаемый набор данных.

Алиса участвует в пяти социальных сетях. Она устала повторно вводить информацию о своих приятелях на каждом новом сайте социальных сетей. С помощью RDFa у Алисы есть возможность описать информацию о приятелях на своей web-странице в одном месте и позволить социальным сетям автоматически читать эти данные. До настоящего момента Алиса перечислила трех индивидов, однако не указала свои отношения с ними. Они могут быть ее друзьями или любимыми поэтами. Чтобы указать, что это ее друзья, она использует свойство FOAF foaf:knows:

```
<div vocab="http://xmlns.com/foaf/0.1/" about="#me" rel="knows">
  <ul>
    <li typeof="Person">
      <a rel="homepage" href="http://example.com/bob" property="name">Боб</a>
    </li>
    <li typeof="Person">
      <a rel="homepage" href="http://example.com/eve" property="name">Ева</a>
    </li>
    <li typeof="Person">
      <a rel="homepage" href="http://example.com/manu" property="name">Маша</a>
    </li>
  </ul>
</div>
```

Фрагмент разметки about=«#me» является форматом записи FOAF, где URI указывающий на *личность* Алиса это http://example.com/alice#me. Но такую запись не стоит путать с указанием домашней странички Алисы: http://example.com/alice.

Достаточно однажды объединить разметку, описывающую Боба, Еву и Машу с Алисой посредством rel=«knows» в одном несущем элементе div. Это достигается благодаря RDFa-концепции сцепления (chaining). Так как у rel верхнего уровня отсутствует связанный href, он соединяется с любой вложенной вершиной (в данном случае с тремя вершинами, заданными посредством typeof.

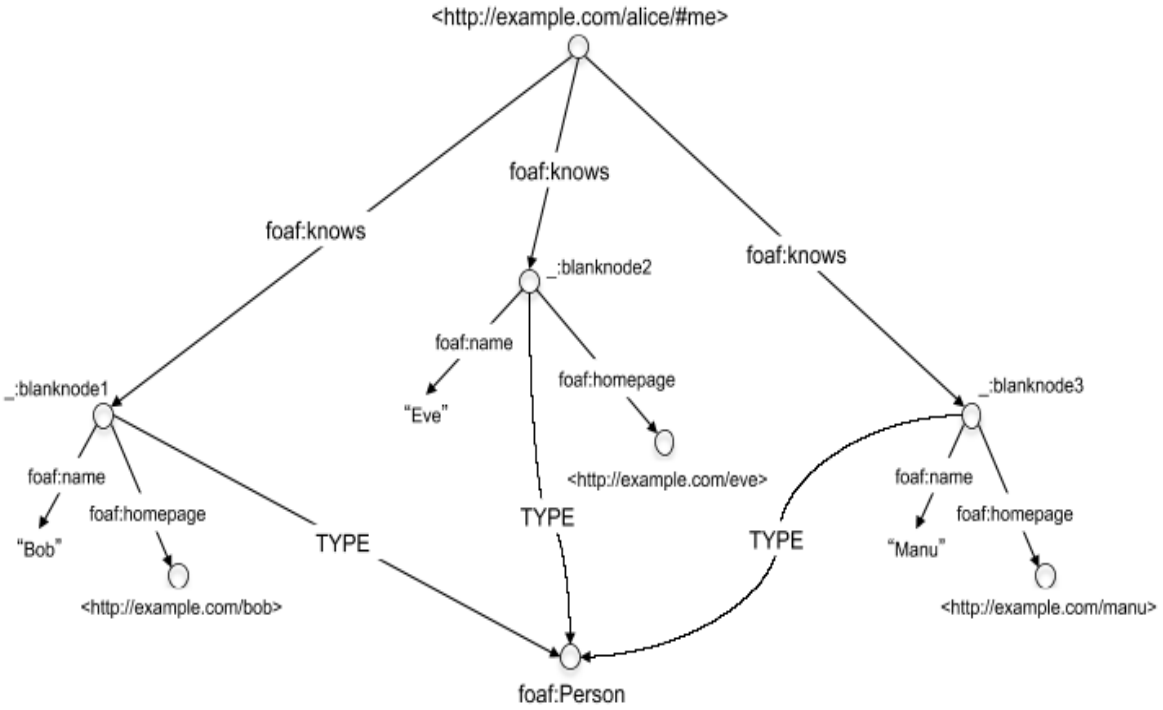

Рис. 5.7. Социальная сеть Алисы

5.4. RDFa и RDF

Технология описания ресурсов RDF – это представление абстрактных данных, которым мы как раз и занимались в предыдущих примерах, описывая структуры графов. Каждая стрелка в графе обозначена триплетом субъект – предикат – объект. Субъект – это вершина, из которой выходит стрелка. Предикат – сама стрелка, а объект – вершина, куда входит стрелка. Набор данных RDF часто называют RDF-графом, который обычно хранится в так называемом хранилище триплетов (Triple Store или Graph Store).

Рассмотрим пример графа (рис. 5.2). Два триплета для этого графа описаны с использованием синтаксиса Turtle [34]:

<http://www.example.com/alice/posts/trouble_with_bob>
<http://purl.org/dc/elements/1.1/title> " Проблема с Бобом ";
<http://purl.org/dc/elements/1.1/creator> "Алиса".

Обычно стрелки TYPE ничем не отличаются от других стрелок, единственное – их надпись rdf:type соответствует основному свойству RDF, где пространством имен rdf является http://www.w3.org/1999/02/22-rdf-syntax-ns#.

Суть RDF заключается в обеспечении пользователей универсальным языком для выражения данных. Единица данных может быть представлена произвольным числом полей, имена которых – это URL, повторно используемые любыми Web-издателями. Точно так же, как любой разработчик вправе перейти на любую Web-страницу, даже если он ее не создавал. Исходные данные представлены RDF-триплетами, собранными из разных источников и использующими язык RDF-запросов SPARQL [31]. Таким образом, можно найти «приятеля Алисы, создавшего элементы, в названии которых присутствует слово “Боб”». Причем независимо от того, являются ли эти элементы постами блога, видеоклипами, событиями календаря или другими типами данных, еще не известными на текущий момент времени.

RDF является абстрактным машиночитаемым представлением данных, призванным максимизировать повторное использование словарей. RDFa – это способ выражения RDF-данных в HTML, в рамках которого данные, предназначенные для человека, используются повторно.

Выборочные словари

По ходу разметки своей страницы посредством RDFa у Алисы может возникнуть необходимость выражения данных (например, своих фотографий), которые не описаны в существующих словарях, как Dublin Core или FOAF. Так как RDFa является лишь представлением RDF, механизм RDF-схемы, открывающий возможности расширяемости RDF, для RDFa тот же. Однажды создав RDF-словарь, он может быть впоследствии использован для разметки RDFa (аналогично существующим словарям).

Инструкции по созданию RDF-схемы в руководстве по RDF [36]. Создание RDF-схемы для RDFa на высоком уровне включает:

1) выбор URL, где будет размещен словарь;

2) размещение RDF-документа в этом URL, где определены классы и свойства, формирующие словарь. К примеру, Алиса может изъявить желание задать классы Photo и Camera, а заодно и свойство takenWith, которое связывает фотографию с камерой, с помощью которой была сделана первая;

3) использование словаря в XHTML+RDFa с обычным механизмом определения префикса, например, prefix="photo: http://example.com/photos /vocab#» и typeof="photo:Camera".

Для тех, кто публикует документы в глобальной сети, ничего не стоит опубликовать и RDF-словарь, определив, таким образом, новые необходимые поля данных. RDF и RDFa полностью поддерживают распределенную расширяемость словарей.

Примеры использования RDFa

Рассмотрим группу дополнительных RDFa примеров. Многие из них описывают не только кодирование данных в RDF, но и возможную область применения приложений, работающих с данными. Обратите внимание на то, что для реализации некоторых из представленных примеров может потребоваться программный доступ к RDFa содержимому (например, с помощью JavaScript). Такой доступ к данным может быть реализован с использованием RDFa API [43].

Импорт данных

RDFa применяются для разметки контента, помогающей поисковым системам (search engines) при отображении расширенных результатов поиска:

```
<div xmlns:v="http://rdf.data-vocabulary.org/#" typeof="v:Review">
  <div rel="v:itemreviewed">
    <span typeof="Organization">
      <span property="v:name">L'Amourita Pizza</span>
      Адрес:
   <span rel="v:address">
        <span typeof="v:Address">
          <span property="v:street-address">123 Мэйн-стрит</span>,
          <span property="v:locality">Альбукерке</span>,
          <span property="v:region">Нью-Мексико</span>.
        </span>
      </span>
      <a href="http://pizza.example.com/" rel="v:url">http://pizza.example.com</a>
    </span>
  </div>
  Автор:
  <span property="v:reviewer">Боб Смит</span>.
  Оценка:
  <span rel="v:rating">
```

```
      <span property="v:value">9</span>/
      <span property="v:best">10</span> (отлично)
  </span>
</div>
```

Поисковая машина Google выполнив обработку приведенного выше фрагмента предоставит следующий отзыва о компании L'Amourita Pizza:

```
<div>
L'Amourita Pizza
Адрес: 123 Мэйн-стрит, Альбукерке, Нью-Мексико.
<a href="http://pizza.example.com">http://pizza.example.com</a>
Автор: Боб Смит.
Оценка: 9/10 (отлично)
<div>
```

Можно также использовать RDFa API для автоматического извлечения информации со странички и дополнительный JavaScript для добавления информации в ее персональный календарь.

Модификация web-страниц на основе существующих данных

У Димы есть сайт с некоторым числом изображений, представляющий его фотоработы. Он использовал RDFa для указания лицензионной информации проследовав инструкциям Creative Commons. Дима пожелал отобразить иконки Creative Commons для каждой своей работы чтобы люди могли получить представление о том, на каких условиях распространяется каждая фотография. Он пишет несколько строк JavaScript с использованием RDFa API, распаковывающим URI закрепленной лицензии и затем использует это в качестве условия загрузки соответствующего лицензии изображения с сайта Creative Commons:

```
<div prefix="cc: http://creativecommons.org/ns#">
  <img src="http://dale.example.com/images/image1.png"
      rel="cc:license"
      resource="http://creativecommons.org/licenses/by/3.0/us/"/>
  <a rel="cc:attributionURL" rel="cc:attributionURL" href="http://dale.example.com"
rel="cc:attributionURL" property="cc:attributionName"></a>
</div>
```

Автоматические сноски-резюме

На Марии лежит ответственность за поддержание в актуальном состоянии страничку проектов компании. Она хочет отображать информационные блоки с комплексной информацией о всех лицах задействованных в проекте. Информация должна появиться в момент наведения курсора на ссылку домашней страницы каждого участника проекта. Поскольку страницы всех участников уже размечены с применением RDFa, Мария пишет скрипт запрашивающий содержимое web-страниц и извлекающий нужные данные с помощью RDFa API.

Чтобы реализовать уникальную идентификацию для разных областей, Мария решает использовать URI вместо обычного текста. Она выбирает термины определенные DBpedia. DBPedia являет собой дамп данных Wikipedia представленных в виде словаря. Это широко применяется семантическим web для обозначения концептов человеческого мира. Мария использует специальные RDFa атрибуты обозначенные как resource, что в некоторой степени играет роль href, но не реализует собой кликабельной ссылки подобно href в атрибуте *a*. Это позволит ей добавить некоторую справочную информацию для человекочитаемой версии интересующей страницы Wikipedia. Поскольку оба атрибута(resource и href) могут быть установлены на какой-то элемент, то один из атрибутов (resource) в RDFa имеет приоритет над href и может быть использован для перенаправления пользователя на человеко-ориентированную страницу с DBPedia. В результате Мария прибегает к помощи RDFa API для извлечения информации подобного рода прямо из HTML-исходника по порядку для наполнения инфобоксов:

```
<div prefix="dc: http://purl.org/dc/terms/ foaf: http://xmlns.com/foaf/0.1/"
    about="#me" typeof="foaf:Person">
  <span property="foaf:name" content="Bob">My</span> interests are:
  <ol about="#me" typeof="foaf:Person" rel="foaf:interest">
    <li><a resource="http://dbpedia.org/resource/Semantic_Web"
        href="http://en.wikipedia.org/wiki/Semantic_Web" property="dc:title">Semantic Web</a>
    </li>
    <li><a resource="http://dbpedia.org/resource/Facebook"
        href="http://en.wikipedia.org/wiki/Facebook" property="dc:title">Facebook</a>
    </li>
    <li><a resource="http://dbpedia.org/resource/Twitter"
        href="http://en.wikipedia.org/wiki/Twitter" property="dc:title">Twitter</a>
    </li>
  </ol>
</div>
```

Визуализация адресов

Михаил создал сайт, предоставляющий список его любимых кафе с описанием их местоположений. У него нет желания писать код, ориентированный на различные сервисы работающие с картами. Вместо того, чтобы реализовывать специальную разметку под Yahoo Maps, Google Maps, MapQuest и Google Earth он решает добавить универсальную информацию об адресах для каждого кафе. Это позволяет ему создать инструмент извлекающий адресную информацию и связывающий эту информацию с сервисом карт на выбор пользователя:

```
<div prefix="vc: http://www.w3.org/2006/vcard/ns# foaf: http://xmlns.com/foaf/0.1/"
typeof="vcard:VCard">
  <span property="vcard:fn">Wong Kei</span>
  <span property="vcard:street-address">41-43 Wardour Street</span>
  <span property="vcardlocality">London</span>, <span property="vcard:country-name">United
Kingdom</span>
```

```
  <span property="vcard:tel">020 74373071</span>
</div>
```

Смешивание связанных данных (Linked Data)

Катя химик, изучающий эффекты производимые этанолом на среду обитания диких тушканчиков. Она делает пометки о своих наблюдениях в своем блоге и часто указывает ссылки на описания химических соединений. Она хотела бы иметь возможность отобразить во всплывающей подсказке графическую информацию о химическом соединении, ссылку на описание, находящееся на сайте Национального Центра Биологической Информации [NCBI]. А также она желает позволить посетителям иметь визуализованные данные на странице используя виджет основанный на canvas, новом элементе HTML5 и RDFa, собранный из различных примеров найденных на разных сайтах посвященных химии:

```
<div prefix="dbp: http://dbpedia.org/ontology/ fb: http://rdf.freebase.com/rdf/">
  My latest study about the effects of
  <span about="fb:en.ethanol"
    typeof="dbp:ChemicalCompound"
    property="fb:chemistry.chemical_compound.pubchem_id"
    content="702">ethanol</span> on mice's spatial orientation show that...
</div>
```

Расширенные браузерные интерфейсы

Денис написал плагин для браузера, который сортирует товары на веб-странице и отображает иконки покупки продукта или помещающает понравившиеся товары в публичный список желаний. Плагин ищет по критериям упоминания, превью и объявленной цены. Информация перечислена в адресной строке в виде иконок, по клику на которые появляется в сайдбаре. Денис может добавить каждый продукт в список, управляемый плагином и опубликовать его в списке планов на web-сайте:

```
<div prefix="foaf: http://xmlns.com/foaf/0.1/">
   <div vocab="http://purl.org/goodrelations/v1#" about="#offering" typeof="Offering">
   <div rel="foaf:page" resource="http://www.amazon.com/Harry-Potter-Deathly-Hallows-
Book/dp/0545139708"></div>
   <div property="rdfs:label">Harry Potter and the Deathly Hallows</div>
   <div property="rdfs:comment"> В этой, заключительной, седьмой части Гарри Поттера,
Джоан Роулинг дает ответы на многие вопросы, которые все с таким нетерпением ждали.
</div>
   <div rel="foaf:depiction">
     <img src="http://ecx.images-amazon.com/images/I/51ynI7I-qnL._SL500_AA300_.jpg" />
   </div>
   <div rel="hasBusinessFunction" resource=http://purl.org/goodrelations/v1#Sell"></div>
   <div rel="hasPriceSpecification">Buy for
    <span typeof="UnitPriceSpecification">
      <span property="hasCurrency" content="USD">$</span>
      <span property="hasCurrencyValue">7.49</span>
    </span>
```

```
        </div> Pay via:
          <span rel="acceptedPaymentMethods" re-
source="http://purl.org/goodrelations/v1#PayPal">PayPal</span>
          <span rel="acceptedPaymentMethods" re-
source="http://purl.org/goodrelations/v1#MasterCard">MasterCard</span>
        </div>
      </div>
    </div>
```

Поскольку многие из страничек используют онтологии Good Relation, широко используемые для разметки продуктов, Денис вполне может применить средства RDFa, такие как vocab, для упрощения задачи. Кроме того, он забывает определить префикс rdfs. К счастью, базовый профиль RDFa уже содержит его, и данные, которые он намеревался описать будут обработаны всеми RDFa совместимыми процессорами.

6. RDF – УНИВЕРСАЛЬНЫЙ СПОСОБ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ

RDF – это способ представления знаний в децентрализованном мире, представляющий основную технологию семантического веба, который позволит компьютерным программам пользоваться всей структурированной информацией, распределенной по узлам веба. RDF – это универсальный способ разложения любых знаний на маленькие кусочки. Он задает определенные правила касательно семантики, т. е. смысла этих кусочков. Идея состоит в том, чтобы одним простым способом можно было бы описать любой факт, притом в таком структурированном виде, чтобы его могли обрабатывать компьютерные программы. Вот пример RDF в формате N3 (субъект-предикат-объект или подлежащее-сказуемое-дополнение) [27]:

Пример 6.1.

```
@prefix : <http://www.example.org/>.
:джон   a          :Person.
:джон   :hasMother  :сусанна.
:джон   :hasFather  :ричард.
:ричард :hasBrother :лука.
```

Первая строка примера определяет префикс «:», который будет относить любую сущность (субъект, предикат, объект) записанную за ним, к ресурсу (документу) http://www.example.org. Смысл представленных в примере 6.1 фактов достаточно очевиден:

Джон – личность.
Сусанна мать Джона.
Ричард отец Джона.
Лука брат Ричарда.

Как RDF, так и XML – простые и универсальные технологии. XML может использоваться для представления более абстрактных вещей, чем тот текст с угловыми скобками, которым он записан: им может задаваться объектная модель какой-либо древовидной структуры. Точно так же, RDF – это не просто формат записи; это формат представления информации, организованной в виде графов или сетей. Можно записывать RDF в виде XML, тогда предыдущий пример будет выглядеть так:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:ns="http://www.example.org/#">
  <ns:Person rdf:about="http://www.example.org/#джон">
    <ns:hasMother rdf:resource="http://www.example.org/#сусанна" />
    <ns:hasFather>
      <rdf:Description rdf:about="http://www.example.org/#ричард">
        <ns:hasBrother rdf:resource="http://www.example.org/#лука" />
      </rdf:Description>
```

```
    </ns:hasFather>
  </ns:Person>
</rdf:RDF>
```

Но использовать XML необязательно – формат N3, показанный в примере 6.1, выглядит значительно проще. Сильнее всего RDF отличается от XML и других технологий – это то, что RDF предназначен для представления знаний в распределенном мире. Другими словами, для RDF особенно важен смысл. Все, с чем работает RDF, имеет определенный смысл – ссылается на какой-то конкретный объект, или на абстрактное понятие, или на некий факт. Стандарты, основанные на RDF, описывают логические выводы, связывающие эти факты, и указывают, как можно найти сами факты в огромной базе данных всех знаний, представленных в RDF.

RDF подходит для работы с распределенными знаниями потому, что приложения могут собирать воедино RDF-файлы, размещенные в Интернете разными людьми, и с легкостью узнавать из собранного документа даже те новые вещи, которых не было ни в одной из его частей. В RDF предусматривается два процесса, благодаря которым это осуществляется – во-первых, объединяются документы, использующие общие языки, и во-вторых, допускается использовать любые языки в каждом из документов. Эта гибкость – одна из отличительных черт RDF.

Рассмотрим второй пример RDF-документа:

Пример 6.2.

```
@prefix : <http://www.example.org/>.
:ричард :hasSister :ребекка
{ ?a :hasFather ?b. ?b :hasSister ?c. } => { ?a :hasAunt ?c }.
```

В этом RDF-документе определяется смысл отношения «быть теткой» – оно выражается через два других отношения. Легко представить себе, как приложение, объединяющее этот документ с документом из первого примера, сможет понять из них, что :ребекка – тетка :джона. Это возможно потому, что имена сущностей глобальные; если в одном документе используются :джон и :hasFather, то приложения подразумевают, что они имеют тот же самый смысл и во всех остальных документах с тем же значением @prefix.

Ниже представлен список областей его применимости RDF, приведенный в [47]:

- объединение данных из различных источников, не прибегая к созданию специализированных программ;
- децентрализация данных так, чтобы ими всеми не «владел» кто-то один;
- работа с большими объемами данных – ввод, извлечение, просмотр, анализ, поиск, и т. д.;

• создание (либо использование готового) универсального инструмента, основанного на модели данных RDF (имеющего то преимущество, что она не привязана к закрытым технологиям хранения и представления данных – в отличие от диалектов СУБД).

Определение RDF

RDF можно определить как совокупность трех простых правил.

1. Факт выражается тройкой вида (Подлежащее, Сказуемое, Дополнение) – похожей на простое предложение на естественном языке.

2. Подлежащие, сказуемые и дополнения – это имена сущностей реального мира, конкретных или абстрактных. Имя может быть 1) глобальным, ссылающимся на одну и ту же сущность во всех RDF-документах, где оно используется, либо 2) локальным, и тогда на сущность, на которую ссылается это имя, нельзя ссылаться из-за пределов RDF-документа непосредственно.

3. Дополнения, кроме этого, могут быть текстовыми строками – «литералами».

Что касается фактов, то каждая из этих строк – факт:

```
:джон  a          :Person.
:джон  :hasMother :сусанна.
```

Имена сущностей реального мира бывают двух типов. Глобальные имена, имеющие всюду один и тот же смысл, всегда имеют вид URI (универсальных идентификаторов ресурсов). Синтаксис (формат) URI может быть таким же, как у адреса веб-сайта; вы можете встретить RDF-файлы, где используются URI наподобие http://www.w3.org/1999/02/22-rdf-syntax-ns#type; такие URI задают глобальные имена для каких-нибудь сущностей. Тот факт, что это похоже по виду на веб-адрес – чистое совпадение. Есть и другие типы URI, не начинающиеся с http:. URN – один из подтипов URI, и он используется, например, для идентификации книг по ISBN-номеру – urn:isbn:0143034650 и т. п. Другой универсальный тип URI – это TAG; пример такого URI – tag:govtrack.us,2005:congress/senators/frist. URI используются для глобальных имен потому, что они позволяют разбить пространство всех возможных имен на блоки, за которыми закреплены владельцы. Это очень важный принцип: какого бы типа URI ни были, они используются в RDF-документах только в качестве имен сущностей, и больше ни для чего. Они могут не иметь никакого отношения к вебу.

Так как URI могут быть довольно длинными, то в форматах, используемых для представления RDF, они обычно сокращаются, используя перенятый из XML механизм «пространств имен». Именно поэтому в именах :джон, :hasMother и других сущностей в приведенных примерах стоят двоеточия – они означают, что используются сокращенные имена. В наших примерах им соответствовали полные имена http://www.example.org/#джон, http://www. example.org/#hasMother и т. д. При записи URI обычно заключаются в угловые скобки, чтобы их можно было отличить от сокращенных имен.

Литералы позволяют включать в RDF текст. Они используются особенно интенсивно, когда с помощью RDF представляются метаданные:

<http://www.rdfabout.net/> a :Website.
<http://www.rdfabout.net/> dc:title "rdf:about".
<http://www.rdfabout.net/> dc:description "Веб сайт о RDF.".

RDF в виде графа

Есть два взаимодополняющих способа рассматривать информацию, представленную в RDF. Первый способ – считать ее набором утверждений, как в примерах выше: каждое утверждение представляет собой факт. Второй способ – считать ее графом.

Граф – это, в общем, то же самое, что сеть. Граф состоит из узлов, соединенных ребрами. Например, в интернете узлы – это компьютеры, а ребра – соединяющие их каналы связи. В RDF узлы – это имена (но не сами сущности), а ребра – утверждения.

Например, на рис. 6.1 каждая стрелка (ребро) – это RDF-утверждение: имя у начала стрелки – это подлежащее утверждения, имя у конца стрелки – его дополнение, и имя у самой стрелки – сказуемое. Когда RDF представлен в виде графа, он содержит всю ту же информацию, что и выписанный в виде троек-утверждений; но графическое представление позволяет человеку легче увидеть структуру описываемых данных.

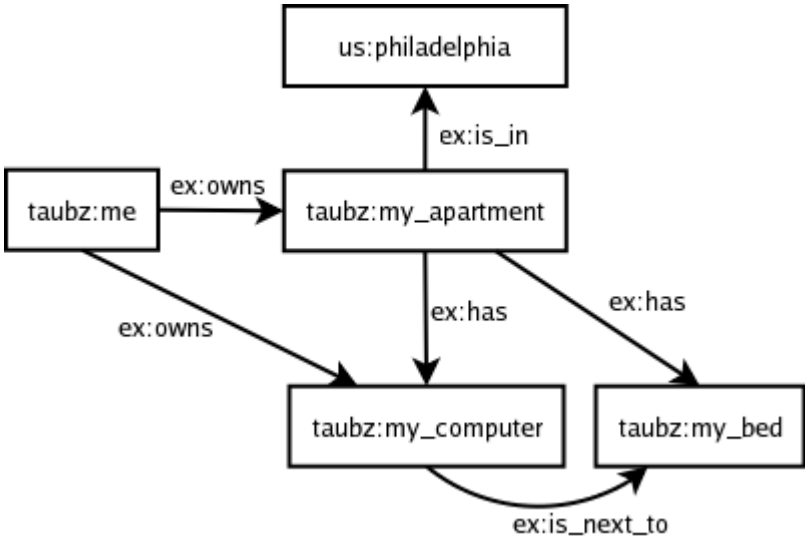

Рис. 6.1. Представление RDF в виде графа

Практический пример

Главное, RDF полезен тем, что это самая подходящая технология, когда нужно свести распределенную информацию в общую систему.

Вот пример ситуации, когда информация была бы распределенной: база данных о продуктах различных поставщиков и оценках этих продуктов различными экспертами. Ни один поставщик не согласится взять на себя

ответственность за поддержание общей центральной базы данных, в том числе и потому, что она содержала бы информацию о его конкурентах, и возможно, отрицательные оценки его собственной продукции. У экспертов же может не хватать ресурсов для постоянного поддержания в общей базе данных актуальной информации.

RDF бы подошел в этом проекте особенно хорошо. Каждый поставщик и каждый эксперт размещали бы на собственном веб-сервере RDF-файл. Поставщики выбирают для своих продуктов URI, и эксперты используют эти URI, когда публикуют свои оценки. Тогда поставщики не вынуждены выбирать общую схему наименования для своих продуктов, и эксперты не привязаны к форматам данных, выбранным поставщиками. RDF позволяет и поставщикам, и экспертам использовать те инструменты, которые им удобнее, и никто не заставляет никого пользоваться каким-то определенным языком.

Такой вид имели бы RDF-файлы, которыми они обмениваются:

Поставщик 1:

```
vendor1:productX      dc:title       "Cool-O-Matic".
vendor1:productX      retail:price   "$50.75".
vendor1:productX      vendor1:partno "TTK583".
vendor1:productY      dc:title       "Fluffertron".
vendor1:productY      retail:price   "$26.50".
vendor1:productY      vendor1:partno "AAL132".
```

Поставщик 2:

```
vendor2:product1      dc:title       "Can Closer".
vendor2:product2      dc:title       "Dust Unbuster".
```

Эксперт 1:

```
vendor1:productX      dc:description "This product is good buy!".
```

Эксперт 2:

```
vendor2:product2  dc:description  "Who needs something to unbust dust?
                                   A dust buster would be a better idea,
                                   and I wish they posted the price.".
vendor2:product2  review:rating   review:Excellent.
```

Отдельный вопрос – как именно они обменивались бы этими файлами; его пока оставим в стороне. Как только приложение получает эти файлы, у него появляется достаточно информации, чтобы соотнести продукты, цены, оценки, и даже такую специфическую информацию, как vendor1:partno (шифр изделия). То, что вы должны понять из этого примера, – это насколько гибок RDF: он не накладывает практически никаких ограничений и все же позволяет приложениям моментально соотносить распределенную информацию.

Поставщикам и экспертам не приходилось идти на компромиссы: они договорились использовать RDF, и все. Им не приходилось договариваться о конкретном формате данных, или даже о конкретных URI. Что важнее всего, им не пришлось перечислять заранее все пункты, которые поставщик может включить в информацию о своем продукте, и в этой системе ни одного из экспертов невозможно лишить возможности публиковать свои оценки.

Кроме того, можно рассмотреть эту систему с точки зрения взаимной совместимости. Формат, используемый Поставщиком 1, полностью совместим с форматом, используемым всеми остальными, хотя они не вырабатывали общий формат целенаправленно. Когда в системе появится новое действующее лицо, которое захочет работать с информацией Поставщика 1, ему не потребуется реализовывать новый формат: все что ему будет нужно – это выбрать те же подлежащие, сказуемые и дополнения, которые выбрал Поставщик 1.

7. OWL – АВТОМАТИЗАЦИЯ ИНТЕРПРИТАЦИИ КОНТЕНТА

Синтаксическое взаимодействие сетей – необходимое условие для того, чтобы множественные приложения могли по-настоящему «понимать» данные и работать с ними как с информацией. Это также необходимое условие для корректной проверки данных. Синтаксическое взаимодействие сетей требует преобразования («мэппирования») между терминами, для чего, в свою очередь, необходим контент-анализ.Такой контент-анализ требует формальных и подробных спецификаций моделей доменов, которые определяют используемые термины и их связи. Подобные формальные модели доменов называются ***онтологиями***. Они определяют модели данных в терминах классов, подклассов и свойств.

Онтологический язык OWL (Web Ontology Language), рекомендуемый консорциумом W3C, помогает в выражении онтологий и добавляет больше словарных возможностей для описания свойств и классов, чем RDF или схема RDF. В частности, он позволяет описывать связи между классами (например, неперекрываемость), мощность множества (например, «ровно один»), равенство, более богатую типологию свойств и их характеристики (например, симметрию).

Онтологический язык OWL (рис. 7.1) разработан для использования приложениями, которые должны работать с содержанием информации, а не просто предоставлять ее пользователю. OWL улучшает возможности автоматической интерпретации содержимого Интернета по сравнению с теми, что могут обеспечить XML, RDF и схема RDF. Это происходит благодаря тому, что OWL предоставляет дополнительные словарные возможности наряду с формальной семантикой.

Примерами онтологий являются каталоги сайтов интерактивных покупок, таких как Amazon.com, стандартные терминологии той или иной области деятельности, например, UNSPSC – The United Nations Standard Products and Services Code (система стандартных продуктов и услуг ООН), или различные таксономические системы Интернета, такие как категории сайта «My Yahoo».

В следующих разделах будет подробнее рассказано о различных компонентах OWL.

Компоненты онтологического языка OWL

Основные компоненты OWL включают ***классы***, ***свойства*** и ***индивидуальные элементы***.

Классы – это основные блоки онтологии OWL. Класс – это концепция в домене. Классы обычно образуют таксономическую иерархию (т. е. систему подкласс – надкласс).

Классы определяются с помощью элемента owl:Class. В языке OWL существует два заранее определенных класса: owl:Thing и owl:Nothing.

Первый из них является наиболее общим и включает все, второй – это пустой класс. Любой класс, определяемый пользователем, является подклассом класса owl:Thing и надклассом класса owl:Nothing. Примеры классов в области банковского дела могут включать классы Счет (Account) или Клиент (Customer).

Ниже представлен пример класса OWL:

```
<owl:Class rdf:ID="SavingsAccount">
 <rdfs:subclassOf rdf:resource="#Account"/>
</owl:Class>
```

Код в примере указывает, что элемент SavingAccount – это класс, являющийся подклассом класса Account.

OWL поддерживает шесть основных способов описания классов. Самый простой – это ***класс с именем*** (named). Другие типы – это классы ***пересечений*** (intersection), ***объединений*** (union), ***дополнений*** (complement), ***ограничений*** (restrictions) и классы ***перечислений*** (enumerated). В примере представлены два из этих способов описания классов: класс ограничений определяет SavingAccount как подкласс класса с именем Account.

Свойства включают две основные категории:

- ***свойства объекта*** (Object properties), которые связывают индивидуальные элементы между собой;
- свойства типов данных (Datatype properties), которые связывают индивидуальные элементы со значениями типов данных, такими как целые числа, числа с плавающей запятой и строки. Для определения типов данных OWL использует схему XML.

Свойство может включать домен и некоторую область, связанную с ним. Любое свойство попадает в одну из следующих категорий:

- функциональная: для любого объекта свойство может принимать только одно значение (например, возраст, рост или вес человека);
- обратно-функциональная: два различных индивидуальных элемента не могут иметь одно и то же значение. Например, у каждого человека свой уникальный номер банковского счета или так называемый SSN (social security number);
- симметричная: если свойство связывает элемент *A* с элементом *B*, то из этого можно сделать вывод, что оно также связывает элемент *B* с элементом *A*. Примеры симметричных свойств включают выражения типа «является братом (сестрой)» или «такой же, как»;
- транзитивная: если свойство связывает элемент *A* с элементом *B*, а элемент *B* с элементом *C*, то можно предположить, что оно также связывает элемент *A* с элементом *C*. Например, если *A* выше *B*, а *B* выше *C*, то *A* выше *C*.

К классам и свойствам могут применяться различные ограничения. Например, ограничения мощности множества указывают на число связей, в которых может участвовать класс или индивидуальный элемент.

Ссылку на полный пакет спецификаций W3C OWL можно найти в [5].

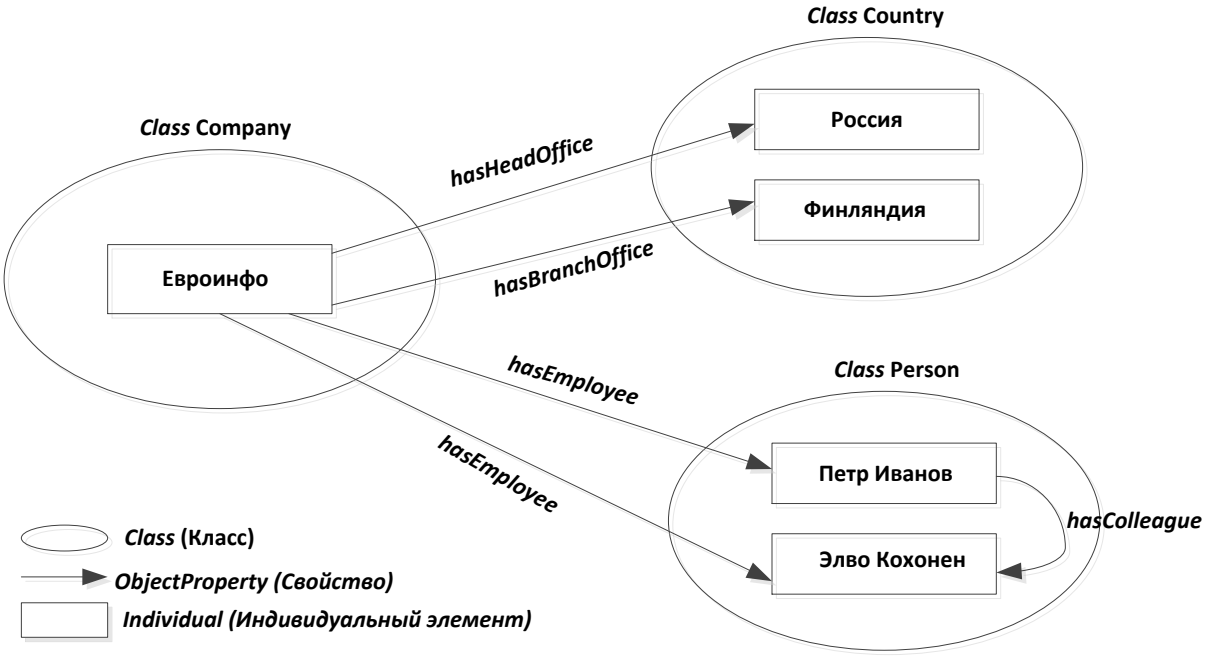

Рис. 7.1. Онтология OWL,
описывающая организационную структурукомпании Евроинфо

Индивидуальные элементы – это элементы классов; свойства могут связывать их друг с другом. Например, индивидуальный элемент «Элво Кохонен» может быть описан как элемент, принадлежащий классу Person. Свойство hasEmployer (имеет работодателя) может связывать его с другим индивидуальным элементом – «Евроинфо», указывая, таким образом, что «Элво Кохонен» работает в компании «Евроинфо».

Ниже приведен пример индивидуального элемента OWL:

```
<owl:Thing rdf:about="SmithAccount">
 <rdfs:type="#Account"/>
</owl:Class>
```

Элемент rdf:type – это свойство RDF, которое связывает индивидуальный элемент с тем классом, к которому он принадлежит. Пример указывает, что элемент SmithAccount принадлежит к типу Account.

В качестве примера приведем онтологию семьи (табл. 7.1, 7.2), которая не требует дополнительных разъяснений.

Таблица 7.1

Пространство имен, классы и подклассы, свойства объектов

Пространства имен	*Свойства объектов*
Namespace: f = <http://example.com/owl/families#> Namespace: g = <http://example.com/owl2/families.owl#> Namespace: dc = <http://purl.org/dc/elements/1.1/>	ObjectProperty: hasWife ObjectProperty: hasHusband Inverses: hasWife ObjectProperty: hasSon ObjectProperty: hasDaughter ObjectProperty: hasGender ObjectProperty: hasChild ObjectProperty: hasAncestor Characteristics: Transitive, Irreflexive ObjectProperty: hasSpouse Characteristics: Symmetric, Irreflexive ObjectProperty: loves Domain: Person DataProperty: hasAge Characteristics: Functional
Классы и подклассы Class: Person Class: Man SubClassOf: Person Class: Woman SubClassOf: Person Class: Parent SubClassOf: Person Class: Teenager Class: Adult Class: Child Class: YoungChild Class: Marriage Class: ReligiousMarriage DisjointWith: CivilMarriage Class: CivilMarriage Class: Narcissist EquivalentTo: Person that loves Self	

Таблица 7.2

Объекты онтологии семьи

Individual: John Types: Person Facts: hasWife Mary, hasSon Bill, hasDaughter Susan, hasAge 33, hasGender male SameAs: Jack Individual: Mary Facts: hasSon Bill, hasDaughter Susan, hasAge 31, hasGender female Individual: Bill Types: not (Narcissist) Facts: hasAge 13, hasGender male Individual: Susan Facts: hasAge 8, hasGender female	Individual: Jeff Types: hasChild exactly 2 Facts: hasWife Emily, hasChild Ellen, hasChild Jack, hasAge 77, loves Jeff Individual: Emily Types: hasAge some {39 , 49} Individual: Ellen Types: hasAge some integer[>= 15 , <= 21] Individual: Jack Facts: not hasAge "53"^^integer DifferentIndividuals: f:John f:Mary f:Bill f:Susan DifferentIndividuals: f:Jeff f:Emily f:Jack f:Ellen f:Susan SameIndividual: f:male g:masculine SameIndividual: f:female g:feminine EquivalentClasses: f:Adult g:Grownup EquivalentObjectProperties: f:hasChild g:child EquivalentDataProperties: f:hasAge g:age

Примером отечественных разработок в области семантических моделей представления знаний может стать OntoQuad RDF Server [48], предназначенный для хранения различных объектных данных, имеющих структуру, которая может быть представлена в виде графа (рис. 7.2).

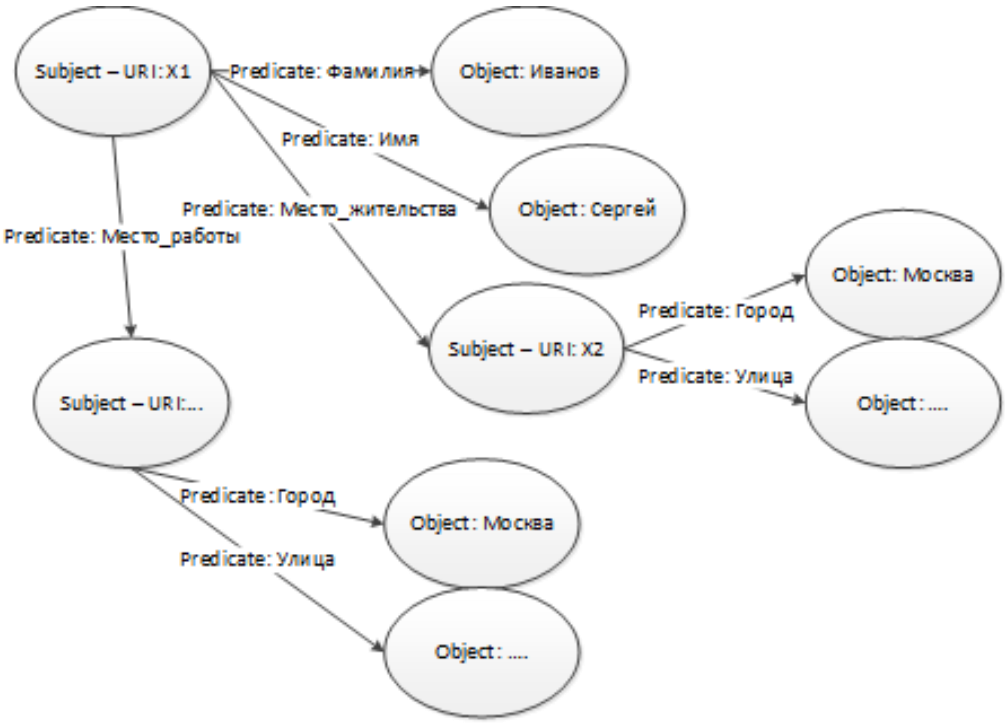

Рис. 7.2. Представление структуры хранения в виде графа

Так как RDF база относится к NoSQL типу баз данных, то для выполнения поиска данных используется специальный язык запросов SPARQL.
К преимуществам RDF баз данных можно отнести:

1) возможности логического вывода на основании имеющихся данных;
2) гибкая модель данных – отсутствие необходимости структуры схем и таблиц для хранения данных;
3) беспрецедентные возможности по интеграции различных систем для обмена данными;
4) получение новых данных из облака связанных открытых данных (Linked Open Data) без необходимости программирования;
5) стандартизация, открытость и экспрессивность.

В частности OntoQuad RDF Server обеспечивает транзакционную многопоточную систему управления базами данных, включая:

1) поддержку стандарта SPARQL 1.1 с небольшими ограничениями для текущей версии;
2) систему пакетной загрузки БД из структурированных текстовых файлов форматов N-Triples и Turtle;
3) точку доступа SPARQL;
4) бэкапирование и восстановление БД из файлов бэкапа.

Роль семантических технологий OWL

Для того чтобы соответствующим образом моделировать и управлять СОА (сервис-ориентированной архитектурой), корпоративные архитекторы должны поддерживать активное представление услуг, доступных для корпорации. В частности, для выявления и организации своих услуг, архитекторы должны использовать передовой опыт в моделировании и объединении услуг с использованием метаданных, преобразовывать бизнес-логику в метаданные для динамического объединения и осуществлять управление с помощью метаданных. Онтология обеспечивает очень мощный и гибкий способ для агрегирования, визуализации и нормализации этого слоя услуг с помощью метаданных.

Онтология – это сеть концепций, связей и ограничений, которые обеспечивают контекст для данных и информации, а также для процессов. Онтология способствует улучшению обнаружения услуг, моделирования, объединения, посредничества и семантического взаимодействия сетей. Она усовершенствует для пользователей способы поиска, изучения и взаимодействия со сложными информационными пространствами метаданных. Бизнес-онтология – это формальная спецификация бизнес-концепций и их взаимосвязей, которая улучшает машинные причинно-следственные связи и взаимодействия. Бизнес-онтология связывает системы, используя метаданные, во многом аналогично тому, как база данных объединяет разрозненные данные. Такая абстракция обеспечивает гибкость и подвижность, поскольку позволяет легко менять интерфейсы, а также добавлять новые ресурсы и пользователей, причем даже во время работы системы.

Семантика – это будущее сервис-ориентированной интеграции. Семантические технологии обеспечивают существование определенного уровня абстракции над существующими IT-технологиями. Этот уровень позволяет осуществлять связь данных, содержания и процессов между различными видами бизнеса и изолированными IT-структурами. Наконец, с точки зрения взаимодействия людей, семантические технологии добавляют новый уровень семантических порталов, которые обеспечивают гораздо более аналитические, соответствующие теме и контексту взаимодействия, чем те, которые доступны с помощью традиционных точечных подходов к интеграции, использующихся в информационных порталах.

8. ЯЗЫК SPARQL

Основным способом получения информации, хранящейся в виде семантических моделей является SPARQL – *рекурсивный акроним* от *SPARQL Protocol and RDF Query Language* [45]. Как следует из названия, стандарт SPARQL определяет две вещи:

- протокол обмена RDF-данными, построенный поверх протокола HTTP;
- язык запросов к RDF-данным.

SPARQL является рекомендацией *консорциума W3C* и очень тесно связан с RDF технологиями. Фактически SPARQL-запросы являются фрагментами RDF-графов, в которых кроме URI и литералов есть еще и переменные. SPARQL разработан таким образом, что он легко осваивается пользователями знакомыми с языком SQL.

Для ввода запросов на получение информации в вебе организованы сервисы, обрабатывающие запросы пользователей и возвращающие результат в различных форматах. Эти сервисы носят название SPARQL-endpoints или точки доступа. Различают два вида точек доступа – общего назначения и локальные. Точки доступа общего назначения позволяют производить запросы к любым указанным RDF-документам, находящимся в сети, в то время как локальные endpoint'ы способны получать данные только от одного ресурса. Пример локальной точки доступа: DBPedia endpoint (http://dbpedia.org/sparql), точки доступа общего назначения: ARQ endpoint (http://sparql.org/sparql.html).

Набор данных DBpedia представляет из себя большую многодоменную онтологию которая была получена из Википедии. Английская версия DBpedia в настоящее время описывает 4,0 миллиона «сущностей» с 470 миллионами «фактов».

Кроме того, реализованы локализованные версии DBpedia на 119 языках. Все эти версии вместе описывают 24,9 миллиона объектов. Полный набор данных DBpedia индексирует и описывает 12,6 миллионов уникальных вещей на 120 различных языках, использует 24,6 миллиона ссылок на изображения и 27,6 миллионов HTML-ссылок на внешние веб-страницы. DBpedia использует 45,0 миллионов ссылок на внешние наборы RDF-данных, 67,0 миллионов ссылок на категории Wikipedia и 41,2 миллиона ссылок на YAGO категории. Этот набор данных состоит из 2,46 миллиарда единиц информации (RDF троек), из которых 470 миллионов были взяты из английского издания Википедии, 1,98 миллиарда были извлечены из других языковых разделов.

Перед началом работы всегда полезно знать, как много данных есть в хранилище, с которым предстоит взаимодействовать. Напишем следующий запрос:

```
SELECT COUNT(?x) WHERE
{
?x a ?z.
}
```

Подадим его в точке доступа DBpedia endpoint и получим численную оценку количества фактов 63553605, которые доступны на текущий момент.

Другой запрос, который используется довольно часто:

```
SELECT DISTINCT *
WHERE
{
?x ?y ?z.
}
LIMIT 10
```

Он позволяет получить случайную выборку из 10 фактов, находящихся в хранилище. Общая схема SPARQL-запроса выглядит так:

```
# определение префиксов
PREFIX foo: http://example.com/resources/
...
# перечень результатов
SELECT...
# определение хранилища
FROM...
# описание условий выборки
WHERE {
  ...
# фильтрация выборки
FILTER (…)
}
# ограничение количества фактов в результате
LIMIT n
# модификация представления результата
ORDER BY...
```

В SPARQL имеется возможность указывать URI двумя способами: с использованием префиксов и без них. В первом случае используется ключевое слово PREFIX:

```
PREFIX prop: <http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/resource/>
SELECT ?Amazon,?Nile
{
 dbpedia:Amazon_River  prop:length  ?Amazon.
 dbpedia:Nile          prop:length  ?Nile.
}
```

Здесь определены два префикса prop и dbpedia, которые и используются при описании условий выборки. В качестве субъектов используются две реки, а свойством (предикатом) связывающим их с искомым результатом является запрашиваемая длина. Достаточно очевидно, что результатом этого запроса будет длина двух рек:

Amazon	Nile
6400	6650

Этот же запрос без использования префиксов будет выглядеть следующим образом:

```
SELECT ?Amazon,?Nile
{
  <http://dbpedia.org/resource/Amazon_River>  <http://dbpedia.org/property/length> ?Amazon.
  <http://dbpedia.org/resource/Nile>  <http://dbpedia.org/property/length>  ?Nile.
}
```

Конечно, использование префиксов более предпочтительно, поскольку существенно сокращается размер запроса и он более удобен для понимания.

При использовании в запросах литералов (независимо от их типа) – они указываются в двойных «""»или одинарных «''» кавычках. Также к литералам можно добавлять указатель языка через @:

```
"текст на русском"@ru
```

Тип литерала указывается посредством «^^». Например:

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT $x
WHERE
{
   <http://example.org/item01> $x "123"^^<xsd:integer>.
}
```

На сегодняшний день литералом может быть только объект, но RDF Work Group рассматривает возможность того, чтобы субъект тоже мог быть литералом.

Переменные в SPARQL имеют *глобальную область действия*. Они отмечаются при помощи «*$*» или «*?*». Причем, «*$a*» и «*?a*» – это одно и то же.

Blank nodes («промежуточные узлы») – пришли из RDF. В приведенном ниже примере RDF/XML, blank node обозначается посредством атрибута rdf:nodeID:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:dc="http://purl.org/dc/elements/1.1/"
       xmlns:ex="http://example.org/stuff/1.0/">
  <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar"
                   dc:title="RDF/XML Syntax Specification (Revised)">
    <ex:editor rdf:nodeID="abc"/>
  </rdf:Description>

  <rdf:Description rdf:nodeID="abc"
             ex:fullName="Dave Beckett">
```

```
    <ex:homePage rdf:resource="http://purl.org/net/dajobe/"/>
  </rdf:Description>
</rdf:RDF>
```

Промежуточный узел – это локальный ресурс, который не виден за пределами документа. Существует два способа разметки промежуточных узлов (в SPARQL):

- посредством символа подчеркивания _:b57 :p "v".
- посредством прямоугольных скобок :x :q [:p "v"].

Все четыре SPARQL запроса, представленные в табл. 8.1, по сути являются одинаковыми. Обратите внимание на синтаксис.

Таблица 8.1

Разные по форме, но идентичные по сути SPARQL запросы

PREFIX dc: <http://purl.org/dc/elements/1.1/> SELECT ?title WHERE { <http://example.org/book/book1> dc:title ?title }	PREFIX dc: <http://purl.org/dc/elements/1.1/> PREFIX : <http://example.org/book/> SELECT *$title* WHERE { :book1 dc:title *$title* }
BASE <http://example.org/book/> PREFIX dc: <http://purl.org/dc/elements/1.1/> SELECT *$title* WHERE { <book1> dc:title *?title* }	BASE <http://example.org/book/> PREFIX dcore: <http://purl.org/dc/elements/1.1/> SELECT *$title* WHERE { <book1> dcore:title *?title* }

Dublin Core (http://purl.org/dc/elements/1.1/) принято записывать как dc. Но можно сделать запрос, как показано в последнй ячейке табл. 8.1. Вообще, не имеет значения как называть пространства имен (в смысле не обязательно называть их так же, как и в других документах). То есть в данных префикс может быть «dc», а в запросе «dcore» и наоборот.

Формат входных данных

Помимо RDF/XML существуют и другие языки, реализующие RDF Concept. В спецификации SPARQL данные в большинстве случаев описывались с помощью формата Turtle. Он позволяет быстро и наглядно записывать триплеты. Вот пример его использования:

```
@prefix dc:   <http://purl.org/dc/elements/1.1/>.
@prefix :     <http://example.org/book/>.
:book1  dc:title  "SPARQL Tutorial".
:book2  dc:title  "SPARQL вторая книга, типа того".
```

Этот формат, естественно, поддерживается и в ARQ endpoint. Кстати, если запрос из таблицы применить к представленным выше данным, ARQ выдает результат:

title
"SPARQL Tutorial"

Формат результата

Для наглядности все результаты запроса здесь приводятся в виде текста. На практике, в большинстве случаев, это будет не текст, а XML. Подробно о формате результата SPARQL запроса в виде XML читайте в [48]. Практически все серверы, обслуживающие точки доступа предоставляют широкие возможности по выбору способа представления результата. Так, например, DBPedia endpoint предлагает следующий список: HTML, Spreadsheet, XML, JSON, Javascript, NTriples, RDF/XML,CSV и TSV.

Данные с применением сокращенной формы записи триплетов и промежуточных узлов _:a и _:b

```
@prefix foaf:    <http://xmlns.com/foaf/0.1/>.
_:a  foaf:name   "Johnny Lee Outlaw".
     foaf:mbox   <mailto:outlaw@example.com>.
_:b  foaf:name   "A. N. Other".
     foaf:mbox   <mailto:other@example.com>.
```

Рассмотрим примеры SPARQL-запросов с применением сокращенной формы записи шаблонов триплетов:

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?mbox
WHERE
  { ?x foaf:name "Johnny Lee Outlaw".
      foaf:mbox ?mbox }
```

Результат:

mbox
<mailto:outlaw@example.com>

Данные:

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/>.
_:a  foaf:name   "Johnny Lee Outlaw".
_:a  foaf:mbox   <mailto:jlow@example.com.
_:b  foaf:name   "Peter Goodguy".
_:b  foaf:mbox   <mailto:peter@example.org>.
```

SPARQL запрос:

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
  { ?x foaf:name ?name.
    ?x foaf:mbox ?mbox }
```

Результат (в таблице две колонки: name и mbox):

name	mbox
"Peter Goodguy"	<mailto:peter@example.org>
"Johnny Lee Outlaw"	<mailto:jlow@example.com>

Данные:

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/>.
_:a  foaf:name   "Alice".
_:b  foaf:name   "Bob".
```

SPARQL запрос:

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?x ?name
WHERE  { ?x foaf:name ?name }
```

Результат:

x	name
_:b0	"Bob"
_:b1	"Alice"

Здесь следует обратить внимание на то, как промежуточные узлы отображаются в результате. Если в данных они имели одно имя, в результате могут иметь совершенно другое.

Классы

Ключевое слово «a» используется как предикат в шаблонах триплетов, и является альтернативой rdf:type (http://www.w3.org/1999/02/22-rdf-syntax-ns#type):

```
?x  a  :Class1.
```

то же самое, что и

```
?x    rdf:type  :Class1.
```

В связи с появлением языка OWL использование классов (при помощи предиката rdf:type или ключевого слова «a») становится все более употребимым. Рассмотрим пример Turtle документа:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dc:  <http://purl.org/dc/elements/1.1/>.
@prefix :    <http://example/ns#>.

_:a   rdf:subject   <http://example.org/book/book1>.
_:a   rdf:predicate dc:title.
_:a   rdf:object    "SPARQL".
_:a   :saidBy       "Alice".

_:b   rdf:subject   <http://example.org/book/book1>.
_:b   rdf:predicate dc:title.
_:b   rdf:object    "SPARQL Tutorial".
_:b   :saidBy       "Bob".
```

Здесь определены предикаты rdf:subject, rdf:predicate и rdf:object, которые описывают факты:

```
# "Alice" сказала:
<http://example.org/book/book1> dc:title "SPARQL".
# A "Bob" сказал:
<http://example.org/book/book1> dc:title "SPARQL Tutorial".
```

Но если мы пошлем SPARQL запрос

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?book ?title
WHERE
{ ?book dc:title ?title }
```

к предыдущему Turtle документу, то результат будет пустой, потому что в действительности в RDF-графе нет ни одного триплета, где dc:title являлся бы предикатом. Есть только два триплета, где он выступает в роли объекта:

```
_:a   rdf:predicate dc:title.
_:b   rdf:predicate dc:title.
```

Запрос к подобным документам должен выглядеть примерно так:

```
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dc:   <http://purl.org/dc/elements/1.1/>
PREFIX :     <http://example/ns#>

SELECT ?book ?title
WHERE
{
   ?t rdf:subject    ?book.
   ?t rdf:predicate  dc:title.
   ?t rdf:object     ?title.
# если мы хотим узнать, – что сказал "Bob"
   ?t :saidBy       "Bob".
}
```

Результат:

book	title
<http://example.org/book/book1>	"SPARQL Tutorial"

Шаблоны триплетов

Классическая форма шаблона триплетов записывается в виде списка, элементом которого является последовательность субъект предикат объект, в конце которой ставится точка «.»

```
?x    foaf:name  ?name.
?y   ?property    ?mbox.
?y    foaf:name  "Миша".
```

Причем, и вместо субъекта, и вместо предиката, и вместо объекта, – может стоять переменная.

Как в Turtle, так и в SPARQL триплеты можно представлять и в других синтаксических формах, например как список из предикатов и объектов, с использованием «;»:

```
?x  foaf:name  ?name;
    foaf:mbox  ?mbox.
```

тоже самое, что и

```
?x  foaf:name  ?name.
?x  foaf:mbox  ?mbox.
```

Можно использовать список из объектов, перечисляя их через «,»:

```
?x  foaf:nick  "Alice" , "Alice_".
```

тоже самое, что и

```
?x  foaf:nick  "Alice".
?x  foaf:nick  "Alice_".
```

Последние две синтаксические формы можно комбинировать:

```
?x  foaf:name  ?name;
    foaf:nick  "Alice" , "Alice_".
```

тоже самое, что и

```
?x  foaf:name  ?name.
?x  foaf:nick  "Alice".
?x  foaf:nick  "Alice_".
```

Шаблон группового графа в SPARQL обозначается при помощи фигурных скобок «{}»:

```
PREFIX foaf:
SELECT ?name ?mbox
WHERE {
      ?x foaf:name ?name.
      ?x foaf:mbox ?mbox.
    }
```

Можно представить его в виде 2 графов (каждый по одному триплету):

```
PREFIX foaf:
SELECT ?name ?mbox
WHERE {
      { ?x foaf:name ?name. }
      { ?x foaf:mbox ?mbox. }
    }
```

Ключевое слово OPTIONAL использует следующий синтаксис:

```
pattern OPTIONAL { pattern }.
```

Приведем ***пример***. Данные:

```
@prefix foaf:     <http://xmlns.com/foaf/0.1/>.
@prefix rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
_:a  rdf:type     foaf:Person.
_:a  foaf:name    "Alice".
_:a  foaf:mbox    <mailto:alice@example.com>.
_:a  foaf:mbox    <mailto:alice@work.example>.
_:b  rdf:type     foaf:Person.
_:b  foaf:name    "Bob".
```

SPARQL запрос, использующий необязательный шаблон (применяя ключевое слово OPTIONAL):

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  { ?x foaf:name  ?name.
       OPTIONAL { ?x  foaf:mbox  ?mbox }
      }
```

И результат:

name	mbox
"Bob"	
"Alice"	<mailto:alice@work.example>
"Alice"	<mailto:alice@example.com>

Если бы мы не использовали ключевое слово OPTIONAL, то «Bob» вообще бы не попал в результат.

Шаблоны в SPARQL могут объединяться с помощью ключевого слова UNION. Рассмотрим ***пример***.

Данные:

```
@prefix dc10:  <http://purl.org/dc/elements/1.0/>.
@prefix dc11:  <http://purl.org/dc/elements/1.1/>.
_:a  dc10:title    "SPARQL Query Language Tutorial".
_:b  dc11:title    "SPARQL Protocol Tutorial".
_:c  dc10:title    "SPARQL".
_:c  dc11:title    "SPARQL (updated)".
```

SPARQL запрос:

```
PREFIX dc10:  <http://purl.org/dc/elements/1.0/>
PREFIX dc11:  <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE  { { ?book dc10:title  ?title } UNION { ?book dc11:title  ?title } }
```

Результат:

title
"SPARQL"
"SPARQL Query Language Tutorial"
"SPARQL (updated) "
"SPARQL Protocol Tutorial"

Можно немного изменить запрос:

```
PREFIX dc10:  <http://purl.org/dc/elements/1.0/>
PREFIX dc11:  <http://purl.org/dc/elements/1.1/>
SELECT ?title10 ?title11
WHERE  { { ?book dc10:title  ?title10 } UNION { ?book dc11:title  ?title11 } }
```

И результат будет представлен в более наглядной форме:

title10	title11
"SPARQL"	
"SPARQL Query Language Tutorial"	
	"SPARQL (updated) "
	"SPARQL Protocol Tutorial"

Набор данных RDF

RDF выражает информацию в виде графа, который представляется в виде триплетов из субъектов, предикатов и объектов. Однако в большинстве случаев массив данных RDF содержит множество графов. Поэтому приложение должно иметь возможность сделать запрос, который затрагивает информацию из более, чем одного графа. В наборе данных может быть только один граф, который не имеет имени – неименованный граф. Все остальные графы (их может быть несколько, а может и не быть вовсе) являются именованными, их имя задается посредством URI.

Набор данных RDF – это множество:

$$\{ G, (<u_1>, G_1), (<u_2>, G_2),\ldots (<u_n>, G_n) \},$$

где G, G_i – графы, $<u_i>$ – URI (причем, каждый $<u_i>$ является уникальным), $i \in 1..n$.

G называется неименованным графом (default graph), $(<u_i>, G_i)$ называется именованным графом (named graph). Именованных графов может и не быть.

Пример:

```
# Неименованный граф
@prefix dc: <http://purl.org/dc/elements/1.1/>.

<http://example.org/bob>    dc:publisher  "Bob".
<http://example.org/alice>  dc:publisher  "Alice".
```

```
# Именованный граф: http://example.org/bob
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

_:a foaf:name "Bob".
_:a foaf:mbox <mailto:bob@oldcorp.example.org>.

# Именованный граф: http://example.org/alice
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
_:a foaf:name "Alice".
_:a foaf:mbox <mailto:alice@work.example.org>.
```

В этом примере неименованный граф содержит URI именованных графов. Триплеты неименованных графов *не располагаются* непосредственно в именованном графе.

Запросы к наборам данных RDF

Чтобы применить шаблоны к именованным графам используется ключевое слово GRAPH вместе с URI именованного графа или переменной.

В следующих примерах будем использовать два именованных графа:

```
# Named graph: file:///work/aliceFoaf.ttl
@prefix  foaf:  <http://xmlns.com/foaf/0.1/>.
@prefix  rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix  rdfs:  <http://www.w3.org/2000/01/rdf-schema#>.
_:a  foaf:name    "Alice".
_:a  foaf:mbox    <mailto:alice@work.example>.
_:a  foaf:knows   _:b.
_:b  foaf:name    "Bob".
_:b  foaf:mbox    <mailto:bob@work.example>.
_:b  foaf:nick    "Bobby".
_:b  rdfs:seeAlso  <http://example.org/foaf/bobFoaf>.
<http://example.org/foaf/bobFoaf>
    rdf:type     foaf:PersonalProfileDocument.
# Named graph: file:///work/bobFoaf.ttl
@prefix  foaf:  <http://xmlns.com/foaf/0.1/>.
@prefix  rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix  rdfs:  <http://www.w3.org/2000/01/rdf-schema#>.
_:z  foaf:mbox    <mailto:bob@work.example>.
_:z  rdfs:seeAlso  <http://example.org/foaf/bobFoaf>.
_:z  foaf:nick    "Robert".
```

Создадим SPARQL-запрос, который позволит нам узнать какой у Боба ник по мнению Алисы (именованый граф file:///work/aliceFoaf.ttl) и по мнению самого Боба (именованый граф file:///work/bobFoaf.ttl):

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?src ?bobNick
WHERE
 {
```

```
  GRAPH ?src
  { ?x foaf:mbox <mailto:bob@work.example>.
    ?x foaf:nick ?bobNick
  }
}
```

Результат:

src	bobNick
<file:///work/aliceFoaf.ttl>	"Bobby"
<file:///work/bobFoaf.ttl>	"Robert"

Изменим запрос, так чтобы он применял шаблон только к графу file:///work/bobFoaf.ttl:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX local: <file:///d:/bin/Arq-1.3/work/>
SELECT ?bobNick
WHERE
 {
  GRAPH local:bobFoaf.ttl
  {
     ?x foaf:mbox <mailto:bob@work.example>.
     ?x foaf:nick ?bobNick
  }
 }
```

и получаем результат:

bobNick
"Robert"

Ключевое слово GRAPH (с помощью него формируют запрос, который применяет шаблон к именованым графам) можно комбинировать с обычными шаболнами (запрос к неименованному графу). Например так:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX : <http://somesite.org/>
SELECT ?src ?bobNick
WHERE
 {
  ?x :somePredicate "someliteral".
  GRAPH ?src
  {
     ?x foaf:mbox <mailto:bob@work.example>.
     ?x foaf:nick ?bobNick
  }
 }
```

Модификаторы представления результатов запросов

В SPARQL используется пять модификаторов представления результатов запросов:

- *Projection*;
- *Distinct* (используя ключевое слово DISTINCT);
- *Order* (используя ключевые слова ORDER BY, ASC и DESC);
- *Limit* (используя ключевое слово LIMIT);
- *Offset* (используя ключевое слово OFFSET).

Projection

Projection – самый простой модификатор представления результатов запросов (не имеет специального ключевого слова). Он просто указывает значения каких переменных запроса должны быть представлены в результате. Перечисление переменных осуществляется в директиве SELECT (через запятую или пробел):

```
SELECT $x,$y
```

приведет к результату в виде таблицы из двух столбцов, поименованных $x и $y.

Distinct

Использование ключевого слова DISTINCT указывает, что *каждая строка* в ответе на запрос должна быть уникальной.

Пример:

```
# Исходный набор данных x.x
# http://localhost/specialists.ttl

@prefix sp: <http://localhost/properties/>.
@prefix   : <http://localhost/>.
:Misha sp:name "Миша" ;
        sp:profession "программист" ;
        sp:city "Сочи".
:Natasha sp:name "Наташа" ;
          sp:profession "веб дизайнер" ;
          sp:city "Санкт-Петербург".
:Lena sp:name "Лена" ;
       sp:profession "веб дизайнер" ;
       sp:city "Сочи".
:Ilia sp:name "Илья" ;
    sp:profession "программист" ;
    sp:city "Сочи".
```

Теперь построим SPARQL-запрос, который позволит нам понять, какие специалисты в каком городе работают:

```
BASE <http://localhost/>
PREFIX sp: <http://localhost/properties/>
SELECT $city $profession
FROM <http://localhost/specialists.ttl>
WHERE
{
    $a sp:profession $profession ;
        sp:city $city.
}
```

Получаем результат:

city	profession
"Сочи"	"программист"
"Сочи"	"веб дизайнер"
"Санкт-Петербург "	"веб дизайнер"
"Сочи"	"программист"

Первая и последняя строчка одинаковы. Чтобы не было совпадающих строк, можно в запросе использовать ключевое слово DISTINCT:

```
BASE <http://localhost/>
PREFIX sp: <http://localhost/properties/>
SELECT DISTINCT $city $profession
FROM <http://localhost/specialists.ttl>
WHERE
{
    $a sp:profession $profession ;
      sp:city $city.
}
```

Получаем результат:

city	profession
"Сочи"	"программист"
"Сочи"	"веб дизайнер"
"Санкт-Петербург"	"веб дизайнер"

Order

С помощью модификатора представления результатов запросов Order результат можно отсортировать по возрастанию или убыванию значения любой переменной.

Синтаксис:

ORDER BY $param_1$ $param_2$...,

где $param_i$ – это *$x* или ASC(*$x*) или DESC(*$x*), где *$x* – некоторая переменная.

Причем, в данном контексте: *$x* – это короткая запись для ASC(*$x*). ASC (*$x*) (англ. *ascending* – возрастать) – сортирует результат по возрастанию переменной *$x*. DESC (*$x*) (англ. *descending* – убывать) – сортирует результат по убыванию переменной *$x*.

Допустим, мы хотим отсортировать города по возрастанию, а профессии по убыванию. Сортировка по убыванию и возрастанию выполняется путем вычисления отношения «<», а оно, как известно, определено и на строках – сравниваются ASCII коды символов. Итак, строим запрос:

```
BASE <http://localhost/>
PREFIX sp: <http://localhost/properties/>
SELECT DISTINCT $city $profession
FROM <http://localhost/specialists.ttl>
WHERE
{
    $a sp:profession $profession ;
      sp:city $city.
}
ORDER BY $city DESC($profession)
```

И получаем желаемый результат:

city	profession
"Санкт-Петербург"	"веб дизайнер"
"Сочи"	"программист"
"Сочи"	"веб дизайнер"

Limit

Ключевое слово LIMIT *n* задает максимальное значение количества строк, которые будут в результате.

Пример:

```
BASE <http://localhost/>
PREFIX sp: <http://localhost/properties/>
SELECT DISTINCT $city $profession
FROM <http://localhost/specialists.ttl>
WHERE
{
    $a sp:profession $profession ;
      sp:city $city.
}
ORDER BY $city DESC($profession)
LIMIT 2
```

Результат (только две строки):

city	profession
"Санкт-Петербург"	"веб дизайнер"
"Сочи"	"программист"

Offset

OFFSET *n* позволяет *не показывать* в результате первые *n* строк.

В предыдущем примере мы получили в результате – 2 решения. Допустим, мы хотим получить еще решения, но при этом не хотим, чтобы их было больше 3. Также не хотим получать уже известные нам решения. Тогда необходимо выполнить следующий SPARQL запрос:

```
BASE <http://localhost/>
PREFIX sp: <http://localhost/properties/>
SELECT DISTINCT $city $profession
FROM <http://localhost/specialists.ttl>
WHERE
{
    $a sp:profession $profession ;
      sp:city $city.
}
ORDER BY $city DESC($profession)
LIMIT 3
OFFSET 2
```

Результат:

city	profession
"Сочи"	"веб дизайнер"

Формы результата

SPARQL имеет четыре формы результата:

- SELECT – возвращает все или часть переменных, связанных в шаблонах запроса;
- CONSTRUCT – возвращает RDF граф, построенный путем замещения переменных в Construct-шаблоне запроса;
- DESCRIBE – возвращает RDF граф, который описывает найденный результат;
- ASK – возвращает true, если хоть что-то найдено, иначе – false.

Если форма результата SELECT или ASK, то результат запроса будет в формате текст или XML. Если форма результата CONSTRUCT или DESCRIBE, то результат запроса будет в формате RDF.

Select

Синтаксис с модификатором Projection:

```
SELECT $a $b...
```

Форма результата SELECT просто возвращает значения переменных (*$a $b*...) из всех найденных результатов (строк).

Синтаксис без модификатора Projection:

SELECT *

возвращает значения *всех* переменных из всех найденных результатов (строк).

Construct

Синтаксис:

```
CONSTRUCT
{
   pattern
}
```

где pattern – некоторый шаблон.

Форма результата Construct возвращает RDF граф, который задается специальным шаблоном. То есть для каждого очередного решения в Construct-шаблон вместо переменных подставляются uri, промежуточные узлы или литералы. Если такую подстановку осуществить невозможно (например, – в триплете, в котором субъектом или предикатом является литерал), то триплет, естественно, не включается в результирующий граф.

Пример. SPARQL запрос (к данным x.x):

```
# http://localhost/newgraph.rq
BASE <http://localhost/>
PREFIX uv: <http://ulskforever.org/vocabulary/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX sp: <http://localhost/properties/>
CONSTRUCT
{
   $person foaf:firstName $name ;
         uv:name $name.
}
FROM <http://localhost/specialists.ttl>
WHERE
{
   $person sp:name $name.
}
```

Этим запросом мы хотим создать граф, в котором вместо свойства sp:name, которое определено на нашем локальном сервере будут использованы аналогичные свойство FOAF (friend of a friend) foaf:firstName и свойство некоего словаря uv:name. Если выбрать представление результата в виде N3, то получим результат:

```
@prefix sp:     <http://localhost/properties/>.
@prefix uv:     <http://ulskforever.org/vocabulary/>.
@prefix foaf:   <http://xmlns.com/foaf/0.1/>.
@prefix :       <http://localhost/>.
```

```
:Misha
    uv:name      "Миша" ;
    foaf:firstName  "Misha".
:Ilia
    uv:name      "Илья" ;
    foaf:firstName  "Ilia".
:Lena
    uv:name      "Лена" ;
    foaf:firstName  "Lena".
:Natasha
    uv:name      "Наташа" ;
    foaf:firstName  "Natasha".
```

Describe

Синтаксис:

DESCRIBE param1 param2..., где $param_i$ – это *переменная URI.*

Форма результата Describe возвращает RDF граф, который *описывает* ресурсы $resource_i$, где $resource_i$ – это $param_i$, если $param_i$ – URI значение $param_i$, взятое из *очередного решения*, если $param_i$ – переменная.

Пример:

```
# http://localhost/newgraph.rq
BASE <http://localhost/>
PREFIX sp: <http://localhost/properties/>
DESCRIBE $person <Natasha>
WHERE
{
   $person sp:name $name ;
            sp:profession "программист"@ru.
}
```

Результат:

```
@prefix sp:     <http://localhost/properties/>.
@prefix :       <http://localhost/>.
:Misha sp:name "Миша" ;
        sp:profession "программист" ;
        sp:city "Сочи".
:Ilia sp:name "Илья" ;
    sp:profession "программист" ;
    sp:city "Сочи".
:Natasha sp:name "Наташа" ;
          sp:profession "веб дизайнер" ;
          sp:city "Санкт-Петербург".
```

Результирующий граф описывает три ресурса:

```
<http://localhost/Misha>
<http://localhost/Ilia>
<http://localhost/Natasha>
```

Причем, первые два URI взяты из набора решений, а <http://localhost/Natasha> взято непосредственно из самого запроса.

Ask

Принцип действия SPARQL-процессора при выполнении Ask запроса простой: если находится хотя бы одно решение, то в результате возвращается – true, иначе – false.

Пример

```
# http://localhost/ask.rq
BASE <http://localhost/>
PREFIX sp: <http://localhost/properties/>
ASK
{
   $person sp:name "Наташа" ;
           sp:city "Санкт-Петербург".
}
```

Результат: Ask => Yes.

Практические примеры

В заключение приведем три примера SPARQL запросов через точку доступа веба. В качестве редактора запросов используем Virtuoso SPARQL Query Editor, расположенный в точке доступа по адресу http://dbpedia.org/sparql.

Пример 1

В поле Query Text составим запрос на получение информации обо всех англоязычных странах и их населении. Упорядочим результат по числу жителей этих стран. Поскольку число строк будет значительным, ограничим результат пятью строками:

```
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX prop: <http://dbpedia.org/property/>

SELECT ?place ?population WHERE {
   ?place rdf:type    yago:English-speakingCountriesAndTerritories.
   ?place rdf:type    dbpedia-owl:Country;
          prop:populationEstimate   ?population .
}
ORDER BY DESC(?population)
LIMIT 5
```

Запускаем запрос на исполнение (Run Query) и получаем результат:

place	population
http://dbpedia.org/resource/Pakistan	180440005
http://dbpedia.org/resource/Nigeria	170123740
http://dbpedia.org/resource/Kenya	43500000
http://dbpedia.org/resource/Ghana	24200000
http://dbpedia.org/resource/Australia	22859335

Пример 2

В поле Query Text составим запрос на получение информации о кошках. Поскольку большинство сведений описательного характера в DBpedia приведены на разных языках, используем фильтр, который определяет русский язык для вывода содержимого переменной ?Cat:

```
PREFIX prop:<http://dbpedia.org/property/>
PREFIX bpedia-owl:<http://dbpedia.org/ontology/>
SELECT ?Cat
{
  <http://dbpedia.org/resource/Cat> dbpedia-owl:abstract ?Cat
  FILTER(langMatches(lang(?Cat), "RU"))
}
```

Запускаем запрос на исполнение (Run Query) и получаем результат:

Cat
"Ко́шка, или дома́шняя ко́шка – домашнее животное, одно из наиболее популярных «животных-компаньонов». С зоологической точки зрения, домашняя кошка – млекопитающее семейства кошачьих отряда хищных. Ранее домашнюю кошку нередко рассматривали как отдельный биологический вид. С точки зрения современной биологической систематики домашняя кошка (Felis silvestris catus) является подвидом лесной кошки. Являясь одиночным охотником на грызунов и других мелких животных, кошка – социальное животное, использующее для общения широкий диапазон звуковых сигналов, а также феромоны и движения тела. В настоящее время в мире насчитывается около 600 млн домашних кошек, выведено около 256 пород, от длинношерстных до лишенных шерсти, признанных и зарегистрированных различными фелинологическими организациями. На протяжении 10 000 лет кошки ценятся человеком, в том числе за способность охотиться на грызунов и других домашних вредителей. "@ru

Пример 3

Изменим запрос примера 1, заменив ограничение количества строк результата на фильтрацию результатов по числу жителей англоговорящих стран. Для записи условия фильтрации значение 4,35 миллиона жителей возьмем из результирующей таблицы примера 1.

```
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX prop: <http://dbpedia.org/property/>

SELECT ?place ?population WHERE {
```

```
    ?place rdf:type    yago:English-speakingCountriesAndTerritories.
    ?place rdf:type    dbpedia-owl:Country;
            prop:populationEstimate   ?population.
FILTER(?population>=43500000)
}
ORDER BY DESC(?population)
```

Результат запроса достаточно очевиден:

place	population
http://dbpedia.org/resource/Pakistan	180440005
http://dbpedia.org/resource/Nigeria	170123740
http://dbpedia.org/resource/Kenya	43500000

Последние две строки результирующей таблицы примера 1 не вошли в результат, так как не удовлетворили условию фильтрации.

Список литературы

1. Анализ и синтез систем управления. Теория. Методы. Примеры решения типовых задач с использованием персонального компьютера / Д. Х. Имаев, З. Ковальски, Н. Н. Кузьмин [и др.]. – Гданьск ; СПб. ; Сургут ; Томск, 1997.

2. Батищев, Д. А. Генетические алгоритмы решения экстремальных задач / Д. А. Батищев. – Воронеж : Изд-во ВГТУ, 1995.

3. Белов, Ю. В. Индустриальные средства проектирования и оценки эффективности программных систем, работающих в реальном времени / Ю. В. Белов, В. С. Проценко, В. В. Федоров, А. А. Хижняк // Вычисл. системы и вопр. Принятия решений. – М., 2001.

4. Бэлэни, Н. Будущее Web – за семантикой / Н. Бэлэни. – 2005, http:// www.iso.ru/rus/document6130.phtml

5. Бэлэни, Н. Будущее Web – за семантикой / Н. Бэлэни. http://www.iso.ru/ rus/document6130. phtml#2

6. Божко, А. Н. Структурный синтез на элементах с ограниченной сочетаемостью / А. Н. Божко, А. Ч. Толпаров. http: // technonew. developer. stack.net / doc/ 44191.html.

7. Бодянский, Е. В. Искусственные нейронные сети: архитектуры, обучение, применения / Е. В. Бодянский, О. Г. Руденко. – Харьков : ХГПУ, 2008.

8. Бодянский, Е. В. Искусственные нейронные сети : архитектуры, обучение, применения / Е. В. Бодянский, О. Г. Руденко. – Харьков : Телетех, 2004. – 372 с.

9. Валидатор разметки Яндекса http://webmaster.yandex.ru/ microtest.xml.

10. Валидатор разметки Google http://www.google.com/webmasters/tools/ richsnippets.

11. Гладков, Л. А. Генетические алгоритмы / Л. А. Гладков, В. В. Курейчик, В. М. Курейчик // под. ред. В. М. Курейчика. – 2-е изд., испр. и доп. – М. : Физматлит, 2010. – 386 с.

12. Джонс, М. Т. Программирование искусственного интеллекта в приложениях / М. Т. Джонс. – М. : ДМК Пресс, 2004.

13. Итоги науки и техники: физические и математические модели нейронных сетей. – Т. 1. – М. : Изд. ВИНИТИ, 1990.

14. Короткий, С. Нейронные сети: алгоритм обратного распространения / С. Короткий // http://ииклуб.рф/neur-2.html

15. Короткий, С. Нейронные сети: обучение без учителя / С. Короткий // http:// www.neuronos.ru / Нейронные-сети-обучение-без-учителя.

16. Кошелев, В. А. Некоторые задачи синтеза оптимальных модульных СОД РВ / В. А. Кошелев // Теоретические и прикладные задачи оптимизации. – М. : Наука, 1995.

17. Кротюк, Ю. М. Синтез оптимальных модульных СОД РВ с относительными приоритетами / Ю. М. Кротюк, В. А. Кошелев // Вопросы кибернетика. Автоматизация проектирования систем обработки данных. – М. : Научный совет комплексной проблеме «Кибернетика», 1985.

18. Круг, П. Г. Нейронные сети и нейрокомпьютеры: учеб. пособие по курсу «Микропроцессоры» / П. Г. Круг. – М. : Изд-во МЭИ, 2002. – 176 с.

19. Круглов, В. В. Искусственные нейронные сети. Теория и практика / В. В. Круглов, В. В. Борисов. – М. : Горячая линия – Телеком, 2001. – 382 с.

20. Кутузов, О. И. Моделирование телекоммуникационных сетей / О. И. Кутузов, Т. М. Татарникова ; СПбГУТ. – СПб., 2001. – 76 с.

21. Мюллер, И. Эвристические методы в инженерных разработках / И. Мюллер. – М. : Радио и связь, 1984.

22. Нестеренко, В. Д. Управление инфокоммуникационными сетями / В. Д. Нестеренко. – СПб. : Политехника, 2007. – 249 с.

23. Обогащенная структурная разметка для web-документов, W3C рабочий проект, 2011.

24. Осовский, С. Нейронные сети для обработки информации / С. Осовский // пер. с пол. И. Д. Рудинского. – М. : Финансы и статистика, 2002. – 344 с.

25. Растригин, Л. А. Системы экстремального управления / Л. А. Растригин. – М. : Наука, 1974. – 632 с.

26. Тауберер, Дж. Краткое введение в RDF / Дж. Тауберер. – 2006, http://xmlhack.ru/texts/ 06/rdf-quickintro/rdf-quickintro.html

27. Тауберер, Дж. Краткое введение в RDF / Дж. Таубер. – http:// rdfabout.com/quickintro.xpd

28. Уоссермен, Ф. Нейрокомпьютерная техника / Ф. Уоссермен. – М. : Мир, 1992.

29. Хартьян, Д. Ю. Результаты корреляционного анализа совокупности параметров инфокоммуникационной инфраструктуры / Д. Ю. Хартьян, В. А. Шапцев // Матер. 8-й междунар. науч.-практ. конф. «Связь-2004: Проблемы функционирования информационных сетей». Новосибирск, 26–30 июня 2004 г. Новосибирск : ЗАО РИЦ Прайс Курьер, 2004. – Т. 2. – С. 298–306.

30. Химмельблау, Д. Прикладное нелинейное программирование / Д. Химмельблау. – М. : Мир, 1975.

31. Язык запросов SPARQL для RDF, рекомендация W3C. 2008,http:// www.w3.org/TR/rdf-sparql-query/

32. Artificial Neural Networks: Concepts and Theory // IEEE Computer Society Press. – 1992.

33. Creative Commons: About Licenses, http://creativecommons. org/about/ licenses/

34. Beckett, D. Turtle: Terse RDF Triple Language / D. Beckett, T. Berners-Lee. – January 2008. W3C Team Submission. http://www.w3.org /TeamSubmission/turtle/

35. Brickley, D. FOAF Vocabulary Specification 0.98 / D. Brickley, L. Miller. – August 2010, http://xmlns.com/foaf/spec/.

36. Brickley, D. RDF Vocabulary Description Language 1.0: RDF Schema / D. Brickley, V. G. Ramanathan. – http://www.w3.org/TR/2004/REC-rdf-schema-20040210

37. Dublin Core metadata initiative: Dublin Core metadata element set, http://purl.oclc.org/ docs/core/documents/rec-dces-19990702.htm

38. Dublin Core metadata initiative, http://purl.oclc.org/docs/core /documents/ rec-dces-19990702.htm

39. Google, Inc., Yahoo, Inc., Microsoft Corporation, 2011, http://schema.org/ docs/gs.

40. Hecht-Nielsen, R. Theory of the backpropagation neural network / R. Hecht-Nielsen // Int. Joint Conf. on Neural Networks. – Washington, DC, 1989. – № 1. – P. 593–606.

41. Hornik, K. Multilayer feedforward networks are universal approximators / K. Hornik, M. Stinchcombe, H. White // Neural Networks. – 1989. – № 2. – P. 359–366.

42. Hornik, K. Approximation capabilities of multiplayer feedforward networks / K. Hornik // Neural Networks. – 1991. – № 4. – P. 251–257.

43. Sporny, M. RDFa API Latest. W3C Working Draft / M. Sporny, B. Adriaan, M. Birbeck. – URL: http://www.w3.org/TR/RDFA-API/

44. Prokhorov, D. Adaptive Critic Designs / D. Prokhorov, D. Wunsch // IEEE Transactions on Neural Networks. – 1997. – Vol. 8, № 5. – P. 997–1007.

45. SPARQL Query Results XML Format http://www.w3.org/TR /rdf-sparql-XMLres/

46. The Open Graph Protocol. 2010, http://www.zencoder.pro/open-graph-protocol-opisanie-socialnih-media

47. http://www.w3.org/2001/sw/interest/

48. http://my-eventos.com/solution/ontoquad/

ПРИЛОЖЕНИЕ

Таблица П1

Программные продукты моделирования нейронных сетей

Название, разработчик/производитель	Платформа	Поддерживаемые парадигмы и алгоритмы обучения	Интерфейс	Комментарии
1	2	3	4	5
Matlab Neural Network Toolbox 7.0 MathWorks, США	Win XP, 7	Поддерживаемые парадигмы: персептрон, обратное распространение, радиальный базис, сети Эльмана, сети Хопфильда, вероятностная и обобщенная регрессии. Неподдерживаемые парадигмы: Хеббиана, Кохонена, карты свойств, самоорганизующиеся карты	GUI для Ms Windows	Генерирует ANSI-совместимый код
SNNS, Институт параллельных и распределенных систем (IPVR) при Штутгартском университете	Unix	Обратное распространение, радиальный базис, ART1, ART2, карты Кохонена, сети Джордана, сети Эльмана, ассоциативная память	GUI для X-Windows	Один из лучших симуляторов. Может работать с MS Windows при использовании эмулятора X-Windows. С программой поставляются исходные коды на C++
Trajan Software, Ltd., Великобритания http://www.traian-software.demon.co.uk/commerce.htm	Win XP, 7	Многоуровневый персептрон, обратное распространение, радиальный базис, карты Кохонена, вероятностная и обобщенная регрессии	GUI для MS Windows	Демо-версия программы доступна на сайте компании-производителя

Продолжение табл. П1

Название, разработчик/производитель	Платформа	Поддерживаемые парадигмы и алгоритмы обучения	Интерфейс	Комментарии
1	2	3	4	5
Delta, Artificial Intelligence Group, Франция (Департамент компьютерных наук) http://www.inf.enst.fr/~milc/dnns/dnns.us.html	HP-UX Sun OS 4.1	Многоуровневый персептрон, обратное распространение, сети Джордана, сети Ельцина, карты Кохонена	GUI для X-Windows	Демо-версия
X-SimIIC, Испания (Мадрид) http://www.iic.uam.es/xsim/Welcome.html	Unix Linux	Многослойный персептрон, обратное распространение, карты Кохонена	Режим командной строки для ОС типа Unix, GUI для X-Windows	
Brain Wave, Университет Куинсланд, США http://www2.psy.uq.edu.u/~brainwav/	Любая	Многослойный персептрон, обратное распространение, сети Хеббиана, карты Кохонена	Internet- браузер с поддержкой Java	Реализован в виде Java-апплета – может работать с любой операционной системой
VieNet2, Австрийский институт исследования проблем искусственного интеллекта www.ai.univie.ac.at/oefai/nn/tool.html	DOS Win Unix Linux	Многослойный персептрон, обратное распространение, сети Джордана, сети Эльмана, карты Кохонена, ассоциативная память	Формируется пользователем	Распространяется в форме исходных кодов, что позволяет активно использовать его для написания собственных программ
NeuroWindows, НейроПроект, Россия http ://www. neuroproi ect.ru/	Win	Обратное распространение, ассоциативная память, карты Кохонена		Библиотека динамической компоновки Visual Basic, C++ и Delphi

Продолжение табл. П1

Название, разработчик/производитель	Платформа	Поддерживаемые парадигмы и алгоритмы обучения	Интерфейс	Комментарии
1	2	3	4	5
Aspirin/MIGRAINES, Mitre Corp.	Unix	Обратное распространение	GUI для X-Windows	Сохраняет веса и вектор узлов нейронной сети на диске в доступном формате
Atree, Билл Армстронг, Университет г. Альберта, США	Dos , Unix	Адаптивные логические деревья	Режим командной строки для Unix-подобных ОС, окна для DOS	Демо-версия
Cnaps, Adaptive Solutions Inc.	SunOS	Обратное распространение, карты Кохонена (одномерные и двумерные), LVQ2 и частотно-чувствительное конкурентное обучение	GUI для X-Windows	Производительность при обучении по алгоритму обратного распространения 1
ICSIM, Международный институт компьютерных наук, Беркли, Калифорния, США	Unix	Предопределенные сети	Shell, GUI для X-Windows	Демо-версия
Neural Shell, Лаборатория SPANN, Департамент инженерной энергетики, Университет Огайо, США ftp://ftp.quanta. eng. ohio-state.edu/	Unix	Сети Хопфильда, сети Хемминга, обратное распространение, карты Кохонена, адаптивное медленное обратное распространение, частот-но-чувствительное конкурентное обучение	Режим командной строки, GUI для X- Windows и SUNTOOLS	
Neuron, Университет Дьюка, США	Unix	Трехмерная реконструированная пирамидальная ячейка, диффузия	GUI для X-Windows	
Sankom, Дортмундский университет, Германия	Unix	Карты Кохонена	Режим командной стоки Shell	

Окончание табл. П1

Название, разработчик/производитель	Платформа	Поддерживаемые парадигмы и алгоритмы обучения	Интерфейс	Комментарии
1	2	3	4	5
SOMPAK, SOM, Лаборатория компьютерных и информационных наук, Хельсинкский университет технологий	Unix, DOS	Самоорганизующиеся карты	Формируется пользователем	Демо-версия
Xerion, Университет Торонто, Департамент компьютерных наук, США ftp://ftp.cs.toronto.edu/pub/xerion	Unix	Обратное распространение, рекуррентное обратное распространение, машина Больцмана, теория среднего поля, манипуляция свободной энергией, жесткое и мягкое конкурентное обучение, карты Кохонена	GUI для X-Windows	
NETS COSMIC, Университет Джорджии, США service@cossack.cosmic.uga.edu	DOS, UNIX	Обратное распространение	Режим командной строки	

Белов Михаил Петрович
Филиппов Феликс Васильевич

ИНТЕЛЛЕКТУАЛИЗАЦИЯ ИНФОКОММУНИКАЦИОННЫХ СИСТЕМ

Часть 2

Учебное пособие

Редактор *Л. К. Паршина*

Компьютерная верстка *Н. А. Ефремовой*

План 2014–2015 гг., п. 43, б

Подписано к печати 10.11.2014
Объем 5,0 усл.-печ. л. Тираж 35 экз. Заказ 526

Редакционно-издательский центр СПбГУТ
191186 СПб., наб. р. Мойки, 61

Отпечатано в СПбГУТ