

группой людей, например программистов (англ. *pool of programmers*). Эта проблема известна как проблема омонимов<sup>1</sup>.

Использование URI не обязательно обеспечивает доступ к ресурсу. Однако в качестве идентификаторов ресурсов рекомендуется применять URL, которые возможно разыменовать (dereferencable URI). Разыменование URI позволяет пользователям получить либо сам ресурс (например, изображение), либо более подробное описание этого ресурса (например, описание человека)<sup>2</sup>. Все ресурсы в данной книге будут идентифицироваться с помощью URL. Использование URI представляет собой одно из ключевых проектных решений, лежащих в основе языка RDF. Именно эта идея позволяет имени любого ресурса сделать уникальным в глобальном масштабе. Использование подобной схемы именования ресурсов существенно снижает остроту проблемы омонимов, которая до настоящего момента являлась препятствием для представления распределенных данных.

## 2.2.2. Свойства

Свойства представляют собой особый вид ресурсов; они описывают отношения между другими ресурсами. Приведем примеры свойств: «является другом», «является автором» «расположен в». Как и все ресурсы, свойства идентифицируются с помощью URI. URL свойств, так же как и URL любых ресурсов, можно разыменовать, чтобы найти их описания.

## 2.2.3. Утверждения

Утверждения задают определенные свойства для определенных ресурсов. Утверждение представляет собой триплет вида «сущность – атрибут – значение», состоящий из ресурса, свойства и значения этого свойства соответственно. В качестве значения свойства может выступать некоторый ресурс или *литерал*. Литералы представляют собой атомарные величины, например числа, строки или даты. Для обозначения сущности в RDF-утверждении часто используется термин «субъект», а для обозначения значения – термин «объект».

<sup>1</sup> Омонимы – разные по значению, но одинаковые по звучанию и написанию единицы языка. Например, ключ – от двери, ключ – природный источник воды, ключ – музыкальный знак, ключ – инструмент, ключ – регистрационный номер программного обеспечения. – *Прим. перев.*

<sup>2</sup> Иными словами, разыменование URI – это предоставление доступа к ресурсу, идентифицируемому данным URI. – *Прим. перев.*



**Рис. 2.1** ♦ Представление RDF-утверждения в виде графа

Рассмотрим пример утверждения «Здание Барон Вэй находится в Амстердаме». На языке RDF оно может быть записано следующим образом:

```

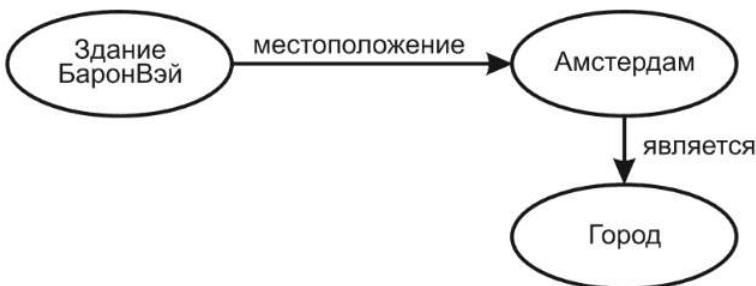
<http://www.semanticwebprimer.org/ontology/apartments.ttl#ЗданиеБаронВэй>1
<http://dbpedia.org/ontology/location>
<http://dbpedia.org/resource/Amsterdam>.
  
```

Обратите внимание, что для идентификации субъекта, свойства и объекта в данном утверждении были использованы URL.

## 2.2.4. Графы

Приведенное выше утверждение может быть представлено в виде *графа* (рис. 2.1). При этом для повышения читабельности в данном графе не используются URI.

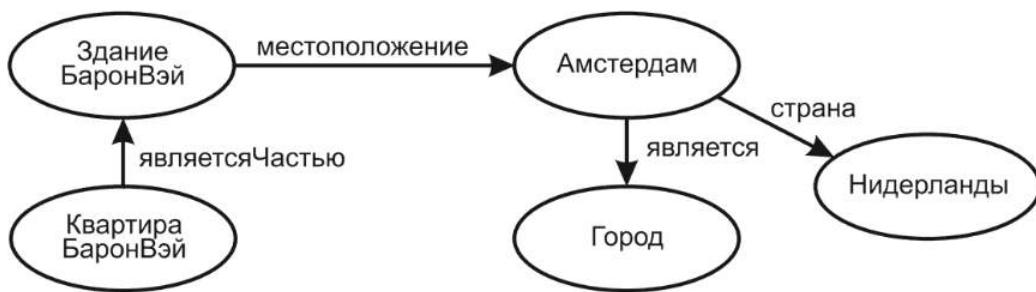
На рис. 2.1 представлен граф, в котором помеченные узлы соединены помеченными дугами. При этом любая дуга в графе направлена от субъекта утверждения к объекту утверждения, а метка на дуге представляет собой свойство утверждения. Метки узлов представляют собой идентификаторы субъектов и объектов утверждений. Объект утверждения может являться субъектом другого утверждения. Например, можно сделать утверждение, что «Амстердам является городом». Данное утверждение представлено в виде графа на рис. 2.2.



**Рис. 2.2** ♦ RDF-граф

<sup>1</sup> Для повышения степени усвоения материала русскоязычными читателями все названия объектов, свойств и субъектов RDF-утверждений в примерах приведены на русском языке. Это не относится к идентификаторам ресурсов из реально существующих в вебе наборов данных, например проекта DBpedia.org. – Прим. перев.

Данное графическое представление показывает, что RDF является графоориентированной моделью данных. Действительно, модель RDF напоминает так называемую семантическую сеть из теории искусственного интеллекта. Можно продолжить расширение графа, описывающего здание Барон Вэй. На рис. 2.3 изображена расширенная версия этого RDF-графа.



**Рис. 2.3** ♦ Расширенный RDF-граф

Важно отметить, что RDF-граф может быть создан в распределенном режиме несколькими различными участниками исключительно за счет использования URL. Это способствует созданию веба данных, который позволяет повторно использовать существующие знания. Например, если мы найдем в вебе описание города Амстердама в RDF-формате, то сможем повторно использовать эту информацию, только за счет указания URL данного ресурса. В целях создания глобального RDF-графа рекомендуется применять так называемые *принципы связанных данных* (Linked Data principles), которые способствуют повторному использованию и доступности информации:

- 1) применение URI в качестве имен сущностей;
- 2) использование HTTP URI для реализации возможности обращения пользователей к этим сущностям;
- 3) предоставление полезной информации тому, кто обращается по URI, с помощью стандартов (RDF);
- 4) включение ссылок на другие URI, позволяющие найти дополнительную информацию.

Несмотря на то что сама модель данных RDF не подразумевает следование этим принципам, их выполнение позволит нам получить большое преимущество – пользоваться знаниями других людей. Обратите внимание на то, как была повторно использована информация из проекта DBpedia.org в нашем примере. Вы можете перейти по указанным URL, чтобы получить больше информации о соответствующих ресурсах (<http://dbpedia.org/ontology/location> и <http://dbpedia.org/resource/Amsterdam>).

## 2.2.5. Выделение утверждений и графов

Иногда необходимо выделить конкретные утверждения и части графов, например для того, чтобы указать степень доверия к этим утверждениям или определить их источник. Предположим, нужно зафиксировать тот факт, что утверждение о местоположении здания Барон Вэй создано человеком – Фрэнком. Для этого язык RDF предоставляет два механизма.

Один из этих механизмов называется *реификация* (или материализация) утверждений. Основная идея реификации заключается во введении вспомогательного объекта, например объекта *Локальное Утверждение*, который связывается с каждой из трех частей заданного утверждения посредством RDF-свойств: *subject* (субъект), *predicate* (предикат) и *object* (объект). В предыдущем примере субъектом для *Локальное Утверждение* является *ЗданиеБаронВэй*, предикатом – местоположение, а объектом – *Амстердам*. Теперь мы можем указать *Локальное Утверждение* в качестве субъекта в другом триплете, который определяет его создателя. На рис. 2.4 изображен полученный таким образом график. Как и в предыдущем случае, для улучшения читабельности в графике не указаны полные URI.



**Рис. 2.4 ♦ Пример реификации**

Этот довольно громоздкий подход необходим, так как в языке RDF нет ничего, кроме триплетов, и мы не можем непосредственно создать идентификатор для триплета (тогда это был бы не триплет (тройка), а четверка). Из-за затрат вычислительных ресурсов, которых требует реификация, в новых версиях стандарта языка RDF было введено понятие «*именованный график*». При данном подходе утверждению или набору утверждений явным образом присваивается идентификатор (URL). Этот идентификатор может быть использован в обычных триплетах. Это более простой механизм для идентификации утверждений и графов. Говоря другими словами, *именованный график* позво-

ляет выделить множество RDF-утверждений и присвоить этому множеству идентификатор. В разделе 2.3.1.3 приводится пример записи рассмотренного выше реифицированного утверждения с помощью именованных графов.

## 2.2.6. Предикаты с несколькими аргументами

Триплет  $(x, P, y)$  можно рассматривать как логическую формулу  $P(x, y)$ , где  $P$  – бинарный предикат, который связывает объект  $x$  с объектом  $y$ . Фактически с помощью языка RDF можно записывать только *бинарные предикаты (свойства)*. Однако в некоторых случаях необходимы предикаты, которые имеют более двух аргументов. К счастью, такие предикаты могут быть смоделированы с помощью последовательности бинарных предикатов. Проиллюстрируем эту методику на примере предиката *брокер* с тремя аргументами. Интуитивное значение предиката *брокер*( $X, Y, Z$ ) заключается в следующем:  $X$  является брокером в сделке по продаже дома между продавцом  $Y$  и покупателем  $Z$ .

Введем новый вспомогательный ресурс *дом-на-продажу*<sup>1</sup> и три бинарных предиката *брокер*, *продавец* и *покупатель*. Тогда предикат *брокер*( $X, Y, Z$ ) может быть предоставлен следующим образом:

*брокер(дом-на-продажу, X),*  
*продавец(дом-на-продажу, Y),*  
*покупатель(дом-на-продажу, Z).*

Несмотря на то что предикаты с тремя аргументами записываются более компактно, использование бинарных предикатов позволяет упростить модель данных в целом.

## 2.3. Синтаксисы RDF

В предыдущем разделе был введен один из синтаксисов языка RDF, а именно графический синтаксис. Однако этот синтаксис не стандартизирован. Кроме того, он не пригоден для машинной интерпретации. В данном разделе вводятся стандартный машинно интерпретируемый синтаксис, называемый *Turtle*, а также дается краткий обзор нескольких других синтаксисов.

---

<sup>1</sup> Подобный вспомогательный ресурс может быть задан в RDF-утверждении как пустой узел (blank node). В соответствии со спецификацией языка RDF пустые узлы представляют собой множество, непересекающееся со множеством URI и литералов. Пустые узлы могут указываться в качестве субъекта и объекта RDF-триплета. – Прим. перев.

## 2.3.1. Turtle

Turtle (Terse RDF Triple Language, лаконичный язык RDF-триплетов) представляет собой текстовый синтаксис для языка RDF. Текстовые файлы, содержащие данные в формате Turtle, имеют расширение «.ttl». Выше уже был приведен пример утверждения, записанного с помощью синтаксиса Turtle:

```
<http://www.semanticwebprimer.org/ontology/apartments.ttl#зданиеБаронВэй>
<http://dbpedia.org/ontology/location>
<http://dbpedia.org/resource/Amsterdam>.
```

Адреса URL заключаются в угловые скобки. RDF-утверждение записывается в следующем порядке: субъект, свойство, объект – и заканчивается точкой. Таким образом можно записать целый RDF-граф:

```
<http://www.semanticwebprimer.org/ontology/apartments.ttl#>
  <http://www.semanticwebprimer.org/ontology/apartments.ttl#являетсяЧастью>
  <http://www.semanticwebprimer.org/ontology/apartments.ttl#зданиеБаронВэй>.
<http://www.semanticwebprimer.org/ontology/apartments.ttl#зданиеБаронВэй>
  <http://dbpedia.org/ontology/location>
  <http://dbpedia.org/resource/Amsterdam>.
```

### 2.3.1.1. Литералы

Как было показано выше, RDF-утверждения связывают ресурсы. Однако они могут содержать и литералы, то есть атомарные значения. В синтаксисе Turtle литералы записываются как некоторые значения, заключенные в кавычки, после которых указывается *тип данных*. Тип данных позволяет интерпретировать указанное значение как строку, дату, целое число или как величину некоторых других типов. Типы данных, так же как и ресурсы, идентифицируются с помощью URL. Рекомендуется использовать типы данных, определенные с помощью языка XML Schema. При применении этих типов данных указанные значения должны соответствовать определению языка XML Schema. Если для литерала тип данных не задан, то предполагается, что тип этого литерала – строковый. Покажем, как некоторые общие типы данных записываются в синтаксисе Turtle:

```
строки (string) - "Барон Вэй"
целые числа (integer) - "1"^^<http://www.w3.org/2001/XMLSchema#integer>
дробные числа (decimal) - "1.23"^^<http://www.w3.org/2001/XMLSchema#decimal>
даты (date) - "1982-08-30"^^<http://www.w3.org/2001/XMLSchema#date>
время (time) - "11:24:00"^^<http://www.w3.org/2001/XMLSchema#time>
дата-время (dateTime) - "1982-08-30T11:24:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
```

Предположим, что мы хотим добавить к нашему RDF-графу утверждение о том, что квартира БаронВэй имеет три спальни. С помощью синтаксиса Turtle это утверждение записывается следующим образом:

```

<http://www.semanticwebprimer.org/ontology/apartments.ttl#КвартираБаронВэй>
<http://www.semanticwebprimer.org/ontology/apartments.
ttl#имеетКоличествоСпален>
"3"^^<http://www.w3.org/2001/XMLSchema#integer>.

<http://www.semanticwebprimer.org/ontology/apartments.ttl#КвартираБаронВэй>
<http://www.semanticwebprimer.org/ontology/apartments.ttl#являетсяЧастью>
<http://www.semanticwebprimer.org/ontology/apartments.ttl#ЗданиеБаронВэй>.

<http://www.semanticwebprimer.org/ontology/apartments.ttl#ЗданиеБаронВэй>
<http://dbpedia.org/ontology/location>
<http://dbpedia.org/resource/Amsterdam>.
```

Приведенные выше примеры являются довольно громоздкими. Для упрощения записи в синтаксисе Turtle используется целый ряд конструкций, рассмотренных в следующем разделе.

### **2.3.1.2. Сокращения**

Часто для определения словарей используются одинаковые URI. В нашем примере ресурсы Квартира Барон Вэй и Здание Барон Вэй описаны с использованием одного и того же URL – <http://www.semanticwebprimer.org/ontology/apartments.ttl>. Этот URL определяет так называемое *пространство имен* для данных ресурсов. Синтаксис Turtle позволяет вводить аббревиатуры (сокращения) для таких URL. Для этого используется ключевое слово `@prefix`, определяющее короткое имя для конкретного пространства имен. Например, для пространства имен <http://www.semanticwebprimer.org/ontology/apartments.ttl> можно ввести сокращенное имя `swp`. Такие сокращенные имена называются также *уточненными*. Перепишем приведенный выше пример с использованием префиксов.

```

@prefix swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
@prefix dbpedia: <http://dbpedia.org/resource/>.
@prefix dbpedia-owl: <http://dbpedia.org/ontology/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
```

```

swp:КвартираБаронВэй swp:имеетКоличествоСпален "3"^^<xsd:integer>.
swp:КвартираБаронВэй swp:являетсяЧастью swp:ЗданиеБаронВэй.
```

```
swp:ЗданиеБаронВэй dbpedia-owl:location dbpedia:Amsterdam.
```

Обратите внимание, в данном примере ресурсы описываются с помощью уточненных имен, при этом угловые скобки не указываются. Заметим, что при описании ресурсов можно сочетать обычные URL и учтенные имена.

Кроме того, синтаксис Turtle позволяет не повторять имя субъекта, если о нем имеется несколько утверждений.

В приведенном выше примере ресурс `swp:КвартираБаронВэй` используется в качестве субъекта в двух триплетах. Данные утверждения можно записать более компактно, применяя точку с запятой после первого утверждения. Например:

```
@prefix swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.  
@prefix dbpedia: <http://dbpedia.org/resource/>.  
@prefix dbpedia-owl: <http://dbpedia.org/ontology/>.  
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
```

```
swp:КвартираБаронВэй swp:имеетКоличествоСпален "3"^^xsd:integer>;
```

```
    swp:являетсяЧастью swp:ЗданиеБаронВэй.
```

```
swp:ЗданиеБаронВэй dbpedia-owl:location dbpedia:Amsterdam.
```

Если повторяются и субъект, и предикат утверждения, то в конце утверждения можно использовать запятую. Например, расширим наш пример, указав, что здание Барон Вэй находится не только в Амстердаме, но и в Нидерландах. С помощью синтаксиса Turtle это можно записать следующим образом:

```
@prefix swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.  
@prefix dbpedia: <http://dbpedia.org/resource/>.  
@prefix dbpedia-owl: <http://dbpedia.org/ontology/>.  
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
```

```
swp:КвартираБаронВэй swp:имеетКоличествоСпален "3"^^xsd:integer>;
```

```
    swp:являетсяЧастью swp:ЗданиеБаронВэй.
```

```
swp:ЗданиеБаронВэй dbpedia-owl:location dbpedia:Amsterdam,
```

```
    dbpedia:Netherlands.
```

Кроме того, синтаксис Turtle позволяет сокращать общие типы данных. Например, числа могут записываться без кавычек. Если число содержит дробную часть (например, 14.3), то оно интерпретируется как десятичная дробь (типа `decimal`). Если число не содержит дробной части (например, 1), оно интерпретируется как целое (типа `integer`). Это еще немного сокращает наш пример:

```

@prefix swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
@prefix dbpedia: <http://dbpedia.org/resource/>.
@prefix dbpedia-owl: <http://dbpedia.org/ontology/>.

swp:КвартираБаронВэй swp:имеетКоличествоСпален 3;
      swp:являетсяЧастью swp:ЗданиеБаронВэй.
swp:ЗданиеБаронВэй dbpedia-owl:location dbpedia:Amsterdam,
      dbpedia:Netherlands.

```

### **2.3.1.3. Именованные графы**

Выше была рассмотрена возможность выделения конкретных утверждений. Она может быть реализована с помощью языка TriG<sup>1</sup>, который является расширением синтаксиса Turtle. Например, укажем, что утверждения о квартире Барон Вэй созданы человеком по имени Фрэнк, который идентифицируется следующим URL: <http://www.cs.vu.nl/~frankh>. Для этого нужно набор утверждений, которому присваивается URL, заключить в фигурные скобки. Данный пример будет записан следующим образом:

```

@prefix swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
@prefix dbpedia: <http://dbpedia.org/resource/>.
@prefix dbpedia-owl: <http://dbpedia.org/ontology/>.
@prefix dc: <http://purl.org/dc/terms/>2.

```

```

{
  <http://www.semanticwebprimer.org/ontology/apartments.ttl#>
    dc:creator <http://www.cs.vu.nl/~frankh>
}

<http://www.semanticwebprimer.org/ontology/apartments.ttl#>
{
  swp:КвартираБаронВэй swp:имеетКоличествоСпален 3;
      swp:являетсяЧастью swp:ЗданиеБаронВэй.
  swp:ЗданиеБаронВэй dbpedia-owl:location dbpedia:Amsterdam,
      dbpedia:Netherlands.
}

```

При таком подходе утверждения, которые не являются частью определенного графа, также заключаются в фигурные скобки, но пе-

<sup>1</sup> См. <http://www.w3.org/TR/trig/>. – Прим. перев.

<sup>2</sup> Словарь «Дублинское ядро», содержащий термины для описания метаданных любых ресурсов в соответствии с инициативой Dublin Core Metadata Initiative (DCMI). – Прим. перев.

ред ними не указывается URL. Данный граф называется *графом по умолчанию*.

### 2.3.2. Другие синтаксисы RDF

Для записи RDF, кроме синтаксиса Turtle, можно использовать и ряд других синтаксисов, среди которых два являются стандартизованными: RDF/XML и RDFa<sup>1</sup>.

### 2.3.2.1. RDF/XML

Синтаксис RDF/XML представляет собой кодирование модели данных RDF на языке XML, что позволяет использовать существующие инструменты языка XML. Первоначально RDF/XML был единственным стандартизованным синтаксисом для RDF. Однако синтаксис Turtle был принят в качестве дополнительного стандарта, так как является более читабельным. Ниже приведен синтаксис RDF/XML. Субъекты утверждений в данном синтаксисе обозначаются с помощью атрибута `rdf:about` тега `rdf:Descriptrion`, заключенного в угловые скобки. Предикаты и объекты, связанные с субъектом, указываются внутри элемента `rdf:Descriptrion`. Используемые пространства имен указываются с помощью XML-конструкции `xmlns:`. Все содержимое документа в формате RDF/XML должно быть указано внутри элемента `rdf:RDF`.

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:dbpedia-owl="http://dbpedia.org/ontology/"
    xmlns:dbpedia="http://dbpedia.org/resource/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:swp="http://www.semanticwebprimer.org/ontology/apartments.ttl#">
    <rdf:Description rdf:about="http://www.semanticwebprimer.org/ontology/
        apartments.ttl#КвартираБаронВэй">
        <swp:имеетКоличествоСпален
            rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
            3
        </swp:имеетКоличествоСпален>
    </rdf:Description>
    <rdf:Description
        rdf:about="http://www.semanticwebprimer.org/ontology/apartments.ttl#
            КвартираБаронВэй ">
```

<sup>1</sup> На момент перевода книги стандартизирован уже целый ряд синтаксисов, помимо двух указанных, а именно: Turtle, N-Quads, N-Triples, TriG, JSON-LD 1.0. См. [http://www.w3.org/standards/techs/rdf#w3c\\_all](http://www.w3.org/standards/techs/rdf#w3c_all). – Прим. перев.

```

<swp: являетсяЧастью
  rdf:resource="http://www.semanticwebprimer.org/ontology/apartments.ttl#
    КвартираБаронВэй"/>
</rdf:Description>
<rdf:Description
  rdf:about="http://www.semanticwebprimer.org/ontology/apartments.
  ttl#ЗданиеБаронВэй">
  <dbpedia-owl:location
    rdf:resource="http://dbpedia.org/resource/Amsterdam"/>
</rdf:Description>
<rdf:Description
  rdf:about="http://www.semanticwebprimer.org/ontology/apartments.ttl#
  ЗданиеБаронВэй">
  <dbpedia-owl:location
    rdf:resource="http://dbpedia.org/resource/Netherlands"/>
</rdf:Description>
</rdf:RDF>

```

### **2.3.2.2. RDFA**

Один из вариантов использования языка RDF заключается в описании и разметке содержимого веб-страниц в формате HTML. Для этого и введен синтаксис RDFA. Синтаксис RDFA предполагает встраивание RDF-элементов в атрибуты HTML-тегов. Приведем пример веб-страницы, содержащей рекламу квартиры Барон Вэй.

```

<html>
<body>
<H1> Продается квартира Барон Вэй</H1>
Квартира Барон Вэй имеет три спальни. Рассчитана на семью. Расположена в здании Барон Вэй, на севере Амстердама.
</body>
</html>

```

Эта страница не содержит какого-либо машиночитаемого описания ресурсов. Можно разметить данную страницу с использованием синтаксиса RDFA следующим образом:

```

<html
  xmlns:dbpedia="http://dbpedia.org/resource/"
  xmlns:dbpediaowl="http://dbpedia.org/ontology/"
  xmlns:swp="http://www.semanticwebprimer.org/ontology/apartments.ttl#"
  xmlns:geo="http://www.geonames.org/ontology#">
<body>

```

```
<H1> Продается квартира Барон Вэй </H1>

<div about="[swp:КвартираБаронВэй]">
Квартира Барон Вэй имеет <span property="swp:имеетКоличествоСпален">3</span>
спальни. Рассчитана на семью. Расположена в <span rel="swp:являетсяЧастью"
resource="[swp:ЗданиеБаронВэй]">здании Барон Вэй</span>

<div about="[swp:ЗданиеБаронВэй]">
Здание расположено на севере Амстердама.
<span rel="dbpediaowl:location" resource="[dbpedia:Amsterdam]"></span>
<span rel="dbpediaowl:location" resource="[dbpedia:Netherlands]"></span>
</div>

</div>
</body>
</html>
```

Эта разметка содержит такое же RDF-описание ресурсов, как и описание в синтаксисе Turtle, приведенное выше. RDF-утверждения кодируются в тегах, таких как `<span>`, `<p>` и `<link>`. RDF-элементы не воспроизводятся браузерами при отображении HTML-страницы. Пространства имен так же, как и в синтаксисе RDF/XML, кодируются с использованием объявления `xmlns`. В некоторых случаях нужно использовать скобки для информирования парсера о том, что используются префиксы. Субъекты определяются с помощью атрибута `about`. Свойства идентифицируются с помощью либо атрибута `rel`, либо атрибута `property`. Атрибут `rel` используется в тех случаях, когда объект утверждения является ресурсом, а атрибут `property` – когда объект утверждения является литералом. Свойства связываются с субъектами утверждений с использованием иерархической структуры HTML-документа.

Каждый из синтаксисов языка RDF, представленных выше, полезен для решения различных задач. Однако важно понимать, что все эти синтаксисы, несмотря на различия, реализуют одну и ту же модель данных и семантику. Таким образом, были рассмотрены способы записи утверждений о сущностях, определяемых с помощью URL. Но что означают эти утверждения? Как компьютер должен интерпретировать эти утверждения? Эти вопросы обсуждаются в следующем разделе, где будет введен язык схем для модели данных RDF.

## 2.4. RDFS: добавление семантики

RDF является универсальным языком, который позволяет пользователям описывать ресурсы, используя свои собственные словари. RDF не рассчитан на использование в какой-то конкретной предметной области и не определяет семантику какой-либо предметной области. Для того чтобы задать семантику предметной области, разработчик или пользователь RDF должен определить, что именно означают используемые им термины. Это осуществляется посредством основных предметно-независимых структур, определённых в языке RDF Schema.

### 2.4.1. Классы и свойства

Каким образом можно описать конкретную предметную область? В качестве предметной области рассмотрим предметную область – аренда квартир. Прежде всего необходимо определить сущности, о которых мы хотим делать высказывания. И здесь нужно сделать первое фундаментальное различие таких сущностей. С одной стороны, необходимо описывать конкретные квартиры, такие как квартира Барон Вэй, и их конкретные места расположения, такие как город Амстердам. И это можно сделать с помощью языка RDF.

Но также необходимо описать квартиры, здания, страны, города и т. д. В чем же заключается разница? В первом случае мы говорим об *отдельных объектах* (ресурсах), а во втором мы говорим о *классах*, определяющих типы этих объектов.

Класс можно рассматривать как множество элементов. Отдельные объекты, которые принадлежат к классу, называются *экземплярами* этого класса. Язык RDF позволяет определить отношение между экземплярами классов и классами с помощью специального свойства `rdf:type`.

Использование классов имеет очень важное значение с точки зрения создания ограничений на утверждения в RDF-документе за счет использования схем. Здесь можно провести аналогию с языками программирования, в которых *типизация* используется как механизм предотвращения бессмысленных операторов (например, таких как  $A + 1$ , где  $A$  – массив; данное выражение некорректно, поскольку предполагается, что аргументы операции  $+$  должны быть числами). Подобный механизм нужен и для языка RDF. Необходимо запретить такие утверждения, как:

Квартира Барон Вэй арендует Джейфа Мейера,  
Амстердам имеет количество спален 3.

Первое утверждение бессмысленно, так как квартира не может арендовать человека. Это накладывает ограничение на значения свойства «арендует». Говоря математическим языком, мы ограничиваем область значения – *диапазон* свойства (*range*).

Второе утверждение бессмысленно, потому что города не имеют спален. Это накладывает ограничение на объекты, к которым может быть применено свойство. Говоря математическим языком, мы ограничиваем область значений – *домен* свойства (*domain*).

## 2.4.2. Иерархии классов и наследование

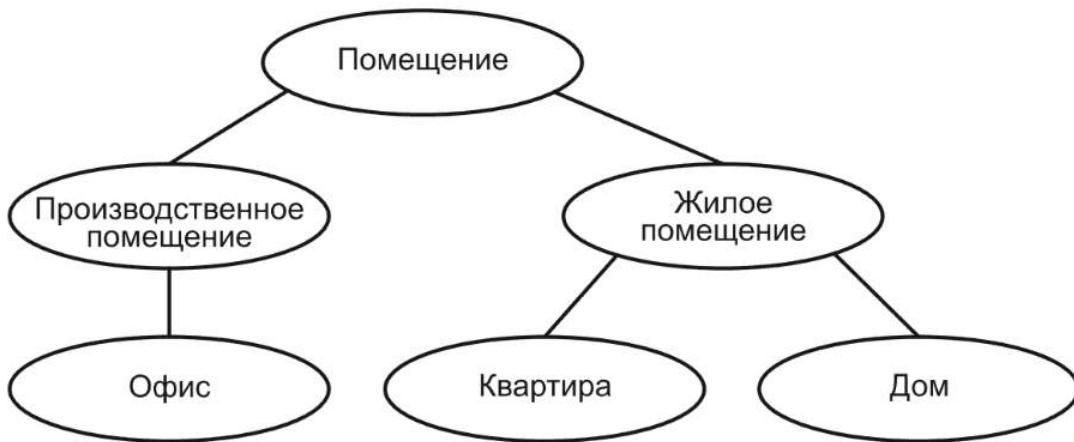
После определения классов следует установить отношения между ними. Например, предположим, что у нас имеются классы для обозначения следующих типов сущностей:

- помещение;
- жилое помещение;
- производственное помещение;
- дом и квартира;
- офис.

Эти классы связаны друг с другом. Например, каждое жилое помещение является помещением. В этом случае можно утверждать, что класс «жилое помещение» является *подклассом* (subclass of) класса «помещение», или, что эквивалентно, класс «помещение» является *надклассом* (superclass of) класса «жилое помещение». Отношения типа «класс–подкласс» определяют иерархию классов, представленную на рис. 2.5. В общем случае класс *A* является подклассом класса *B*, если каждый экземпляр класса *A* является также и экземпляром класса *B*. Язык RDF Schema не требует, чтобы классы образовывали строгую иерархию. Другими словами, граф, отображающий отношения «класс–подкласс», например граф, представленный на рис. 2.5, не должен быть деревом. Класс может иметь несколько надклассов. Если класс *A* является подклассом как класса *B<sub>1</sub>*, так и класса *B<sub>2</sub>*, это просто означает, что каждый экземпляр класса *A* является одновременно экземпляром и класса *B<sub>1</sub>*, и класса *B<sub>2</sub>*.

Иерархическая организация классов имеет очень важное практическое значение. Рассмотрим следующее ограничение на диапазон свойства:

Люди могут арендовать только жилые помещения.



**Рис. 2.5 ♦ Иерархия классов**

Пусть квартира Барон Вэй определена как экземпляр класса «квартира». Она не квалифицируется как жилое помещение, так как отсутствует явное утверждение о том, что квартира Барон Вэй является жилым помещением, и, следовательно, не подпадает под указанное ограничение. Для решения данной проблемы было бы нелогично добавлять утверждение о том, что квартира Барон Вэй является экземпляром класса «жилое помещение». Вместо этого следует указать, что квартира Барон Вэй *наследует* способность быть арендованной от класса «жилое помещение». Именно это и происходит при использовании языка RDF Schema.

Таким образом, язык RDF Schema фиксирует семантику отношения «класс–подкласс», и именно эта семантика используется всеми приложениями, работающими с RDF-документами. Так как язык RDFS позволяет записывать подобные семантические определения, то можно говорить о том, что он является языком для определения семантики конкретных предметных областей (хотя и достаточно ограниченным). Другими словами, RDF Schema – это примитивный язык онтологий.

Термины «класс», «наследование», «свойство» известны и в других областях компьютерных наук, например в объектно-ориентированном программировании. Но, несмотря на сходство значений данных терминов, есть и отличия. В объектно-ориентированном программировании объект класса характеризуется набором свойств, которые определены в этом классе. Добавление новых свойств к классу означает его модификацию.

А в языке RDFS свойства определяются в глобальном масштабе. То есть они не инкапсулируются в качестве атрибутов в определениях классов. Допускается определение новых свойств, которые

могут применяться к существующему классу без изменения самого класса.

С одной стороны, это мощный механизм, применение которого имеет важные следствия: мы можем использовать классы, определенные другими пользователями, и адаптировать их к нашим требованиям с помощью создания новых свойств. С другой стороны, такая трактовка свойств отличается от стандартного подхода, который сложился в области моделирования и объектно-ориентированного программирования. Это еще одна отличительная особенность языков RDF/RDFS.

### 2.4.3. Иерархии свойств

В предыдущем разделе было показано, что между классами могут быть определены иерархические отношения. То же самое можно сделать и для свойств. Например, свойство «арендует» является *подсвойством* (*supproperty*) свойства «проживает в». Если человек  $p$  арендует жилое помещение  $r$ , то человек  $p$  также и проживает в жилом помещении  $r$ . Обратное не всегда верно. Например, человек  $p$  может быть ребенком, проживающим в квартире с семьей, или просто гостем, который не платит арендную плату.

В общем случае если некоторое свойство  $P$  является подсвойством  $Q$ , то если выполняется  $P(x, y)$ , тогда выполняется и  $Q(x, y)$ .

### 2.4.4. Различие слоев RDF и RDFS

В заключение проиллюстрируем разницу слоев RDF и RDF Schema на простом примере. Рассмотрим следующее RDF-утверждение:

Джефф Майер арендует Квартиру Барон Вэй.

Схема для этого утверждения может содержать такие классы, как «человек», «квартира», «дом», «помещение», и такие свойства, как «арендует», «проживает в», «адрес». Рисунок 2.6 иллюстрирует слои языков RDF и RDF Schema для данного примера. На этом рисунке в прямоугольных блоках указаны свойства, в овалах над пунктирной линией – классы, а в овалах под пунктирной линией – экземпляры классов.

На рис. 2.6 схема представлена с использованием элементов формального языка RDF Schema, таких как `subClassOf`, `Class`, `Property`, `subPropertyOf`, `Resource` и т. д. Ниже язык RDF Schema будет описан более подробно.

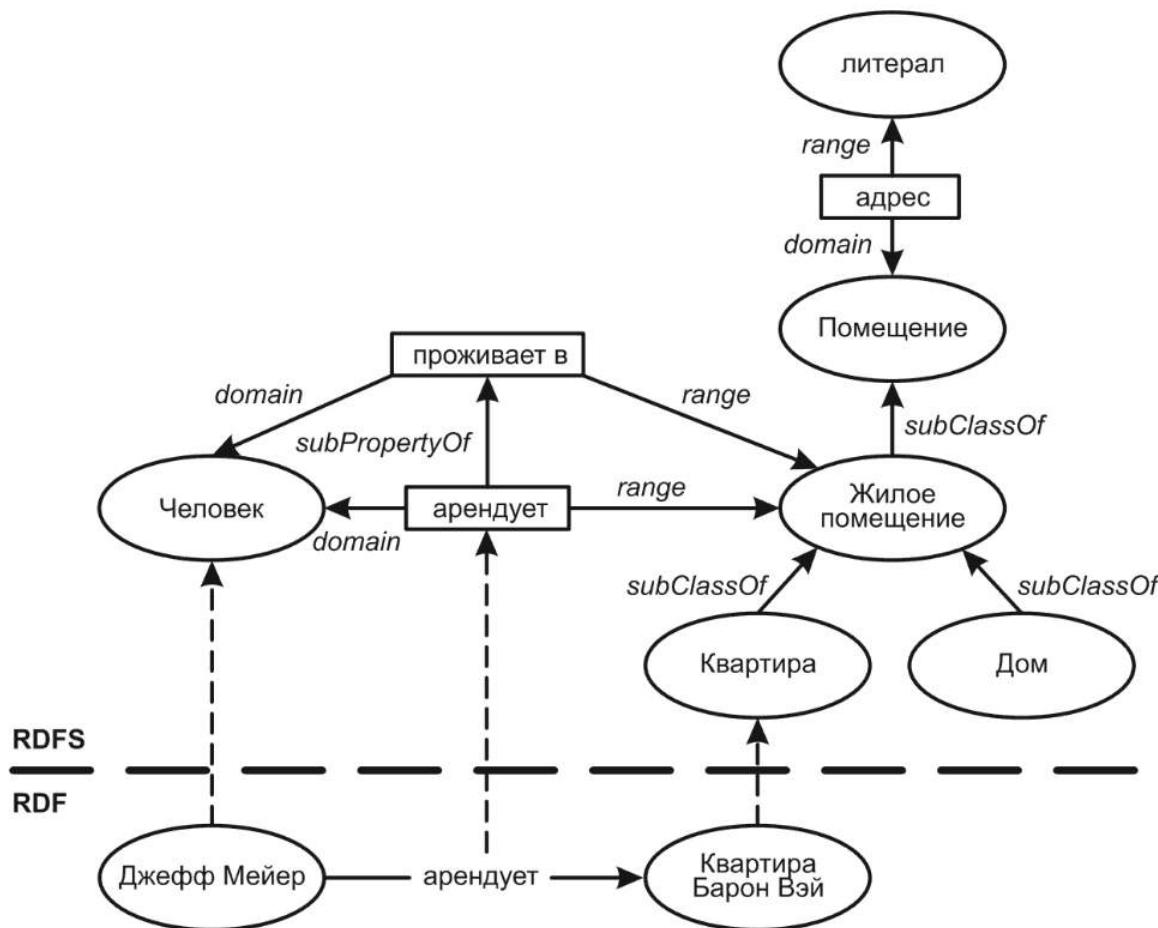


Рис. 2.6 ♦ Слои RDF и RDFS

## 2.5. RDF Schema: язык

Язык RDF Schema предоставляет различные примитивы моделирования для выражения информации, описанной в разделе 2.4. Однако важно понимать, как именно используются эти примитивы. Вас не должно удивлять, что для записи выражений на языке RDF Schema будет использоваться модель данных RDF: примитивы моделирования языка определяются с помощью ресурсов и свойств. Такой подход становится понятен, если еще раз посмотреть на рис. 2.6.

Мы описали этот рисунок как изображение иерархии классов и свойств, а также экземпляров классов, но фактически он представляет собой помеченный граф, который может быть закодирован на языке RDF. Напомним, что язык RDF позволяет выразить любое утверждение о любом ресурсе и что любая сущность, идентифицируемая URI, может рассматриваться как ресурс. Таким образом, если требуется выразить тот факт, что класс «квартира» является подклассом класса «жилое помещение», то это можно сделать следующим образом:

- 1) определить необходимые ресурсы: квартира, жилое помещение и subClassOf;
- 2) определить subClassOf как свойство;
- 3) записать триплет: квартира subClassOf жилое помещение.

Все эти шаги можно реализовать средствами языка RDF. Таким образом, RDFS-документ представляет собой RDF-документ, который может быть записан с помощью одного из известных синтаксисов RDF.

Ниже определяются примитивы моделирования языка RDF Schema.

### **2.5.1. Основные классы**

К основным классам языка RDFS относятся следующие классы:

- rdfs:Class – класс всех классов;
- rdfs:Resource – класс всех ресурсов;
- rdfs:Literal – класс всех литералов;
- rdfs:Datatype – класс типов данных;
- rdf:Property – класс всех свойств;
- rdf:Statement – класс всех реифицированных утверждений.

### **2.5.2. Основные свойства для определения отношений**

К основным свойствам языка RDFS, которые предназначены для определения отношений между ресурсами, относятся следующие свойства.

Свойство rdf:type связывает ресурс и класс, к которому он относится (см. раздел 2.4.1), то есть с помощью этого свойства ресурс объявляется как экземпляр конкретного класса.

Свойство rdfs:subClassOf связывает класс с одним из его надклассов. Все экземпляры класса являются также и экземплярами его надкласса. Подчеркнем, что класс может быть подклассом более чем одного класса. Например, класс ПрофессорЖенщина может быть подклассом и класса Женщина, и класса Профессор.

Свойство rdfs:subPropertyOf связывает свойство с одним из своих надсвойств.

В качестве примера приведем утверждение о том, что любая квартира является жилым помещением:

swp:Квартира rdfs:subClassOf swp:ЖилоеПомещение.

Заметим, что свойства `rdfs:subClassOf` и `rdfs:subPropertyOf` являются по определению транзитивными. Кроме того, подчеркнем интересные факты: класс `rdfs:Class` является подклассом класса `rdfs:Resource` (то есть каждый класс представляет собой ресурс), а класс `rdfs:Resource` является экземпляром класса `rdfs:Class` (`rdfs:Resource` – это класс всех ресурсов, то есть это класс!). По той же причине каждый класс является экземпляром класса `rdfs:Class`.

### **2.5.3. Основные свойства для определения ограничений**

К основным свойствам языка RDFS, которые используются для определения ограничений на другие свойства, относятся следующие.

Свойство `rdfs:domain` определяет домен свойства  $P$  и означает, что любой ресурс, имеющий свойство  $P$ , является экземпляром класса, указанного в качестве домена.

Свойство `rdfs:range` определяет диапазон свойства  $P$  и означает, что значения свойства  $P$  являются экземплярами класса, указанного в качестве диапазона.

В качестве примера приведем утверждения о том, что если некоторый ресурс имеет адрес, то этот ресурс является помещением (что может быть заключено путем логического вывода), и что значением адреса является литерал:

```
swp:адрес rdfs:domain swp:Помещение.  
swp:адрес rdfs:range rdf:Literal.
```

### **2.5.4. Свойства, используемые для реификации**

Ниже приведены некоторые полезные свойства для реификации RDF-утверждений:

- Свойство `rdf:subject` связывает реифицированное утверждение с его субъектом.
- Свойство `rdf:predicate` связывает реифицированное утверждение с его предикатом.
- Свойство `rdf:object` связывает реифицированное утверждение с его объектом.

### **2.5.5. Контейнерные классы**

Кроме того, язык RDF позволяет представить контейнеры стандартным образом. К контейнерам в языке относят пакеты, последовательности и альтернативы (то есть выбор):

- rdf:Bag – класс пакетов;
- rdf:Seq – класс последовательностей;
- rdf:Alt – класс альтернатив;
- rdfs:Container – суперкласс всех контейнерных классов, в том числе трех предыдущих.

### 2.5.6. Вспомогательные свойства

Ресурс может быть определен и описан в вебе многократно. Следующие свойства позволяют определить ссылки на адреса ресурса:

- Свойство rdfs:seeAlso связывает ресурс с другим ресурсом, содержащим любую информацию о нем.
- Свойство rdfs:isDefinedBy является подсвойством свойства rdfs:seeAlso и связывает ресурс с другим ресурсом, который определяет его, как правило, с помощью RDF-схемы.

Часто бывает полезно представить более подробную информацию о ресурсе, предназначенную для чтения людьми. Это может быть сделано с помощью следующих свойств.

- Свойство rdfs:comment задает для ресурса комментарий, то есть относительно большой фрагмент текста.
- Свойство rdfs:label задает для ресурса человекочитаемую метку (имя). Кроме прочего, данная метка может использоваться в качестве имени узла в графовом представлении RDF-документа<sup>1</sup>.

### 2.5.7. Пример: жилье

В качестве примера рассмотрим предметную область – жилье – и представим концептуальную модель данной предметной области, то есть онтологию<sup>2</sup>.

---

<sup>1</sup> Для задания многоязычных имен в свойствах rdfs:label и rdfs:comment можно использовать языковые теги. Например, rdfs:label "Apartment"@en, "Appartement"@nl, "Квартира"@ru. – *Прим. перев.*

<sup>2</sup> Хотя не существует формальных правил именования классов и свойств, рекомендуется использовать следующие походы. Называть класс следует именем существительным в единственном числе. При этом желательно, чтобы имя класса начиналось с прописной буквы и не содержало пробелов, например ЖилоеПомещение. Альтернативным способом наименования классов может быть разделение слов в имени класса знаком подчеркивания, например Жилое\_Помещение. Имя свойства рекомендуется начинать с глагола со строчной буквы, а остальные слова в имени свойства начинать с большой буквы и указывать без пробелов, например проживаетВ. Не важно, какую из предложенных нотаций для именования классов и свойств вы будете использовать, важно придерживаться в именовании имен одной выбранной нотации. – *Прим. перев.*

@prefix swp: <<http://www.semanticwebprimer.org/ontology/apartments.ttl#>>.  
@prefix rdf: <<http://www.w3.org/1999/02/22-rdf-syntax-ns#>>.  
@prefix rdfs: <<http://www.w3.org/2000/01/rdf-schema#>>.

swp:Человек rdf:type rdfs:Class.  
swp:Человек rdfs:comment "Класс людей"@ru.

swp:Помещение rdf:type rdfs:Class.  
swp:Помещение rdfs:comment "Часть пространства здания или другого объекта недвижимого имущества, выделенная для самостоятельного использования"@ru.

swp:ЖилоеПомещение rdf:type rdfs:Class.  
swp:ЖилоеПомещение rdfs:subClassOf swp:Помещение.  
swp:ЖилоеПомещение  
rdfs:comment "Класс всех помещений, в которых живут люди"@ru.

swp:Квартира rdf:type rdfs:Class.  
swp:Квартира rdfs:subClassOf swp:ЖилоеПомещение.  
swp:Квартира rdfs:comments "Класс квартир"@ru.

swp:Дом rdf:type rdfs:Class.  
swp:Дом rdfs:subClassOf swp:ЖилоеПомещение.  
swp:Дом rdfs:comment "Класс домов"@ru.

swp:проживаетВ rdf:type rdfs:Property.  
swp:проживаетВ rdfs:comment "Связывает человека и место его проживания"@ru.  
swp:проживаетВ rdfs:domain swp:Человек.  
swp:проживаетВ rdfs:range swp:ЖилоеПомещение.

swp:арендует rdf:type rdfs:Property.  
swp:арендует rdfs:comment "Свойство наследует домен (swp:Человек) и диапазон (swp:ЖилоеПомещение) от своего надсвойства (swp:проживаетВ)"@ru.  
swp:арендует rdfs:subPropertyOf swp:проживаетВ.

swp:адрес rdf:type rdfs:Property.  
swp:адрес rdfs:comment "Это свойство помещений, имеющее литеральное значение"@ru.  
swp:адрес rdfs:domain swp:Помещение.  
swp:адрес rdfs:range rdf:Literal.

## 2.5.8. Пример: автотранспорт

В данном разделе представлена простая онтология автотранспорта. Отношения между классами данной онтологии представлены на

рис. 2.7.

```

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<#Минифургон> a rdfs:Class ;
    rdfs:subClassOf <#ПассажирскийТранспорт >, <#Фургон> .

<#Автотранспорт> a rdfs:Class .

<#ПассажирскийТранспорт> a rdfs:Class ;
    rdfs:subClassOf <#Автотранспорт> .

<#ГрузовойАвтомобиль> a rdfs:Class ;
    rdfs:subClassOf <#Автотранспорт> .

<#Фургон> a rdfs:Class ;
    rdfs:subClassOf <#Автотранспорт> .

```

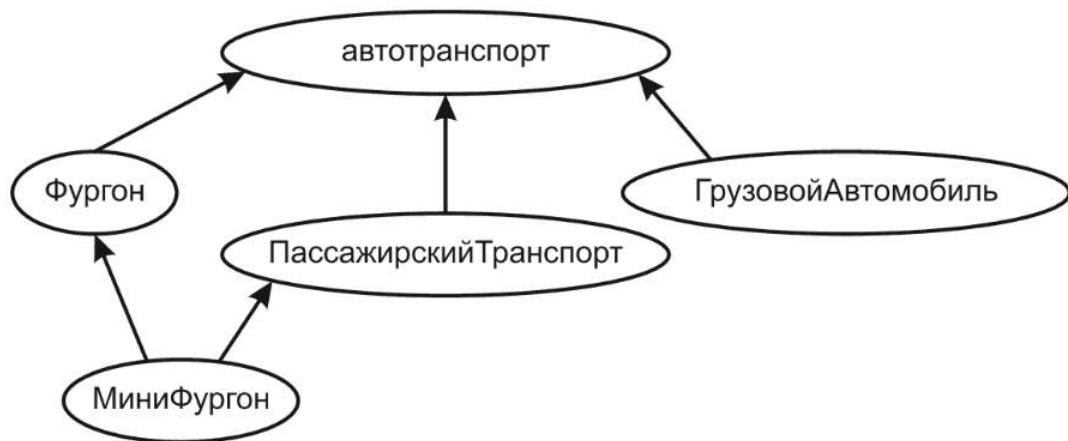


Рис. 2.7 ♦ Иерархия классов для онтологии автотранспорта

## 2.6. Формальные определения языков RDF и RDF Schema

После изучения основных элементов языков RDF и RDF Schema полезно ознакомиться и с их формальными определениями. Эти определения выражены на языке RDF Schema. Проанализируйте, насколько легко они читаются, за счет того, что значение каждого из компонентов было разъяснено в предыдущих разделах.

Следующие определения представляют собой лишь часть полной спецификации языка. Остальные части можно найти в пространствах

имен, указанных в теге rdf:RDF. Мы предоставляем их в оригинальном синтаксисе XML.

## 2.6.1. RDF

```
<?xml version="1.0" encoding="UTF-16"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

  <rdfs:Class rdf:ID="Statement"
    rdfs:comment="The class of triples consisting of a
      predicate, a subject and an object
      (that is, a reified statement)" />

  <rdfs:Class rdf:ID="Property"
    rdfs:comment="The class of properties"/>

  <rdfs:Class rdf:ID="Bag"
    rdfs:comment="The class of unordered collections"/>

  <rdfs:Class rdf:ID="Seq"
    rdfs:comment="The class of ordered collections"/>

  <rdfs:Class rdf:ID="Alt"
    rdfs:comment="The class of collections of alternatives"/>

  <rdf:Property rdf:ID="predicate"
    rdfs:comment="Identifies the property used in a statement
      when representing the statement in reified form">
    <rdfs:domain rdf:resource="#Statement"/>
    <rdfs:range rdf:resource="#Property"/>
  </rdf:Property>

  <rdf:Property rdf:ID="subject"
    rdfs:comment="Identifies the resource that a statement is
      describing when representing the statement in reified form">
    <rdfs:domain rdf:resource="#Statement"/>
  </rdf:Property>

  <rdf:Property rdf:ID="object"
    rdfs:comment="Identifies the object of a statement
      when representing the statement in reified form"/>

  <rdf:Property rdf:ID="type"
```

```
    rdfs:comment="Identifies the class of a resource.  
    The resource is an instance of that class."/>  
  
</rdf:RDF>
```

## 2.6.2. RDF Schema

```
<rdf:RDF  
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">  
  
<rdfs:Class rdf:ID="Resource"  
    rdfs:comment="The most general class"/>  
  
<rdfs:Class rdf:ID="comment"  
    rdfs:comment="Use this for descriptions">  
<rdfs:domain rdf:resource="#Resource"/>  
<rdfs:range rdf:resource="#Literal"/>  
</rdfs:Class>  
  
<rdfs:Class rdf:ID="Class"  
    rdfs:comment="The concept of classes.  
    All classes are resources.">  
<rdfs:subClassOf rdf:resource="#Resource"/>  
</rdfs:Class>  
  
<rdf:Property rdf:ID="subClassOf">  
<rdfs:domain rdf:resource="#Class"/>  
<rdfs:range rdf:resource="#Class"/>  
</rdf:Property>  
  
<rdf:Property rdf:ID="subPropertyOf">  
<rdfs:domain rdf:resource="&rdf;Property"/>  
<rdfs:range rdf:resource="&rdf;Property"/>  
</rdf:Property>  
  
</rdf:RDF>
```

Указанные пространства имен не обеспечивают полного определения языков RDF и RDF Schema. Рассмотрим, например, свойство rdfs:subClassOf. Согласно приведенному выше определению, можно сказать лишь то, что данное свойство применяется к классам и имеет класс в качестве значения. Но смысл языкового примитива «подкласс» (а именно что все экземпляры класса являются также экземплярами его надкласса) никаким образом не зафиксирован. Фактически он

и не может быть выражен в RDF-документе. Если бы это можно было сделать, то отпала бы необходимость в языке RDF Schema.

Формальная семантика языков RDF и RDF Schema будет представлена в следующем разделе. Конечно, RDF-анализаторы и другие программные инструменты для работы с RDF-документами (в том числе процессоры запросов) должны иметь средства работы с этой полной семантикой.

## 2.7. Аксиоматическая семантика языков RDF и RDF Schema

В этом разделе будет формализован смысл примитивов моделирования языков RDF и RDF Schema и, таким образом, будет полностью задана *семантика* этих языков.

В качестве формального языка описания семантики будем использовать *логику предикатов*, общепризнанный формализм для представления всех (символьных) знаний. Формулы, используемые в логике предикатов, называются *аксиомами*.

За счет описания семантики языков RDF и RDFS с помощью формального языка, такого как логика предикатов, она становится однозначной и пригодной для машинной интерпретации. Кроме того, создается основа для поддержки логического вывода с помощью автоматизированных средств (машины вывода), которые манипулируют логическими формулами.

### 2.7.1. Общий подход

Все языковые примитивы в языках RDF и RDF Schema представлены константами: *Resource*, *Class*, *Property*, *subClassOf* и т. д. Для выражения отношений между константами используется несколько предопределенных предикатов.

Также используется вспомогательная теория списков, а именно следующие функциональные символы:

- *nil* (пустой список),
- *cons(x, l)* (добавляет элемент в начало списка),
- *first(l)* (возвращает первый элемент списка),
- *rest(l)* (возвращает остаток списка),

и предикатные символы:

- *item(x, l)* (имеет значение «истина» тогда и только тогда, когда элемент встречается в списке),