CHAPTER 1

Getting Started:
What Do We Need to Say?

1.1 WHAT IS AN ONTOLOGY? WHAT IS OWL?

There are countless definitions of “ontology” that you may wish to explore further, and whole papers were written on the subject in the formative years of the field. For our purposes, just think of an ontology as a model which represents some subject matter. We avoid the common usage of the term “domain” as a synonym for subject matter, because it has a formal meaning in OWL. An ontology communicates what kinds of things there are (for the subject matter of interest) and how they are related to each other. It is built so that automated reasoning software can draw conclusions resulting in new information.

An ontology is different from other models you may have seen in that it represents some (suitably scoped) subject matter as a whole, rather than a model of a particular thing (like an airplane) or a predictive model (of hurricanes, earthquakes, or climate). Also, an ontology can provide a structure for data like a database schema can. However, unlike the latter, an ontology can provide great value even in the absence of data.1

Many notations and languages have been used over the years to represent ontologies, some more rigorous than others. The Web Ontology Language (OWL), developed by the World Wide Web Consortium (W3C), is a language for representing ontologies that is based on formal logic, a discipline that evolved from philosophy and mathematics. It is the only standard for representing ontologies that is widely used both in academia and industry. This book uses examples from a variety of industries based on the commercial ontologies that have been developed for our clients.

In the remainder of this chapter, we explore some of the key concepts in a variety of industries that one might wish to model using OWL. We will discover the kinds of things we need to be able to say about the subject matter to be modeled and show many examples. In doing so, we identify requirements for what OWL must be able to express. In Chapter 2, we explain how to express them in OWL.

---

1 See: Ontologies and Database Schema: What’s the Difference?
1.2 IN THE BEGINNING THERE ARE THINGS

An ontology is a model that represents some subject matter. For any subject, there are things that you care about and want to identify and express in an OWL ontology. What are some specific things in different industries? Say you are a healthcare provider. What are the most important things in healthcare? Stop reading for a moment and brainstorm. Write down a dozen or more of the things that come to mind. For now, focus mostly on things that are written down as nouns (or noun phrases). These nouns will correspond to kinds of things in your subject. Then ask yourself, what are the one, two, or three most fundamental things in healthcare that would concern you as a provider?

Say you are in the finance industry—perhaps a discount broker, or maybe the CIO of a full service investment bank. What are the things that come to mind? What are the most important things in this industry? If it helps, limit the scope to managing assets. Do the same exercise. Brainstorm, write down a dozen or more of the key things, and identify up to three of the most central things in this industry.

Say your job is to manage registration and documentation of ongoing changes for corporations and charities for some particular jurisdiction (e.g., the state of Washington). Go through the same exercise once more. What are the most central things of importance in this subject area?

If none of these examples stimulate you, think of others, ones you know a lot about and have passion for. Pick any subject matter that you would like to have an ontology for. Identify a dozen or so key things, and select the most important ones. In the database field, this activity of identifying what is important in some subject matter is part of data modeling.

<table>
<thead>
<tr>
<th>Table 1.1: Core ideas for different subjects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Healthcare</td>
</tr>
<tr>
<td>Disease</td>
</tr>
<tr>
<td>Diagnosis</td>
</tr>
<tr>
<td>Health insurance policy</td>
</tr>
<tr>
<td>Regulations</td>
</tr>
<tr>
<td>Medicine</td>
</tr>
<tr>
<td>Hospital</td>
</tr>
<tr>
<td>Doctor</td>
</tr>
<tr>
<td>Nurse</td>
</tr>
<tr>
<td>Patient</td>
</tr>
<tr>
<td>Appointment</td>
</tr>
<tr>
<td><strong>Patient visit</strong></td>
</tr>
<tr>
<td>Credentials</td>
</tr>
</tbody>
</table>
Table 1.1 shows some of the things you might have written down. The items in bold are central to that subject.

1.3 KINDS OF THINGS VS. INDIVIDUAL THINGS

Whatever the purpose of your ontology, there is always a dance that goes on between thinking about a particular individual vs. the generic kind of thing that it is. For example, Google Inc. is an individual thing. The kind of thing it is, is corporation. We need to be clear whether we are talking about an individual thing, or the kind of thing that individual is.

If John Doe saw the doctor on January 12, 2014, the kind of thing that “seeing the doctor” is, is called a “patient visit.” The kind of thing that John Doe is, is patient. If John was treated by Jill Smith, then the kind of thing Jill Smith is, would be, person, and more specifically, a doctor or nurse. The patient, John Doe is also a person, and might also be a cancer patient.

Figure 1.1: Individual things and their kinds. Individuals have rounded corners; the rectangles depict kinds.

Figure 1.1 depicts individuals as shapes with rounded corners and their kinds as rectangles. Pick a few of the concepts listed in Table 1.1, especially those in your favorite subject. Then identify the kinds of things and, for each, identify some specific individuals.

1.4 NO THING IS AN ISLAND

By now, I hope you have written down a few dozen things in a few industries. The things that came to mind likely have important roles to play in that subject area, and thus will be related to a number of other things that are also important for that subject. Decide which of the things you wrote down are most important in the sense that they have the richest set of relationships with other things in the same subject area. These are likely to be the most important things you chose in the last exercise.
1. GETTING STARTED: WHAT DO WE NEED TO SAY?

1.4.1 HEALTHCARE

Arguably the most central thing in healthcare is the event of a person receiving healthcare of some sort. After all, the primary goal of healthcare is to keep people healthy. Do you have something in your list that includes this? Perhaps a doctor’s visit or hospital stay. We will use the more general notion of a patient visit which links to both a patient and a healthcare provider (for example, a doctor). There is also the care that was provided, which often includes a diagnosis and treatment. Figure 1.2 illustrates some key things in healthcare and how they are related.

The dotted lines can be concluded from knowing the solid line connections. If you know who the care recipient and care provider are for a given patient visit, you can conclude who received care from who (or, conversely, who gave care to whom).

![Diagram of healthcare relationships](image)

Figure 1.2: Inter-related things in healthcare.

1.4.2 FINANCE

In the asset management side of finance, the goal is to get a return on assets. This is accomplished by a series of financial transactions, most prominently trades. For example, you might purchase 50 shares of Google common stock (GOOG). You might have to sell a bond to free up assets to make that purchase. Each of these transactions is a trade, so the trade is central in asset management.

A trade is not an island. Like a patient visit, it is connected to a variety of other things. What is it related to? There is a buyer, a seller, and possibly a broker. Money changes hands, and ownership is transferred from one legal entity to another along with associated formal documentation. Figure 1.3 illustrates some key things in asset management and how they are related.
Figure 1.3: Inter-related things in finance.

1.4.3 CORPORATE REGISTRATIONS

Exercise 1: Draw a diagram highlighting the key concepts and relationships for the subject of registering corporations just like we did for finance and healthcare. The idea is to scan your list of items or the list in the third column of Table 1.1 and to pick what you think is the central concept. Then list a few of the key relationships that link it to other concepts that are in the table, or that you think are important but are not in the table.

1.5 THINGS CAN HAVE A VARIETY OF ATTRIBUTES

We can say quite a lot about a given thing by specifying relationships connecting it to other things, as in the above examples. But there is more that we want to say that is not so easily handled in that way. For example, many things have associated names and dates. We may wish to state how old someone is. For example, Jill Smith has the first name, “Jill” and her age is 32. Google’s official name is “Google LLC” and it was incorporated on September 4, 1998 (see Figure 1.4).

These kinds of statements are very important, but are different from the other statements we have been making. We have been talking about two individual things being related to one another. But capturing information about names and dates is specifying information about what characteristics or attributes a given individual has. Unlike connecting one individual to another, we are connecting something to a literal value, typically a string, a number or a date.

From another perspective, note that it makes sense to say a trade is related to the broker on that trade or that Jane is related to her patients. But it is quite awkward to say “Google is related to the string ‘Google LLC’”, or that Jane is in a relationship with the string “Smith.” Instead of connecting one individual to another, we are connecting an individual to a literal value. Things like
1. GETTING STARTED: WHAT DO WE NEED TO SAY?

“age,” “date of incorporation,” and “first name” are commonly referred to as attributes. Rather than saying one thing is in a relationship with another thing, we say that something has an attribute whose value is a literal of some kind.

The most common kind of literal is a string, which is used for names, descriptions, and many other things. Other kinds of literals include dates and different kinds of numbers like integer or decimal. Numbers will be used for measuring and counting things like weight and age. A date is a specially formatted item with a very specific meaning. From a modeling perspective, the main thing that characterizes a literal is that we won’t be saying anything more about particular literals such as “John” or the number 32. Literals can be thought of as pure values; they don’t have properties or attributes of their own (see Figure 1.4).

![Figure 1.4: Some common attributes and their literal values.](image)

1.6 MORE GENERAL THINGS AND MORE SPECIFIC THINGS

In our examples so far, we have come across different kinds of things, including patient visit, trade, person, doctor, nurse, patient, legal entity, and corporation. Notice that every corporation is a legal entity. Thus, a corporation is a specific kind of legal entity. What other kinds of legal entity are there? Persons are legal entities, as are many other kinds of organizations, e.g., partnerships, limited liability companies (LLCs), cooperatives, and some charities to name a few. We can think of a legal entity as generalizing these other kinds of things.

Consider the concepts doctor and nurse. There is a more general kind of thing that each of these can be seen as more specific variations of. What might that be? Both are healthcare providers, and both are also persons. What other examples can you find in the things we have seen so far,
where one is more general or more specific than another? Have a look at Figure 1.5 that covers the three subject areas we have considered.

![Figure 1.5: A hierarchy of different kinds of things.](image)

This concludes the discussion about what kinds of things you need to say to build an ontology. Recall that we said an ontology was built in such a way as to support drawing conclusions from existing information. We consider that next.

### 1.7 DRAWING CONCLUSIONS

Despite their remarkable capabilities, computers take things quite literally and at times seem rather dumb. They don’t know the most simple and obvious things. Fortunately, that is changing. These days once a man indicates he is male on an online health form, there is good chance it won’t bother to ask him whether he is pregnant. This is a simple example of the computer doing something a little smarter. It was able to draw the conclusion that the man was not pregnant because it knows that he is male and males cannot be pregnant. Computer programs that are designed to draw conclusions that logically follow from an existing set of data or assertions are called automated reasoners or inference engines.

While this sort of thing can easily be accomplished through hard-coded rules, that approach does not scale. We want a more general way to tell the computer things and have it apply some general principles that allow it to draw a wide variety of interesting and useful conclusions.

Given our examples so far, can you think of some situations where you would want the computer to automatically draw some conclusions for you? Below are a couple examples to get you started. They are depicted in Figure 1.6. Bold lines are for a kind of links, thinner lines are for instance of links. Solid lines are directly asserted, dotted lines indicate the drawing of conclusions.

---

Excerpted from "Demystifying OWL for the Enterprise" - Michael Uschold, June 2018

www.morganclaypoolpublishers.com/owl
1. If a cancer patient is a kind of patient, and a patient is a kind of person, then we want the computer to be able to figure out what common sense tells us, that a cancer patient is also a kind of person.

2. If Google is a corporation, and a corporation is a kind of legal entity, then we want the computer to conclude that Google is in fact a legal entity, just as common sense tells us.

Figure 1.6: Drawing simple conclusions.

When we say the system should be able to “figure out what common sense tells us” we mean it should be able to take the existing information it has and to conclude some new information that follows logically from that information. OWL is based on formal logic, which we will discuss in Chapter 3. It specifies exactly when something follows logically from something else and thus what conclusions should be drawn. It’s one thing to draw a conclusion that is immediately obvious to a human. Automated reasoning with OWL can also draw conclusions that logically follow through a chain of reasoning, even when the conclusions are not obvious.

One kind of conclusion is determining that there is a logical inconsistency. This tells you there is a bug in your ontology or in your data. For example, in Figure 1.7 the red line with an X in it denotes that nothing can be both a person and a corporation. If we already know that Google is a corporation, and someone comes along and mistakenly says that Google is also an instance of person, there is a problem that an automated reasoner can detect. The computer has been explicitly told that:

1. nothing can be both a corporation and a person and

2. Google is both a corporation and a person.
Figure 1.7: Reasoning helps to find an inconsistency.

This is an example of two kinds of things having no overlap. Look again at Table 1.1. See if you can find other examples of two kinds of things that cannot overlap.

**Exercise 2:** How do you resolve the apparent contradiction that the U.S. Supreme Court’s Citizen’s United decision declared that a corporation is legally a person with the above common sense example in the ontology? Can you think about it in such a way that there is in fact no contradiction?

### 1.8 DATA AND METADATA

To create a good model of the subject matter of interest, OWL needs to give modelers a way to say the following.

1. There are individual things.
2. There are kinds of things, some of which do not overlap.
3. An individual is an instance of a certain kind of thing.
4. There are more specific and more general kinds of things.
5. There are relationships between things.
6. Things have attributes with literal values.

These things must be said in a way that supports drawing conclusions both to add new information, and to detect and debug logical inconsistencies. They can be viewed as requirements for OWL, or any ontology modeling language, for that matter.

So far we have been very informal in describing different subjects. The diagrams I have been drawing are meant to be the kinds of diagrams you might draw on a whiteboard in a brainstorming session. Afterward, you would use them as the basis for creating an ontology using the more formal
notation of OWL. That is covered in detail in the next chapter. We will now give a hint about what that will look like.

Once you have an ontology, it can be used to communicate meaning to other humans. While this is important in its own right, we will focus on how humans can get the computer to do useful things with the ontology. One of the main things you will want to do with an ontology is to use it as the basis for creating individuals and relationships between them and storing that as data. This is called populating the ontology.

The ontology provides a vocabulary for creating individuals and making statements about them. For example, consider the following statements in (not-too-stilted) English.

• John Doe is an instance of Person.
• Jill is an instance of Doctor.
• Jane is an instance of Nurse.
• John Doe’s visit to the doctor is an instance of Patient Visit.
• John Doe's visit to the doctor was on date 12Jan2013.
• John Doe’s visit to the doctor has care recipient, John Doe.
• Jill was a care provider on John Doe’s visit to the doctor.
• Jane was a care provider on John Doe’s visit to the doctor.
• John Doe received care from Jill.
• John Doe received care from Jane.

The vocabulary for the subject matter of healthcare is in gold for kinds of things, blue for relationships connecting individuals, and green for attributes with literal values. While not as natural-sounding, it is technically accurate to say that attributes are relationships that connect individuals to literals.

The individuals are in burgundy, and the single literal is in black. Note the generic relationship, is an instance of; it is neither a created individual nor part of the vocabulary of healthcare. Rather, it is part of the vocabulary for modeling. The formal name for this in OWL is rdf:type. We will get into that in the next chapter.

Each of the above sentences in English has a subject, a predicate (i.e., verb), and an object and is asserting something to be true. Count the parts: one, two, three—each sentence is a triple. Some example triples are graphically depicted in Figure 1.8.
So the ontology is the vocabulary for talking about the subject matter of interest. It is used to create and give meaning to data. That vocabulary is also represented as triples. For that reason the ontology is said to play the role of metadata for a database of triples.

In the next chapter, we describe how OWL meets the six requirements (at the beginning of this section) for describing subject matter in general, and how to create data using the subject matter vocabulary.

1.9 SUMMARY LEARNING

In this chapter, we learned what kinds of things we need to say when building an ontology.

What Is an Ontology?

An ontology is a model of some subject matter that you care about and OWL is a formal language for expressing the ontology. It communicates what kinds of things there are and how they are related to each other in a way that supports automated reasoning. An ontology can also be used informally to communicate shared meaning among humans.

What Do You Need to Say When You Build an Ontology?

The kinds of things we need to say to build an ontology are relatively few. They constitute an informal set of requirements for OWL. What you need to say is that there are:

1. individual things;
2. kinds of things (some of which do not overlap);
3. individuals of a certain kind;
4. more specific things and more general kinds of things;
5. relationships that connect things to other things; and
6. relationships that connect things to literals.
1. GETTING STARTED: WHAT DO WE NEED TO SAY?

The things that are said are called assertions, and they are represented as triples. The ontology provides a vocabulary that plays the role of metadata.

**Drawing Conclusions**

If we are careful to say things very precisely, then the computer can draw conclusions for us. The act of drawing conclusions is referred to as performing inference. Special computer programs do this; they are called inference engines or reasoners. Two ways that this helps are (1) it makes the computer seem smarter and (2) the reasoner can detect and explain logical inconsistencies which leads to better ontologies. Stating that two kinds of thing are not overlapping is a big help in spotting inconsistencies.
CHAPTER 2

How Do We Say it in OWL?

2.1 INTRODUCTION

In Chapter 1, we described the key things that need to be said in order to build a model of some subject matter and to create data based on that model. In this chapter, we describe how to say those things in OWL. In doing so, we transition from mostly non-technical language to the more technical language of OWL.

This chapter presents the approximately 30% subset of OWL that you will use 90% of the time when you create ontologies. We mostly illustrate the concepts in the healthcare industry, taking up where we left off in Chapter 1. We also use examples from other industries and subjects.

2.2 SAYING THINGS

At the beginning of Section 1.8 we identified six kinds of assertions that we need to make to create and populate an ontology in OWL. Do you remember what they are? It would be a good idea to memorize them.

You also need to be able to say that a particular literal is of a certain kind (e.g., a string, an integer or a date). Because OWL provides the ability to say this directly in a way that is independent of the subject matter being modeled, it is not one of the six kinds of assertions.

2.2.1 AN ONTOLOGY IS A SET OF TRIPLES

In natural language, many sentences declare that something is so; these are assertions. The basic sentence structure in English has a subject, a predicate, and an object. Table 2.1 gives examples for each of the six kinds of assertions, split up into the three constituent parts.
16  2. HOW DO WE SAY IT IN OWL?

Table 2.1: Example assertions in somewhat natural language

<table>
<thead>
<tr>
<th></th>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Google</td>
<td>is an instance of</td>
<td>An individual</td>
</tr>
<tr>
<td>2</td>
<td>Corporation</td>
<td>is an instance of</td>
<td>A kind of thing</td>
</tr>
<tr>
<td>3</td>
<td>Google</td>
<td>is an instance of</td>
<td>Corporation</td>
</tr>
<tr>
<td>4</td>
<td>Corporation</td>
<td>is a special kind of</td>
<td>Legal Entity</td>
</tr>
<tr>
<td>5</td>
<td>Google</td>
<td>is a subsidiary of</td>
<td>Alphabet</td>
</tr>
<tr>
<td>6</td>
<td>Google</td>
<td>has official name</td>
<td>“Google Inc.”</td>
</tr>
</tbody>
</table>

The legend for fonts is the same as used at the end of Section 1.8.

1. OWL constructs: black italics is for things that are part of the OWL modeling language.

2. Subject matter (i.e., metadata)
   a. gold is for kinds of things
   b. blue is for relationships connecting individuals to individuals
   c. green is for relationships connecting individuals to literals.

3. Data
   a. burgundy is for individuals
   b. black plain font is for literals

This same structure is used by OWL. An ontology is a set of assertions and each assertion is represented as a triple with a subject, a predicate, and an object.

Thus, an ontology is represented as a set of triples.

1. **Subject**: The subject of a triple is the thing about which something is said.

2. **Predicate**: The predicate of a triple indicates the sort of thing that is being said; it corresponds to a way that two things can be related, or a way that something is related to a literal. Echoing English, it often corresponds to a verb.

3. **Object**: The object of the triple is the individual or literal that is linked to the subject via the predicate. When the object is a literal, it is often thought of as a “value.”
Figure 2.1: Assertions represented as triples.

This is extremely important and it bears repeating.

1. An ontology is a set of assertions.

2. Each assertion is represented as a triple with a subject, a predicate, and an object.

3. An ontology is represented as a set of triples.

Of course, OWL uses a much more precise and formal notation than the quasi-English sentences above. Table 2.2 has the formal triples that correspond to the assertions in Table 2.1.

A set of triples forms a directed graph—which is just a set of nodes with arrows connecting one node to another. The subjects and objects are the nodes and the predicates are the labels on the arrows. Each arrow corresponds to a single triple. The arrow goes in the direction from the subject to the object.

Notice that some of the entries in Table 2.2 appear in more than one triple (e.g., doe:_Google & doe:Corporation). The latter appears twice as the subject and once as the object. When represented as a graph, multiple occurrences snap together as one as in Figure 2.2. Graphs representing knowledge and data as depicted in Figure 2.2 are commonly referred to as knowledge graphs.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>doe:_Google</td>
<td>rdf:type</td>
<td>owl:NamedIndividual</td>
</tr>
<tr>
<td>doe:Corporation</td>
<td>rdf:type</td>
<td>owl:Class</td>
</tr>
<tr>
<td>doe:_Google</td>
<td>rdf:type</td>
<td>doe:Corporation</td>
</tr>
<tr>
<td>doe:Corporation</td>
<td>rdfs:subClassOf</td>
<td>doe:LegalEntity</td>
</tr>
<tr>
<td>doe:_Google</td>
<td>doe:isSubsidiaryOf</td>
<td>doe:_Alphabet</td>
</tr>
<tr>
<td>doe:_Google</td>
<td>doe:hasOfficialName</td>
<td>“Google Inc.”^^xsd:string</td>
</tr>
</tbody>
</table>
Each predicate is called a property in OWL. As depicted in Figure 2.1, there are two main kinds of properties:

1. **object property**: relates an individual to another individual and

2. **data property**: relates an individual to a literal

Literals should be thought of as being primitive—you cannot say anything about them. As such, OWL does not permit a literal to be the subject of a triple. This bears repeating: *a literal cannot be the subject of a triple.*

The two triples from Figure 2.1 are depicted as triples numbered 5 and 6 in Figure 2.2. Notice how even though Google is the subject of several different triples there is just one node for Google. That’s because it is just one thing. There is no need to depict it more than once. Collapsing together different occurrences of the same node is sometimes called “node-folding.”

We are using the following naming conventions for individuals, classes, and properties.

1. Classes: upper camel case (e.g., LegalEntity)

2. Properties: lower camel case (e.g., isSubsidiaryOf)

3. Individuals: leading underscore (e.g. _Google)

Note that other than the literal in the extreme lower right, all the entries in Table 2.2 have three parts. There is a short prefix, a colon, and the main term. We look into this further in the next section.

Note also that some of the triples help to define the subject matter (2 and 4) and others represent data relevant to that subject (1, 3, 5, and 6). When building an ontology, the focus is on the subject matter. The main work is to create and specify the meaning of classes and properties. In addition, it’s not unusual for there to be a few special individuals that help to define the subject matter. For example, suppose Google was building an enterprise ontology and wanted to define
what an internal organization is (from its perspective). An internal organization would include Google itself, or any organization within Google. To define this requires referring to an individual that represents Google itself. We give an example of this in Section 5.4.8.

Normally we use the term “ontology” to collectively refer to the triples representing the subject matter. In this sense, we can think of the ontology as being metadata; it serves the role of a data schema and gives meaning to the data. For example, triple 3 says Google is a corporation and triple 4 tells us something about what it means to be a corporation.

The fact that there are triples that represent both the metadata and the data is a very important feature of OWL that is unlike a traditional relational database. Because it’s all triples, the data schema and the data go into the same data store—called a Triple Store.

Finally, notice that the literal for Google’s official name indicates the kind of literal it is, in this case a string. The technical term in OWL for “a kind of literal” is rdfs:Datatype. Next we examine how terms such as doe:_Google and rdfs:Datatype are constructed. Table 2.3 summarizes the transition from informal English to formal OWL.

<table>
<thead>
<tr>
<th>Kind of Thing to Say</th>
<th>Example of Saying It.</th>
<th>OWL Construct Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 There are individual things</td>
<td>Google is an individual</td>
<td>(an) owl:Thing &amp;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(an) owl:NamedIndividual</td>
</tr>
<tr>
<td>2 There are kinds of things</td>
<td>Corporation is a kind of thing</td>
<td>(an) owl:Class</td>
</tr>
<tr>
<td>3 An individual is an instance of a certain kind of thing</td>
<td>Google is an instance of Corporation</td>
<td>rdfs:type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>links an individual to a class</td>
</tr>
<tr>
<td>4 There are more specific and more general kinds of things</td>
<td>A Corporation is a specific kind of Legal Entity.</td>
<td>rdfs:subClassOf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>links two classes</td>
</tr>
<tr>
<td>5 There are relationships between things</td>
<td>Google is a subsidiary of Alphabet</td>
<td>an owl:ObjectProperty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>links two individuals</td>
</tr>
<tr>
<td>6 Things have attributes that relate them to literals</td>
<td>Google’s official name is “Google Inc”</td>
<td>an owl:DatatypeProperty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>links an individual to a literal</td>
</tr>
</tbody>
</table>

2.2.2 NAMESPACES, RESOURCE IDENTIFIERS, AND OWL SYNTAX

There is some new notation in Table 2.2. Except for the one literal, every subject, predicate, and object is of the form xxx:yyy. Each of these is a globally unique identifier specifically designed for the Web: it is called a “uniform resource identifier” (URI). A “resource” is what the URI is identifying. For example: doe:_Google identifies the company, Google; owl:Class and rdf:type
identify constructs in the OWL language. These are actually abbreviations using XML namespaces (see Figure 2.3). For example, “rdf:type” is the abbreviated form for the full URI which is:

http://www.w3.org/1999/02/22-rdf-syntax-ns#type.

Figure 2.3: A namespace prefix is an abbreviation.

The part before the colon is called the namespace prefix and the part after the colon is called the local name or sometimes a fragment identifier. A namespace gives the ability to have independently governed and managed ontologies without terminology clashes. For example, you might have an ontology for finance with the term fin:Bank. An ontology about river ecosystems might have the term riv:Bank. The different namespace prefixes distinguish the vocabulary of terms related to finance from the vocabulary describing river ecosystems. A namespace corresponds to a subset of URIs and can be thought of as vocabulary. Each term in the vocabulary starts with the same namespace prefix.

The OWL language makes use of multiple vocabularies. The main ones that you need to be familiar with are listed in Table 2.4. This is because OWL was not built from scratch, it reused existing vocabularies. OWL has its own namespace, but the OWL language uses constructs from three other namespaces: rdfs, rdf, and xsd. For now, think of RDFS as a subset of OWL and think of RDF as the language that OWL uses for representing triples.

OWL borrows specific datatypes (like string and integer) from XML Shema. The table lists one additional namespace, for SKOS, which stands for Simple Knowledge Organization Systems. It was designed to represent taxonomies and thesauri in OWL syntax. A few terms from this namespace are coming into common use in ontology development. We will say a bit more about this in subsequent chapters.
When you create your own ontologies, you will create and use one or more of your own namespaces (with corresponding prefixes) for the vocabulary of terms that are in your ontology. In the above examples, we use the prefix “doe” which is meant to suggest: Demystifying OWL for the Enterprise. The expression “doe:Corporation” is short for http://ontologies.demystifyingowl.com/examples#Corporation. The namespace itself contains everything up to and including the “#”.

Note that a namespace prefix is local to a particular file and although it is unusual to do so, two different files can specify two different prefixes for the same namespace. For example, there are two commonly used prefixes for the XML Schema namespace. The one listed above is xsd, but xs is also commonly used. The parser processing the file for use in a target system substitutes in the full namespace when constructing the URI, as depicted in Figure 2.3.

Even though some namespace prefixes are pervasive (e.g., rdf, rdfs, and owl) they still have to be present in each file. Typically, tools will add the standard namespaces for you. See the top several lines of the OWL in Figure 2.5. It would be technically possible to use other prefixes for these common namespaces or to use these standard namespace prefixes for other namespaces, but it would be perverse to do so.

**URIs vs. IRIs**

OWL uses IRIs, a more general version of URIs for international use. It supports Unicode characters, which include any special characters or scripts in other languages. In case you ever want to use bowl of spaghetti as a namespace prefix, you can. There is a Unicode character for that. For the remainder of this book, we will mostly use the more general term “IRI” rather than “URI.”

---


2. HOW DO WE SAY IT IN OWL?

OWL Constructs and Expressions

OWL is a language for representing ontologies. A natural language such as English is made up of words and punctuation. Words are combined to make noun phrases and verb phrases, which in the simplest case are individual nouns and verbs. Noun and verb phrases combine with other kinds of words to make sentences.

An OWL construct is analogous to a word. OWL constructs are combined to make OWL expressions. A triple is an assertion, and corresponds to a typical sentence in English.

OWL Syntax

There are a number of OWL syntaxes in common use. The main ones are listed below and will be introduced later in this chapter. Depending on the point we are trying to make we will use a variety of different syntaxes and notations.

- **RDF/XML** is the original OWL syntax, but is not very easy to read. It is still in widespread use, but is growing less and less popular. It is not used in this book.
- **Turtle** makes the triples nature of OWL more obvious. It is emerging as the de facto standard.
- **Manchester syntax** is the most concise and was designed for human readability.
- **e6Tools** is a compact visual syntax.

2.2.3 SUMMARY: INFORMAL TO FORMAL

The following summarizes the shift from informal to formal terminology that takes place in moving from Chapter 1 to Chapter 2.

1. Rather than speaking of “building a model of some subject matter,” we simply say “building an ontology.”

2. Rather than speak of “things,” “kinds of things,” and “generalization/specializations” we speak of “Individuals,” “Classes,” and of “subClassOf” relationships, respectively.

3. Rather than saying that something is “some kind of thing,” we say an Individual is an “instance or member of a Class,” or, equivalently, that the “individual’s type is some Class.”

4. Rather than saying that a literal is of a certain kind, we say that literal “has a datatype.”
5. Rather than speaking of “relationships between things,” we will speak of “object properties.”

6. Rather than speaking of “attributes of things,” we will speak of “data properties.”

7. Rather than speaking of “kinds of literals,” we will speak of “datatypes.”

8. Rather than speaking of “saying something,” we will speak of “asserting a triple.”

9. Instead of speaking of “drawing conclusions” to add new information, we will speak of “inference,” which adds new triples.

2.2.4 NOTATIONAL CONVENTIONS

Throughout this book we will frequently be referring to generic concepts in English, independently of how they will be represented. These will generally appear in an ordinary font, even if they are later represented as classes or properties. OWL syntax will appear using a fixed font. In paragraphs of text when I refer to properties or classes I will generally leave out the namespace and leading colon. For example, I might say that a HealthCareProvider connects to a PatientVisit using the careProvider property.

When OWL syntax is set apart from regular paragraphs, such as in a set of triples, or in figures or tables, I either use a namespace prefix with a colon, or where brevity counts, just a leading colon. That serves as a reminder that the full IRI needs to have more than just the local name, even if it is not being depicted. For example:

\[
\text{:}\_\text{JohnDoeDoctorVisit12Jan :careProvider :}\_\text{DrJillSmith}
\]

2.3 A SIMPLE ONTOLOGY IN HEALTHCARE

2.3.1 HEALTHCARE DATA

Figure 1.2 contains an informal diagram that sketched out a few of the key components of what an ontology in healthcare would require. Figure 1.4 showed some sample data. These are just pictures created in a drawing tool that does not understand OWL. However, these diagrams depict preliminary requirements for a healthcare ontology. To create an ontology, you need to have an OWL authoring tool, often referred to as an ontology editor. In principle, all you need is a text editor and sufficient understanding of OWL syntax. In practice, most people use a tool with a graphical UI which saves out a file in valid OWL syntax.
2. HOW DO WE SAY IT IN OWL?

Throughout this book we will use a variety of notations and syntaxes. Diagrams are easy to look at and understand. For that, we use a very compact and fairly self-explanatory notation from a proprietary Visio plugin called e6Tools designed by Simon Robe. It is used to recast the informal diagrams from Chapter 1 into a formal diagrammatic syntax that is designed to be an alternative OWL syntax for easier viewing. We will also show portions of ontologies in Turtle or Manchester syntax.

Figure 2.4: A patient visit in e6Tools OWL notation.

Figure 2.4 depicts a patient visit where both a doctor and a nurse were involved in providing care. We created it by using material from the diagrams in Figures 1.1, 1.2, and 1.4 in and re-drawing it in Visio e6Tools syntax (henceforth abbreviated as “e6”). Notice the strong similarity between the boxes in Figure 1.2 depicting Jill Smith and John Doe and the e6 diagrams. E6 notation is explained below and summarized in Appendix A.2.

Exercise 1: First, see if you can determine which of the six kinds of assertions listed Table 2.3 are being made in Figure 2.4 and which are not. Pay particular attention to whether, where, and how object properties are used vs. data properties.

Recall that the former relate individuals to other individuals, and the latter relate individuals to literals. Here is a summary:

- Instances of: owl:Thing
  - doe:_JohnDoeDoctorVisit12Jan
  - doe:_DrJillSmith
  - doe:_PatientJohnDoe
  - doe:_NurseJaneWilson
The diagram is very compact, but there is a lot going on. Each of the rounded boxes depicts an OWL Individual. At the top of each is the (globally unique) IRI for that individual. The convention we use for this book is for the IRI of an OWL Individual to have a leading underscore. There is an optional rdfs:label which is a human-readable name for the individual. It is an example of an annotation (described more fully in Section 4.10).

Each arrow corresponds to a triple that connects one individual to another. The label on the arrow is the name of the object property—more specifically, it is the IRI for that property. The subject is at the beginning of the arrow, the object is at the end of the arrow. Note how careProvider is used twice. These triples are called object property assertions.

There are also several data property assertions. To make the diagram more compact, these are not shown using arrows. Instead, the box has the IRI of the data property (in black bold font) and the literal value in a green font. We adopt the widely used convention for naming properties, which is to use lower camelcase. If the datatype is string, it is not depicted in the diagram, otherwise it often will be (e.g., dateTime and integer). The boxes depicting data property assertions are light green.

A representative sample of the key assertions in the above diagram is listed below, first in English then in Turtle syntax. The first four triples relate only to healthcare, the last two are about computer systems using OWL. The label says how to present information to the user. The last triple has information used by OWL that is automatically created behind the scenes. Table 2.5 shows how these assertions are represented in OWL as triples. Figure 2.6 shows a more complete set of triples for the example in Figure 2.4.

1. John Doe’s January 12 doctor’s visit has care recipient: John Doe.

2. Dr. Jill Smith is of age 32.
2. HOW DO WE SAY IT IN OWL?

3. John Doe’s first name is “John.”

4. Nurse Jane Wilson’s last name is “Wilson.”


6. `doe:_NurseJaneWilson` has the type, `owl:NamedIndividual` (which is a subclass of `owl:Thing`)

<table>
<thead>
<tr>
<th>Subject (Individual)</th>
<th>Predicate (Property)</th>
<th>Object (Individual/Literal)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>doe:_JohnDoeDoctorVisit12Jan</code></td>
<td><code>doe:careRecipient</code></td>
<td><code>doe:_PatientJohnDoe</code></td>
</tr>
<tr>
<td><code>doe:_DrJillSmith</code></td>
<td><code>doe:age</code></td>
<td>32</td>
</tr>
<tr>
<td><code>doe:_PatientJohnDoe</code></td>
<td><code>doe:firstName</code></td>
<td>“John”^^xsd:string</td>
</tr>
<tr>
<td><code>doe:_NurseJaneWilson</code></td>
<td><code>doe:lastName</code></td>
<td>“Wilson”^^xsd:string</td>
</tr>
<tr>
<td><code>doe:_NurseJaneWilson</code></td>
<td><code>rdfs:label</code></td>
<td>“Nurse Jane Wilson”^^xsd:string</td>
</tr>
<tr>
<td><code>doe:_NurseJaneWilson</code></td>
<td><code>rdf:type</code></td>
<td><code>owl:NamedIndividual</code></td>
</tr>
</tbody>
</table>

Recall that (except for literals) the subject, predicate, and object must each have a globally unique identifier. OWL uses IRIs. They usually take the form of a uniform resource locator (URL), which ideally points to a location on the web. In the example above, the IRI for nurse Jane Wilson is: `doe:_NurseJaneWilson`. Because this is not very nice looking for humans, it is common to add a label that is used to refer to the individual when displayed by software, e.g., “Nurse Jane Wilson.”
```turtle
@prefix doe: <http://ontologies.demystifyingowl.com/examples#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

doe:healthABox rdf:type owl:Ontology .

# Object Properties
doe:careProvider rdf:type owl:ObjectProperty .
doe:careRecipient rdf:type owl:ObjectProperty .

# Data properties
doe:age rdf:type owl:DatatypeProperty .
doe:date rdf:type owl:DatatypeProperty .
doe:firstName rdf:type owl:DatatypeProperty .
doe:lastName rdf:type owl:DatatypeProperty .

# Individuals
doe:_DrJillSmith rdf:type owl:NamedIndividual ,
    owl:Thing ;
    rdfs:label "Dr. Jill Smith" ;
doe:age 32 ;
doe:firstName "Jill"^^xsd:string ;
doe:lastName "Smith"^^xsd:string .

doe:_JohnDoeDoctorVisit12Jan rdf:type owl:NamedIndividual ,
    owl:Thing ;
    doe:date "12-Jan-2013"^^xsd:dateTime ;
doe:careProvider doe:_DrJillSmith ,
doecareRecipient doe:_PatientJohnDoe .

doe:_NurseJaneWilson rdf:type owl:NamedIndividual ,
    owl:Thing ;
    rdfs:label "Nurse Jane Wilson" ;
doe:firstName "Jane"^^xsd:string ;
doe:lastName "Wilson"^^xsd:string .

doe:_PatientJohnDoe rdf:type owl:NamedIndividual ,
    owl:Thing ;
    rdfs:label "Patient John# Doe" ;
doe:age 57 ;
doe:lastName "Doe"^^xsd:string ;
doe:firstName "John"^^xsd:string .
```

Figure 2.5: Turtle syntax.
2. HOW DO WE SAY IT IN OWL?

2.3.2 HEALTHCARE METADATA

The healthcare data we have in the previous section can be created in OWL and loaded into any OWL software. Although we are thinking about the kinds of things there are, e.g., patient, person, doctor, nurse, etc., they are not represented yet. We have referred to the specific properties (careRecipient, firstName), but we have not defined them in any way. This is an important thing to be aware of. You can create and load data as a set of triples without defining the metadata. This is convenient when you are getting started, but it is bad practice to never have any metadata.

![Diagram of healthcare metadata classes and properties]

The metadata tells you about the data. In OWL, the metadata gives meaning to the data and supports inferences that can help find bugs as well as draw new conclusions. Figure 2.6 includes all the relevant metadata for the instances in Figure 2.4. In OWL, metadata mainly consists of:

1. classes for representing the kind of thing an individual is, e.g.,
   Person, PatientVisit

2. subClassOf relationship to indicate that a one class is a specialization of another class, e.g.,
   Patient is a subclass of Person, a PatientVisit is a subclass of Event

3. properties for characterizing the nature of the relationship between two individuals or between an individual and a literal, e.g.,
   careRecipient, firstName.

The figure shows classes as rectangles and properties with rounded corners. The data properties look pretty similar to the object properties. The former are green and the latter blue. There are three new things we have not yet seen. One is that we have added comments to explain the meaning in an informal way. The comments do not affect inference, but they are very important to help others understand your ontology. When ontologies get large, the comments will help remind you what you were thinking when you added that item to the ontology.
The second new thing is the [F] on the upper right of the careRecipient definition. This is a reference to the term “function” from mathematics. For our purposes it means that patient visit can only have one care recipient. This reflects the real world fact that in this particular healthcare organization, there is never more than one patient cared for during a single patient visit (at least not officially). Being functional is but one of several characteristics that we may specify for properties.

The other new thing is a way to say that properties are only relevant for certain types of subjects and objects. For example, whenever anyone enters an age, we want it to be an integer (ideally a positive one). In the ontology, we see string in the firstName and lastName properties; and we see integer in the age property.

Regarding the careRecipient property, we have in mind that the subject will be a PatientVisit and the object will be a Person. It would make no sense to say a Regulation had a careRecipient, or that the careRecipient was a Disease. So we want to specify two things:

1. What kind of thing must the Subject be, in a triple using a particular property? In OWL, this is called the “domain” of the property

2. What kind of thing must an Object be, in a triple using a particular property? In OWL, this is called the “range” of the property

If you do not find these terms very natural or intuitive, you have plenty of company. The terms come from mathematics and are used to describe functions, among other things. Think of it this way.

1. “domain” translates informally to “only applies to.” If a property has domain C, then that means the property only applies to things of type C, e.g., careRecipient only applies to PatientVisits.

2. “range” translates informally to “range of possible values.” If a property has range C, then the range of possible values that a property can have must all be instances of type C, e.g., the range of possible values for careRecipient is any kind of Person.

Being able to specify domains and ranges for properties is one of many ways to add semantic information to the ontology. The primary benefit is to remove ambiguity and make it easier for both people and machines to use the ontology. Domain and range can also be used help the inference engine to spot errors.

It is important to be aware that OWL domain and range are not constraints. There is no gate keeper checking to make sure every care recipient is in fact a person. If someone accidentally says that the care recipient is an individual known to be an organization, then that organization is inferred to be a Person. Something similar happens for domain. Don't worry if this is a bit confusing. We go into more depth on domain and range in Section 4.4.
2. HOW DO WE SAY IT IN OWL?

### 2.3.3 INDIVIDUALS AND THEIR TYPES

The astute observer will notice that there is one key thing missing. We have data representing several individuals, but did you notice what types they are? Look again at Figure 2.5. What are the types of the individuals? How helpful is that? The only types given are `owl:NamedIndividual` and `owl:Thing`. The ontologist does not specify these types, they are created by the system interpreting the OWL. Every individual is necessarily an instance of `owl:Thing`. Every individual that you give a name to (i.e., a IRI) is an instance of `owl:NamedIndividual`. We purposely we created IRIs that are indicative of their meaning so we can readily create triples that explicitly indicate the types.

- `doe:_JohnDoeDoctorVisit12Jan` `rdf:type` `doe:PatientVisit`
- `doe:_PatientJohnDoe` `rdf:type` `doe:Patient`
- `doe:_NurseJaneWilson` `rdf:type` `doe:Nurse`
- `doe:_DrJillSmith` `rdf:type` `doe:Doctor`

This seems pretty obvious. We definitely want these individuals to be of the types indicated. However, it may not always be best to directly assign these types. Can you think of why that might be? Here is a hint. People can play many roles. A person might be a Doctor and a Patient, at different times. So one individual might belong to more than one class. Suppose a new class is defined in such a way that the person now fits that description too. For example, maybe there is a class: `FormerHospitalAdministrator` that Dr. Jill Smith belongs to because of a job she did in the past. It can be a lot of work keeping track of all the classes an individual belongs to. However, if we define our classes carefully, we can to some extent automatically classify individuals. In the next section, we will look at how we can perform this kind of inference.

Note that there are a number of different and more or less interchangeable ways to say in English what kind of a thing an individual is. The last one is closer to OWL, but less natural-sounding.

1. An individual is a instance of some Class.
2. An individual is a member of some Class.
3. An individual’s type is some Class.

### 2.3.4 RICHER SEMANTICS AND AUTOMATIC CATEGORIZATION

We have said relatively little about the meaning of the classes. We have specified some subclass relationships and indicated how some of the classes are used with properties (via domain and range). But we can do much more.
Using Properties to Add Meaning to Classes

Consider a PatientVisit. We know that it is an Event. What else can we say about a PatientVisit? More specifically, what do we know to be true about every single PatientVisit, by definition? Think in terms of what relationships it must have with other individuals. If there is a PatientVisit, must there not be a Patient? Must there not also be a care provider of some sort?

Although OWL gives us a way to say this, it takes some practice to translate what you are thinking in natural language into OWL. One good way to do this, especially when initially learning OWL, is to first think of what you want to say and write it down in precise and natural English. After that, rephrase it step by step into language that more directly fits the subject-predicate-object pattern using the classes and properties already defined. For example:

1. every PatientVisit has at least one Patient;
2. every PatientVisit is associated with one Person who is receiving care;
3. every PatientVisit has a careRecipient relationship with at least one Person; and
4. every PatientVisit is the subject of at least one triple where the predicate is careRecipient and the object is of type Person.

**Exercise 2:** Go through the same process starting with the statement “Every PatientVisit has at least one healthcare provider.”

You should end up with something very similar.

- Every PatientVisit is the subject of at least one triple where the predicate is careProvider and the object is of type Person.

The last sentences in each case are directly expressible in OWL. Most OWL syntaxes for representing this kind of thing are hard to understand. This is one reason we draw pictures. One exception is Manchester syntax, which is depicted on the right in Figure 2.7. On the left is the e6 visual notation. Some of it is surprisingly close to understandable (albeit stilted) English, the rest needs some explanation. What Figure 2.7 says about PatientVisit is:

1. PatientVisit is a Class
2. Every PatientVisit is an Event
3. Every PatientVisit necessarily has a care provider that is a Person
4. Every PatientVisit necessarily has a care recipient that is a Person

See: https://www.w3.org/TR/owl2-manchester-syntax/.
2. HOW DO WE SAY IT IN OWL?

The last three statements list conditions that must be true for every member of the Person class. We sometimes refer to them as necessary conditions. Note that the graphical notation echoes the Manchester Syntax very closely. We see something new here. An oval that represents a property is inside a rectangle that represents a Class. This reflects the fact that we are using properties to specify the meaning of the class. The (N) that you see is used in e6 to mean necessary.

```
Class: doe:PatientVisit

SubClassOf:
    doe:Event,
    doe:careProvider some doe:Person
    doe:careRecipient some doe:Person
```

Figure 2.7: Adding necessary conditions.

There is one thing here that will not be immediately obvious. Namely, why is the OWL construct, subClassOf is used to connect the class PatientVisit to the expression “doe:careProvider some doe:Person”? Only a class can be a subclass of another class. Therefore, the expression “doe:careProvider some doe:Person” must represent a class. But what class is that? Unfortunately, this one is pretty hard to work out, because it not intuitive. If you are the adventurous type, see what you can come up with before reading on.

The class expression, “doe:careProvider some doe:Person” represents “the set of all things that have at least one care provider that is a person.” To understand why the subclass construct is used we will again develop a series of sentences that are equivalent in meaning, like we did above. We will start with a simple subclass relationship.

1. [The class] PatientVisit is a subclass of Event.
2. Every [instance of] PatientVisit is an Event.
3. The set of all PatientVisit(s) is a subset of the set of all Event(s).
4. IF X is a PatientVisit, THEN X is also an Event.
5. Every [instance of] PatientVisit is necessarily an Event.

This is fine for ordinary named classes. However, “doe:careProvider some doe:Person” is a class without a name; it just is an expression. It can be very handy to not have to give things names, but it is very important to be able to think of a name that reflects the meaning. Such names will often be long and unwieldy, erring on being unambiguous about the meaning.
So, in English, the expression “doe:careProvider some doe:Person” means “The set of all things that have a care provider that is a person.” A long but accurate name for this class might be ThingWithACareProviderThatIsAPerson. We can now do the same thing we did for PatientVisit and Event. To see how this works, we will just replace the class Event with the class ThingWithACareProviderThatIsAPerson in the above lines numbered 1–5. We add one sentence at the end that reads better as English.

1. [The class] PatientVisit is a subclass of ThingWithACareProviderThatIsAPerson.

2. Every [instance of] PatientVisit is a ThingWithACareProviderThatIsAPerson.

3. The set of all PatientVisits is a subset of the set of all Thing(s)WithACareProviderThatIsAPerson.

4. If X is a PatientVisit, then X is also a ThingWithACareProviderThatIsAPerson.

5. Every [instance of] PatientVisit necessarily is a ThingWithACareProviderThatIsAPerson.

6. Every PatientVisit necessarily has a care provider that is a Person.

This illuminates why the subclass relationship is being used to connect the PatientVisit to the class expression “doe:careProvider some doe:Person.” Convince yourself that all of these sentences mean the same thing. The last sentence is what we started with, so we have come full circle.

Let us revisit the original purpose. We were trying to capture the fact that every patient visit necessarily has a care provider that is a person. To do that, we create the class expression: “doe:careProvider some doe:Person” and make PatientVisit a subclass of it. Because of the way subclass works, this does the trick (see Figure 2.8). A good way to visualize this is using Venn diagrams. In the diagram, shapes depict sets and a shape within a shape depicts a subset relationship. Points inside a shape indicate members of the set. I hope this is at least slightly clearer than mud; but, if not, take heart—we will be seeing many more examples.

Perhaps you are wondering whether a class expression like: “doe:careProvider some doe:Person” has a name. Indeed it does, such an expression is called an OWL restriction. Because there are many meanings of the word “restriction,” we will refer to an OWL restriction as a “property restriction.” This makes a certain amount of sense, in that it is mainly used to narrow

---

5 Per common usage, the term “Venn diagram” is being used loosely to include Euler diagrams. If you are curious, see: https://www.cs.kent.ac.uk/events/conf/2004/euler/eulerdiagrams.html.
down (i.e., restrict) the possible meaning of a given class. The use of the property restriction “`doe:careProvider some doe:Person`” on the class `PatientVisit` restricts the set of possible individuals that can be members of the class, `PatientVisit`. For example, if it is known that something does not have a care provider, then it cannot be a `PatientVisit`.

![Figure 2.8: Property restrictions as sets.](image)

In a similar way, we could restrict the meaning of a class `Bicycle`, to include just those things that have exactly two wheels. Making the class `Bicycle` a subclass of such a property restriction would mean that if you know something has more than two wheels or less than two wheels, then we can rule out the possibility of it being a member of the class `Bicycle`. See the Venn diagram in Figure 2.8.

This part is intuitive enough. What is not very intuitive, from a terminology perspective, is that strictly speaking, a property restriction always represents a class. At times I will use the term “property restriction class” to emphasize this fact. There are two main things that make an OWL property restriction class different than other OWL classes we have seen so far. First, it is a class that is defined by what properties its members have. Second, it need not have a name, it can be used purely as an expression.

Whether or not to give a property restriction class a name is a modeling decision. When learning how to use property restrictions, it is a good idea to at least think of a name for the property restriction classes, such as `ThingWithACareProviderThatIsAPerson` or `ThingWithExactlyTwoWheels`. For experienced OWL users, the main reason to give a property restriction class a name is if the class will be used in a variety of situations.
Data Properties

For the most part, data properties can be used in the same ways that object properties are. They can be arranged in hierarchies, they have domains and ranges, and they can be functional. For example, you can ensure that individuals can have at most one last name by making the property `doe:lastName` be functional. A data property can also be used in a property restriction. To ensure that every person has a last name you could create a property restriction class such as “`doe:lastName some xsd:string`” and make `Person` a subclass of that class. The differences all arise from the fact that they connect individuals to literals, not individuals.

**Exercise 3:** If you had to give that class “`doe:lastName some xsd:string`” a name, what might it be? What needs to be true to be a member of this class?

Automatic Categorization

We could go further still in defining `PatientVisit`. We might decide that for our purposes in our healthcare organization, anything at all for which the following is true, always has to be a `PatientVisit`.

1. It is an `Event`.

2. It has a `careProvider` that is a `Person`.

3. It has a `careRecipient` that is a `Person`.

After all, what else could possibly have these three things be true about it and *not* be a `PatientVisit`? Let’s suppose that if there are any counter-examples, that they are obscure edge cases that we can ignore for our business. What is the impact of doing this? If we come across any individual for which the above three statements hold, then we know it must be a `PatientVisit`. Thus, inference has the ability to automatically determine that an individual belongs in certain classes. Two things are going on. First, we still infer that:

**IF:** an individual is known to

1. be a member of the `PatientVisit` class,

**THEN:** we can infer the following new information:

1. It is an `Event`  
2. It has a `careRecipient` that is a `Person`  
3. It has a `careProvider` that is a `Person`

We could do this before. What we have added is the ability to go the other way around.
2. HOW DO WE SAY IT IN OWL?

IF: an individual is known to:
   1. be a member of the class Event AND
   2. have a careRecipient that is a Person AND
   3. have a careProvider that is a Person
THEN: we can conclude that the individual is a member of the PatientVisit class.

See Figure 2.9. It is the same as Figure 2.7 except that class equivalence is used instead of subclass.

As previously noted, people can play various roles. Being a patient is such a role, as is being a healthcare provider. Specifically, any Person that is a careRecipientOn some PatientVisit is a Patient. We just have to define Patient using one of these newfangled OWL property restrictions (see Figure 2.10). Notice that until now we have only been talking about the relationship between a PatientVisit and the Patient from the perspective of the PatientVisit, which points to the careRecipient. However, to define patient in the way we just described, we have to talk about that relationship from the perspective of the Patient.

Let us consider this idea of perspective using a more familiar example of a relationship in everyday life: being a parent. If Peter says he is the “parent of” Patricia, what does Patricia say? Essentially, we are looking at the relationship going the other way, or its inverse. Patricia might say she “has parent,” Peter. So a good name for the “parent of” relationship going the other way is “has parent.” If we think of this as triples, we are just flipping the subject and the object.

We now revisit our PatientVisit example. The name of the careRecipient relationship going the other way is careRecipientOn. These names are chosen so that when we write them down as triples; we can read them in a way that sounds a bit like English. The PatientVisit has careRecipient Person, the Person is a careRecipientOn a PatientVisit.
In the two examples we have just seen, the pair of names for the property and its inverse are relatively easy to come up with, and are very suggestive of their inverse relationship. An equally accurate name for the inverse of parentOf would be childOf. This sounds good, but you do lose the direct clue in the names that one is the inverse of the others. It is not always easy to think of good names for inverse properties.

Figure 2.10: Equivalence using an inverse property.

In Figure 2.10 we show both the metadata and the data for this example. This is the same triple that we saw in Figure 2.4, the only difference is that we have added the name of the inverse property (non-bold in parentheses under the main property name). We have also removed some of the attributes to keep things simple. The data portion with the two linked individuals states that:

1. _:JohnDoeDoctorVisit12Jan rdf:type :PatientVisit
2. _:PatientJohnDoe :careRecipientOn _:JohnDoeDoctorVisit12Jan
3. _:PatientJohnDoe rdf:type :Person

Since we defined Patient to be equivalent to anything for which the above statements are true, the inference engine will automatically categorize John Doe to be of type Patient. We have a very similar situation to the patient visit example. First:

IF: we know an individual is a Patient,
THEN: we know that (1) it is a Person and (2) that is a careRecipientOn some PatientVisit.

Going the other way around:

IF: we know that an individual is (1) a Person and (2) a careRecipientOn some PatientVisit
THEN: we know the individual is a Patient.
2. HOW DO WE SAY IT IN OWL?

That’s very nice in principle. If you want to see what happens in practice, you can start using an ontology editor like Protégé⁶ to create this simple ontology and single data triple and then run inference. This is illustrated in Figure 2.11. The top part of the figure shows a portion of a screen from Protégé with the inferences highlighted in yellow. In everyday English, the inferred assertions are:

1. John Doe is the care recipient on a particular patient visit (upper right of figure); and
2. John Doe is a patient. This is true because of what we said above (upper left).

The first inference is justified because the inverse triple is directly asserted. The second inference is justified by the second inference rule above. The tool itself will automatically generate an explanation that says what triples were used to conclude that John Doe is an instance of Patient. This is shown in the lower part of the Figure 2.11.

Note that an intermediate inference was made, in order to conclude John Doe is a Patient. Namely, if a relationship holds in the forward direction, the inverse relationship automatically holds. For example, if we say Peter hasChild Patricia, we can infer that Patricia hasParent Peter. The same thing is going on here. We assert the triple going in the direction from the PatientVisit to the Person receiving care, and the inverse relationship automatically holds. This is important because the definition of Patient used the inverse relationship: careRecipientOn.

Figure 2.11: Inferring John Doe to be a Patient.

⁶ http://protege.stanford.edu/.
2.3.5 OTHER WAYS TO SPECIFY MEANING

There are a few more things that commonly arise in creating OWL ontologies. There are some additional ways to create classes from other classes. We can specify more meaning about properties and we can do some work to help the inference engine find simple mistakes.

Creating Classes from Other Classes

In the example above we saw how the class Patient was defined to be anything that was both a Person and that was a careRecipientOn some PatientVisit. Our graphical notation intentionally echoes this very closely. In a simplified Manchester syntax it looks like:

Class: doe:Patient
   EquivalentTo: doe:Person and (doe:careRecipientOn some doe:PatientVisit)

Let us consider some examples using a simpler variation of the same pattern. Suppose we needed to represent the class PerformedProcedure. Let’s say we already have a class for Procedure, and another one for HistoricalEvent, where the latter is any event that has already happened. So a PerformedProcedure is something that is both a Procedure and a HistoricalEvent. In the world of vehicles, we might define the class Motorcycle to be something that is both a TwoWheeledVehicle and a MotorizedVehicle. The simplified Manchester syntax for PerformedProcedure is:

Class: doe:PerformedProcedure
   Annotations: rdfs:comment
                 "A procedure that has already been performed."
   EquivalentTo: doe:HistoricalEvent and doe:Procedure

Figure 2.12: Class expressions using intersection.

7 For readability, I sometimes modify whitespace and indenting.
2. HOW DO WE SAY IT IN OWL?

The pattern is the same in all three cases. We are taking two base classes, \((C1, C2)\) and combining them to define a third class, \(C3\). Specifically, we are saying that a class \(C3\) is equivalent to the expression “\(C1 \text{ and } C2\).” To be a member of the third class, you have to be a member of \(C1\) and a member of \(C2\).

Our Patient example also fits this pattern. \(C1\) is Person, \(C2\) is the property restriction class “careRecipientOn some PatientVisit” that we might have called ThingThatIsACareRecipientOnSomePatientVisit, if we gave it a name. The set of all members of the class Patient, is precisely the set of all things that are members of both \(C1\) and \(C2\).

It turns out that the formal mathematical logic underpinning OWL regards every class as a set. The \(\text{and}\) operation we have just seen corresponds to set intersection. If we can have set intersection then we should of course also have set union. We do; it entails using an “\(\text{or}\)” operation in a completely analogous manner as we used “\(\text{and}\)” above. Below are two very common examples, one in healthcare and one for almost any enterprise. There is a third set operation that is used much less frequently corresponding to set complement. We will examine that in Section 5.3.2.

![Figure 2.13: Class expression using union.](www.morganclaypoolpublishers.com/owl)

More Meaning for Properties

Subproperties

We can also add meaning to a property by saying it is a more specialized version of another property. In most large enterprises there are regular employees as well as a variety of contractors that do valuable work. There are important legal and tax implications regarding this distinction, so the ontology must support it. However, for some uses, you may just want to get a count, or a dollar amount that relates to persons that work for the enterprise, and you don’t care whether they are contractors or employees, you want both.

To accomplish this, we start with a general property called \texttt{worksFor} that connects a person to the person or company that they do work for. We also need specialized properties that indicate
whether the relationship is an employee/employer one, or whether it is about contracting. We might call these properties: employedBy and contractorFor. This means that:

\[
\text{IF: } X :\text{contractorFor} \ Y \quad \text{IF: } X :\text{employedBy} \ Y \\
\text{THEN: } X :\text{worksFor} \ Y. \quad \text{THEN: } X :\text{worksFor} \ Y.
\]

Figure 2.14: Subproperty hierarchy.

In general, for any property SP that is a specialized version of property P in the same way as in the worksFor example, we have:

\[
\text{IF: the triple } <X \ SP \ Y> \text{ is asserted} \\
\text{THEN: the triple } <X \ P \ Y> \text{ is necessarily true, and will be inferred.}
\]

Having a more specialized property is similar to having a more specialized class using the subClassOf relationship.

- IF: something is a member of the more specialized class
  THEN: it is also a member of the more general class.

- IF: the specialized property relationship holds between X and Y
  THEN: the general property relationship also holds between X and Y.

So, just as we say the specialized class is a rdfs:subClassOf the more general class, we also say the specialized property is a rdfs:subPropertyOf the more general property. This is why the same line and arrow head is used to mean both subclass and subproperty in the e6 notation.

Property characteristics

Recall the property, careRecipient, which points from a PatientVisit to a Person. We noted that for the healthcare company we are modeling, there is only ever one person that is receiving care on a PatientVisit. We saw that OWL gives us a way to say that in the ontology. Specifically, we say that the property is functional. Most properties are not functional, but many
2. HOW DO WE SAY IT IN OWL?

Every time you create a property, you should decide whether it is functional. This is but one of several characteristics that different properties may or may not have.

**Exercise: 4:** Can you think of other functional properties? Can someone have more than one first name? more than one last name? Can there be more than one person providing care on a given PatientVisit?

Another characteristic that commonly arises relates to what happens if you have a chain of triples linked by the same property. If you want to order people by their age, you might have an “older than” property. If Tommy is older than Jill, and Jill is older than Peter, then common sense tells us that Tommy is older than Peter. This relationship has the characteristic that if A is related to B and B is related to C, then A is related to C.

We have already seen an example of this characteristic, which is called *transitivity*. Do you remember it? We wanted the inference engine to have enough common sense to know that if a cancer patient is a subclass of patient, and if patient is a subclass of person, then a cancer patient is a subclass person, without having to say it explicitly. In this case the property is the OWL construct, `rdfs:subClassOf`. Just like we can specify in OWL that a property is functional, we can also specify whether a property is transitive. In short, a transitive property is one that you can chain as many together as you want, and the property also holds from the beginning to the end of the chain. Consider some of the other properties we have seen, can you find any that are also transitive?

Just like you should ask whether every property is functional, you should also ask whether it is transitive. This is how the inference engine knows how to make the right inferences. You use whatever OWL constructs are available to express the semantics of the things you are creating. The ontology gives meaning to the data, and adding more meaning to the ontology adds more meaning to the data.

**Exercise: 5:** Is the property, `careRecipient`, transitive? Why or why not?

**Common Techniques for Helping the Inference Engine Find Mistakes**

Recall that there are two main benefits of inference. One is to generate new triples that logic and common sense tell you are true, without the need to write them down explicitly. This saves time up front; it also makes the ontology easier to maintain.

The other main benefit of inference is to help find mistakes. Some mistakes are very simple errors, not much more than a typo, others are much more subtle, and would be nearly impossible for a human to spot just by looking at the ontology. An example of a simple mistake would be to create an individual and put it in the wrong class. You might say an individual is a member of the class, `Patient`. Somewhere else, you or someone else might accidentally say that the same indi-
individual is member of the class \texttt{PatientVisit}. Common sense tells you that you cannot be both a \texttt{Patient}, which is a \texttt{Person} and also a \texttt{PatientVisit}, which is an \texttt{Event}. We want the inference engine to catch this mistake. We need a way to say that some combinations of classes do not have any members in common. This is another idea relating to sets. To say that there are no members in common means that the intersection is empty. In OWL we do this by saying that two classes are disjoint.

We want the inference engine to know that the two classes \texttt{PatientVisit} and \texttt{Patient} never overlap. There are far too many combinations of non-overlapping classes for it to be practical to manually make disjointness statements for each pair. It turns out we do not need to. We can go up to the more general classes like \texttt{Person} and \texttt{Event}, and just make those disjoint instead.

![Diagram of classes and disjoint relationships]

\textbf{Figure 2.15:} High-level disjoints.

\textbf{Exercise: 6:} Explain why this works. How can just specifying the two high-level classes being disjoint make the others also disjoint?

\section*{2.3.6 PAUSE AND REFLECT}

Let’s pause for a moment, because there is quite a lot going on here. For starters, you have just seen everything you will need for 90\% of the time you spend building ontologies. The main effort now is to learn the ins and outs of using these constructs. There are quite a few additional features that you need from time to time, and we will introduce and explain them in subsequent chapters. Your initial focus should be to learn these core constructs cold. With experience, you will see the need for the more advanced features of OWL.

\section*{2.4 SUMMARY OF KEY OWL CONCEPTS AND ASSERTIONS}

The core concepts in OWL are: individuals, classes, and properties. The ontology itself is a set of assertions (i.e., statements) of various sorts using the above core concepts. An assertion has a sub-
2. HOW DO WE SAY IT IN OWL?

Object, predicate and object and is represented as a triple (see Table 2.2). Below we briefly summarize some key points about namespaces and then review the most common ontology assertions rendered in a simplified Manchester syntax of OWL, along with an English translation.

2.4.1 VOCABULARIES AND NAMESPACES

When you create an ontology you are creating a vocabulary of terms to define the subject matter of interest. It is much more than just a vocabulary, of course, because you are carefully defining the meaning of the terms using the logic-based OWL language.

You use the OWL vocabulary to create terms and expressions and give them meaning. Recall that OWL uses other vocabularies as well, mainly rdf and rdfs. After you create your own vocabulary of classes and properties, you can use it to create your data. Unlike most databases, the metadata and the data are expressed using the same representation (triples) and are loaded into and queryable from the same in the same triple store.

You must create globally unique IRIs to identify the resources you create, chiefly classes, properties, and individuals. Most IRIs look just like the URLs we commonly use in web browsers. An IRI has a long name that is usually shortened for convenience using namespace prefixes. For example, owl:Class is the short form for the full IRI: http://www.w3.org/2002/07/owl#Class where “owl” is the namespace prefix, and “Class” is the local name of the IRI. They are joined by a “:”.

Selecting names is important for understanding the ontology. They should be highly suggestive of their meaning, and any naming conventions that you choose should be used consistently.

2.4.2 INDIVIDUALS AND CLASSES

• **Individual**: An individual is a specific thing e.g., the house you live in, Joe Brown’s health insurance policy, Janet Smith. The rdf:type construct is used to say what class an individual is a member of.

• **Class**: A class represents a kind of thing, e.g., building, health plan policy, patient. A class corresponds to the set of all things of that specific kind.

• **subClassOf**: A class is said to be a subclass of another class if it represents a more specific kind of thing than its parent class, e.g., a house is a specific kind of building. Every member of the subclass is also a member of the superclass. For example, if your particular house is a member of the class, House, then an inference engine can draw the conclusion that your house is also a member of the class, Building, without your having to say so.
The following are the main kinds of assertions relating to individuals and classes.

- **Create a class:**
  - **Class: Patient**
    - Patient is a class referring to the set of all patients.
  - **Class: Person**
    - Person is a class, referring to the set of all persons.

- **Specify a subclass relationship:**
  - **Class: Patient**
    - **SubClassOf: Person**
    - Patient is a subclass of Person, meaning that all patients are persons.

- **Create an individual and give it a type:**
  - **Individual: _JohnSmith**
    - **Annotations:**
      - rdfs:label “John Smith”
    - **Types:**
      - Person
  - The individual with unique identifier _JohnSmith is a member of the class Person, and uses the human readable label “John Smith.”

- **Set one class to be equivalent to another class**
  - **Class: Person**
    - **EquivalentTo:**
      - HumanBeing

    The two classes Person and HumanBeing have exactly the same members.

### 2.4.3 Properties

**Property:** A property is a way to relate individuals to each other or two literals. There are three main kinds as well as a way to specify subproperty relationships.

- **Object property:** An object property relates two Individuals, e.g., John isParentOf Peter.

- **Data property:** A data property associates a literal with an Individual, e.g., John hasAge 33.
2. HOW DO WE SAY IT IN OWL?

- **Annotation property:** An annotation property is just like an object property or data property, except that they do not affect conclusions that the reasoner draws. They are used for such things as labels and comments, e.g., Building [has] comment “An enclosed structure that is to provide shelter.”

- **subPropertyOf:** A property is said to be a subproperty of another property if it represents a more specific kind of relationship than its parent property, e.g., `employedBy` is more specific than `worksFor`.

  Note that, strictly speaking, `rdfs:subClassOf` is also a property, but it is different from the rest in that it relates two classes rather than individuals or literals. This is OK because it is part of OWL itself. When we speak of properties we generally refer to properties created by the user, not those that are part of OWL. The following are the main kinds of assertions relating to properties.

  - **Create an object property and specify its inverse:** for example: specify a way that two individuals can be related to each other from both perspectives,
    - `ObjectProperty: parentOf
      ObjectProperty: hasParent
      InverseOf:
        parentOf`
    - `parentOf` and `hasParent` are object properties. The first means that the subject is the parent of the object. The second is the inverse of the first, so it means that the object is the parent of the subject.

  - **Create an object property assertion:** for example: assert that one individual has a certain relationship with another individual,
    - `Individual: _JohnSmith
      Facts: parentOf PeterJones`
    - John Smith is the parent of Peter Jones.

  - **Create a data property:** for example: specify a way that individuals can be related to a literal,
    - `DataProperty: hasAge`
    - `hasAge` is a data property

  - **Create a data property assertion:** for example: assert that an individual has an attribute expressed as some literal value.
• **Specify a subproperty relationship**
  
  - **ObjectProperty**: employedBy
    - **SubPropertyOf**: worksFor
  
  - Being employed by someone is a more specific relationship than [just] working for them.

### 2.4.4 CLASS EXPRESSIONS AND RESTRICTIONS

OWL supports specifying meaning in a variety of ways by building up potentially complex expressions from the core primitives. The following are the main ways to create assertions using class expressions.

- **Define a class using set union**
  
  - **Class**: Party
    - **EquivalentTo**: Person or Organization
  
  - Every member of the class **Party** is either a member of the class **Person** or it is a member of the class **Organization**. In addition, any member of either of the classes **Person** or **Organization** is also a member of the class **Party**. Set union here means logical “and.”

- **Define a class using set intersection**
  
  - **Class**: Motorcycle
    - **EquivalentTo**: TwoWheeledVehicle and MotorizedVehicle
  
  - Every member of the class **Motorcycle** is a member of both of the classes **TwoWheeledVehicle** and **MotorizedVehicle**. In addition, anything that is a member of both of the classes **TwoWheeledVehicle** and **MotorizedVehicle** is member of the class **Motorcycle**. Set intersection here means logical “and.”

- **Specify what properties members of a class must have.** By itself, creating a class and giving it a name doesn't tell us anything about the class. We can say more about a class by specifying what properties it must have.
2. HOW DO WE SAY IT IN OWL?

- **Class**: Person
  - **SubClassOf**: 
    - hasPart exactly 1 Heart

- Every member of the class Person has exactly 1 Heart. The expression “hasPart exactly 1 Heart” is a property restriction class that does not have a name. It means: anything that is in the hasPart relationship with exactly 1 member of the class Heart.

- We can give a property restriction class a name using equivalence
  - **Class**: ThingWithExactlyOneHeart
  - **EquivalentTo**: 
    - hasPart exactly 1 Heart

  The class ThingWithExactlyOneHeart has the same members as the set of all things that are in the hasPart relationship with exactly one member of the Heart class.

Notice that in the latter examples we made new classes by combining other classes and properties in useful ways. We created the class Party using the expression: “Person or Organization.” The expression: “hasPart exactly 1 Heart” is also a class; we can give it name to if we choose. There are other ways to create classes, e.g., using set intersection and complement.

What we did above may not be the most obvious way to say a Person has exactly one Heart—but that’s the way it is done in OWL. Think of it this way. Saying that Patient is a subclass of Person says every Patient is a Person. Similarly, saying that every Person is a subclass of ThingWithExactlyOneHeart means that every Person is also a ThingWithExactlyOneHeart. Don’t worry if you don’t follow all of this now—it takes a while to wrap your head around it. Chapter 5 explains this in much more detail with many more examples.

2.4.5 DRAWING CONCLUSIONS

Reasoners, also known as inference engines, can draw new conclusions from the information that is asserted into the ontology. For example, if John Smith is a Person, and Party is defined as a Person or Organization, then the reasoner can conclude that John is a Party. If an individual is known to be a Person, the reasoner can conclude that it has exactly one Heart. Many other conclusions can be drawn.

One important use of drawing conclusions is to ensure that the ontology is logically consistent. For example, suppose we explicitly declare that nothing can be both a Person and a Building. If a mistake is made in creating the ontology, if an individual is found (by any combination of

www.morganclaypoolpublishers.com/owl
direct assertion or inference) to be both a Person and a Building, the reasoner draws attention to the problem and provides an explanation based on a trace of the reasoning that led to the error.

2.5 SUMMARY LEARNING

Saying Things with Triples

The first chapter identified what things need to be said when creating an ontology. This chapter explained how to say those things in OWL. An OWL ontology is a set of assertions in the form of triples, each composed of a subject, a predicate and an object. The subject is an individual that has an IRI. The predicate is a property with an IRI. The object may either be another individual or a literal. IRIs and namespaces are a key part of OWL. Anything with an IRI is called a resource. There are various syntaxes for OWL, including Turtle and Manchester syntax.

We introduced formal terminology for the key concepts introduced in the previous chapter.

• owl:Thing is used to create individual things.
• owl:Class is used to create kinds of things
• rdf:type is used to say that an individual is an instance of a certain kind
• rdfs:subClassOf is used to say one kind is a specialization of another kind
• owl:ObjectProperty is used to create a way that two things can be related to another.
• owl:DatatypeProperty is used to create a way that something can be related to a literal.

Although everything is represented as triples, there is an important distinction between
• Data: the data you create by creating individuals and connecting them to other individuals and literals; and
• Metadata: the metadata which gives meaning to the data and supports inference.

Expressing Meaning

Meaning is expressed in various ways. There are class and property hierarchies that work very similarly. Properties may have domains which say what class the subject of a triple using the property must be a member of. They have ranges which say what class the object of a triple using the property must be a member of. Properties have certain characteristics, such as being functional or transitive.
2. HOW DO WE SAY IT IN OWL?

There are numerous ways to create class expressions, chiefly including property restrictions and set operations like union (or) and intersection (and). Property restrictions are used to say what properties individuals of a given class have. One important use of inverse properties is for creating property restrictions.

Class equivalence can be used along with inference to automatically determine what classes an individual belongs to. That is an important difference between using subclass and class equivalence. The inference engine can also be used along with high-level disjoints to determine logical inconsistencies with an ontology.

Final Remarks

This completes our comprehensive introduction to OWL. Before we continue to delve more deeply into what we have learned, we will pause and consider some fundamentals on which OWL is based. That is the subject of the next chapter.