5 RDF and Inferencing

In Chapter 1XXX, we introduced the notion of "dumb" data, and how a more connected web infrastructure can result in behavior that lets smart applications perform to their potential. One way to make data less dumb is to provide an integrated representation of information, and present information by making queries to this representation. In the semantic web, we have seen how RDF allows this integrated representation to be distributed over the web. But we have also stressed the importance of modeling to making sense of this network of data. How can our web infrastructure allow a model to help us to make sense of an integrated network of data? Let's look at another simple example of dumb data to show how this can work.

Suppose you hit the web page of an online clothing retailer, and you search for "chamois" in the category of "Shirts". Your search comes up empty. You are surprised, because you were quite certain that you saw a chamois Henley in the paper catalog that landed in your mailbox. So you look up the unit number in the catalog, and search for that. Sure enough, there is the chamois Henley. Furthermore, you find that "Henley" is shown as a sub-category in the broad category of "Shirts." "That's dumb," you mutter to yourself, "if it finds it under 'Henley,' it should find it under 'Shirts.' What's the matter with this thing?"

What would it take to make this example less dumb? We want any search, query, or other access to the data that references "Shirts" to also look at "Henley". What is special about the relationship between "Shirts" and "Henley" to make us expect this? That is what we *mean* when we say "Henley is a sub-category of Shirts." How can we express this meaning in a way that is consistent and maintainable?

One solution to this problem is to leverage the power of the query; after all, in conventional database applications, it is in the query where relationships between data elements are elaborated. In the case of the Henley shirts, a complex query could ask, "Show me all items in category 'Shirts,' or in any sub-category of 'Shirts', or any sub-sub-category of 'Shirts,' and so on." Depending on the syntax of any particular query language, this could be a bit cumbersome to express, but there is no essential difficulty with it. In fact, just such a solution is available using many semantic web tools. Just as in the relational database case, a single RDF store can be queried in different ways to create variant presentations of a single data store, ensuring information consistency across the various views. Relationships like the *sub-category* relationship in this example are represented in the query language. Systems that support this style of solution provide query languages that make this sort of query convenient, so that the "and so on" doesn't require the query writer to write a program loop.

In contrast to this approach, the Semantic Web provides a model of data expression that allows for explicit representation of the relationship between various data items. In this sense, it genuinely allows a data modeler to create data that is more connected, better integrated, and dare we say, smarter. Data in which the consistency constraints on the data can be expressed *in the data itself*. It is for this reason that some people have described the Semantic Web as allowing us to model 'smart data." By this we don't mean that the data is going to start diagnosing cancer or solving complex problems, but the data can describe something about the way it should be used.

As an alternative to the "smart query" approach, the Semantic Web stack includes a series of layers on top of the RDF layer to describe consistency constraints in the data.

They key to these levels is the notion of *inferencing*. In the context of the Semantic Web, inferencing simply means that given some stated information, we can determine other, related information that we can also consider as if it had been stated. Inferencing is a powerful mechanism for dealing with information, which can cover a wide range of elaborate processing. For the purposes of making our data more integrated and useful, very simple inferences are actually more useful than elaborate ones. As a simple example, in the previous chapter, we saw how to write a complex query to make up for the fact that although we stated that Anne Hathaway married Shakespeare, we did not assert that Shakespeare married Anne Hathaway. It is this sort of mundane consistency completion of data that can be done with inferencing in the Semantic Web. While inferencing of this sort seems trivial from the point of view of the natural world (after all, doesn't everyone *just know* that this is the way marriage works?), it is just this sort of correlation that is missing in dumb data applications.

5.1 Inference in the Semantic Web

To make our data seem more connected and consistently integrated, we need to be able to add relationships into the data that will constrain how the data is viewed. We want to be able to express the relationship between 'Henleys' and 'Shirts' that will tell us that any item in the 'Henleys' category should also be in the 'Shirts' category. We want to express the fact about locations that says that if a hotel chain has a hotel at a particular location, then that location is served by a hotel in that chain. We want to express the list of planets in terms of the classifications of the various bodies in the solar system.

Many of these relationships are familiar to information modelers in many paradigms. Let's take the relationship between 'Henleys' and 'Shirts' as an example.

Taxonomists and thesaurus writers are familiar with the notion of "broader term;" 'Shirts' is a *broader term* than 'Henleys'. Object oriented programmers are accustomed to the notion of *subclasses* or *class extensions*; 'Henleys' is a subclass of, or extends, class 'Shirts'. In the RDF Schema language to be described in the next chapter, we say 'Henleys' *subClassOf* 'Shirts'. It is all well and good to say these things, but what do they mean?

Thesauri and taxonomies take an informal stance on what these things mean in a number of contexts. If you use a broader term in a search, you will also find all the entries that were tagged with the narrower term. If you classify something according to a broad term, you may be offered a list of the narrower terms to choose from, to focus your classification.

Many readers may be familiar with terms like "Class" and "subclass" from their use in Object-Oriented Programming (*OOP*). There is a close historical and technical relationship between the use of these and other terms in OOP and their use in the Semantic Web, but there are also important and subtle differences. OOP systems take a more formal, if programmatic, view of class relationships than that taken by thesauri and taxonomies. An object whose type is 'Henleys' will respond to all messages defined for object of type 'Shirts'. Furthermore, the action associated with this call will be the same for all Shirts, unless a more specific behavior has been defined for Henleys, and so on. The semantic web also takes a formal view of these relationships, but in contrast to the programmatic definition found in OOP, the semantic web bases the meaning of these things on the notion of Inference.

The Semantic Web infrastructure provides a formal and elegant specification of the meaning of the various terms like subClassOf. For example, the meaning of "*B* is a SubClassOf *C*" is "Every member of class *B* is also a member of class *C*". This specification is based on the notion of inference. From the information "*x* is a member of *B*," one can derive the new information, "*x* is a member of *C*".

For the next several chapters, we will introduce terms that can be used in an RDF model, along with a statement of what each term means. This statement of meaning will always be in the form of an inference pattern: "Given some initial information, the following new information can be derived." This is how the RDF Schema language (RDFS, Chapter **Error! Reference source not found.**) and the Web Ontology Language (OWL, Chapter **Error! Reference source not found.**) work. We will take our first example from RDFS. The details of RDFS are given in a systematic fashion in Chapter **Error! Reference source not found.**.

The pattern for the *subClassOf* in RDFS says that:

IF

```
?A rdfs:subClassOf ?B .
```

AND

```
?x rdf:type ?A .
```

THEN

```
?x rdf:type ?B .
```

In plain English, this says that if one class (A) is a subclass of another class (B), and there is any individual (x) that belongs to class A (where by 'belongs to' we mean it is related by the predicate *rdf:type*), then that individual x also belongs to class B. This simple statement is the entire definition of the meaning of *subClassOf* in the RDF

Schema language. We will refer to this rule for the rest of this chapter as the *type propagation* rule. This definition is consistent with the informal notion of broader term in a thesaurus or taxonomy, since it is natural to think that any individual listed under 'Henleys' should also be listed under 'Shirts'.

The Semantic Web definition of subClassOf is consistent to some extent with the definition of sub-class or extension in Object Oriented Programming (OOP). In OOP, an instance of some class responds to the same methods in the same way as instances of its superclass. In Semantic Web terms, this is because that instance is also a member of the superclass, and hence must behave like any such member. For example, the reason why an instance of class 'Henleys' responds to methods defined in 'Shirts' is because the instance actually *is* also a member of class 'Shirts'.

This consistency is misleading when, in the OOP system, the subclass defines an override for a method defined in the superclass. In Semantic Web terms, the instances of Henleys are still instance of Shirts, and should respond accordingly. But in most OOP semantics, this is not the case; the definitions at Henleys take precedence over those at Shirts, and thus Henleys need not actually behave like shirts at all. In the logic of the Semantic Web, this is not allowed.

5.1.1 Virtues of an Inference-based Semantics

Inference patterns constitute an elegant way to define the meaning of a data construct. But is this approach really useful? Why is it a particularly effective way to define the meaning of constructs in the Semantic Web?

Since our data is living in the Web, a major concern for making our data more useful is to have it behave in a consistent way when it is combined with data from multiple sources. The strategy of basing the meaning of our constraint terms on inferencing provides a robust solution to understanding the meaning of novel combinations of terms. Taking *subClassOf* as an example, it is not out of the question for a single class to be specified as *subClassOf* two other classes. What does this mean?

In an informal thesaurus setting, the meaning of such a construct is decided informally – what do we *want* such an expression to mean? Since we have a clear but informal notion of what *broader term* means, we can use that intuition to argue for a number of positions, including but not limited to deciding that such a situation should not be allowed, to defining search behavior for all terms involved. When the meaning of a construct like *broader term* is defined informally, the interpretation of novel combinations must be resolved by consensus or authoritative proclamation.

OOP also faces the issue of deciding an appropriate interpretation for a single subclass of two distinct classes. The issue is known as *multiple inheritance,* and is well-discussed in OOP circles. Indeed each OOP modeling system has a response to this issue, ranging from a refusal to allow it (C++), a distinction between different types of inheritance (*interface* vs. *implementation* inheritance, e.g., Java), to complex systems for defining such things (e.g., the Meta-Object Protocol of the Common Lisp Object System). Each of these provides an answer to the multiple inheritance question, and each is responsive to particular design considerations that are important for the respective programming language.

In an inference-based system like the Semantic Web, the answer to this question (for better or worse) is defined by the interaction of the basic inference patterns. How does multiple inheritance work in the RDF Schema language? Just apply the rule twice. If A is *subClassOf* B and also A is also *subClassOf* C, then any individual *x* that is a member of A will also be a member of B and of C. No discussion is needed, no design decisions. The meaning of *subClassOf*, in any contexts, is given elegant expression in a single simple rule: the type propagation rule. This feature of inference systems is particularly suited to a Semantic Web context, in which novel combinations of relationships are bound to occur as data from multiple sources is merged.

5.1.2 Where's the smarts?

An inference-based system for describing the meaning of Semantic Web constructs is elegant and useful in a distributed setting, but how does it help us make our data more useful? In order for our application to behave differently, we will need a new component in our deployment architecture, something that will respond to queries based not only on the triples that have been asserted, but also on the triples that can be inferred based upon the rules of inference. This architecture is shown in Figure 5-1, and is very similar to the RDF query architecture shown in **Error! Reference source not found.**.

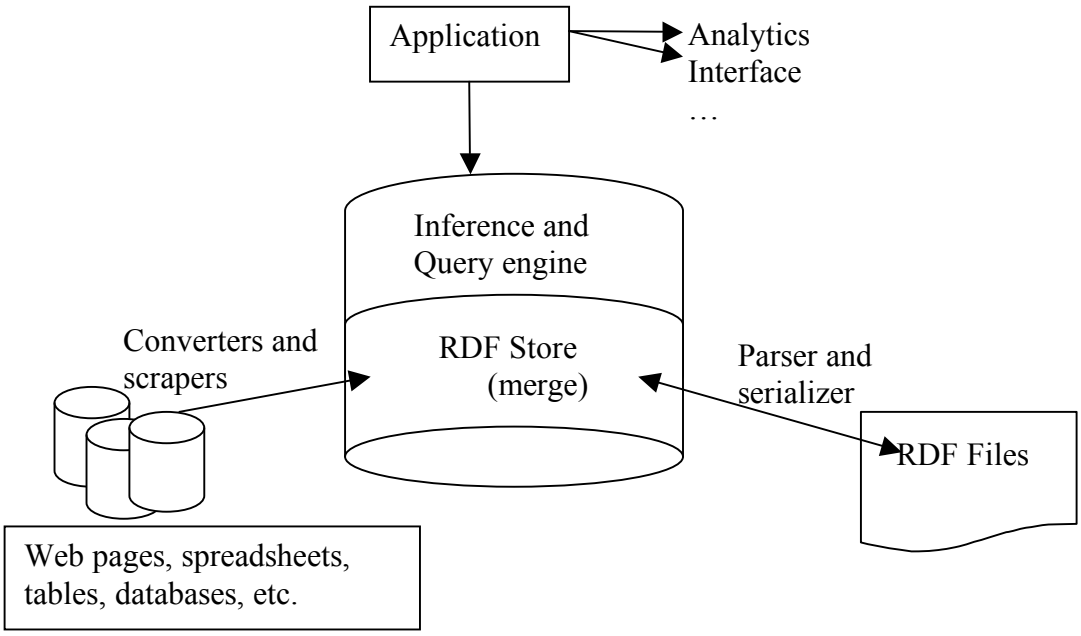

Figure 5-1 Semantic Web Architecture with Inferencing

The new item in this architecture is an inferencing component that stands with the query component between the application and the RDF data store. The power of the inferencing query engine is determined by the set of inferences it supports. An RDFS inference query engine supports a small set of inferences defined in the RDFS standard; an OWL inference query engine supports the larger set of OWL inferences.

5.1.2.1 Example. Simple RDFS Query

Suppose we have an RDFS inference query engine working over an RDF store that contains just the following two triples:

```
shop:Henleys rdfs:subClassOf shop:Shirts .
shop:ChamoisHenley rdf:type shop:Henleys .
```

Suppose we have a SPARQL triple pattern that we use to examine these triples, thus:

```
?x rdf:type shop:Shirts .
```

In a plain RDF query situation, this pattern will match no triples, because there is no triple with predicate *rdf:type* and object *shop:Shirts*.. However, since the RDFS inference standard includes the type propagation rule listed above, with an RDFS inferencing query engine, one result will be returned, namely

```
?x = shop:ChamoisHenley
```

5.1.3 Asserted Triples vs. Inferred Triples

It is often convenient to think about inferencing and queries as separate processes, in which an inference engine produces all the possible inferred triples, based on a particular set of inference rules. Then, in a separate pass, an ordinary SPARQL query engine runs over the resulting augmented triple store. It then becomes meaningful to speak of *asserted triples* vs. *inferred triples*.

Asserted triples, as the name suggests, are the triples that were asserted in the original RDF store. In the case where the store was populated by merging triples from many source, all the triples are asserted. Inferred triples are the additional triples that are inferred by one of the inference rules that govern a particular inference engine. It is, of course, possible for the inference engine to infer a triple that has already been asserted. In this case, we still consider the triple to have been asserted. It is important to note that

the distinction between inferred and asserted triples is a distinction for rhetorical and pedagogical purposes only; the inference engine will draw exactly the same conclusions from an inferred triple as it would have done, had that same triple been asserted.

5.1.3.1 Example. Asserted vs. Inferred triples

Even with a single inference rule like the type propagation rule, we can show the distinction of asserted vs. inferred triples. Suppose we have the following triples in a triple store:

```
shop:Henleys rdfs:subClassOf shop:Shirts .
shop:Shirts rdfs:subClassOf shop:Menswear .
shop:Blouses rdfs:subClassOf shop:WomensWear .
shop:Oxfords rdfs:subClassOf shop:Shirts .
shop:Tshirts rdfs:subClassOf shop:Shirts .
shop:ChamoisHenley rdf:type shop:Henleys .
shop:ClassicOxford rdf:type shop:Oxfords .
shop:ClassicOxford rdf:type shop:Shirts .
shop:BikerT rdf:type shop:TShirts .
shop:BikerT rdf:type shop:MensWear .
```

These triples are shown graphically in Figure 5-2.

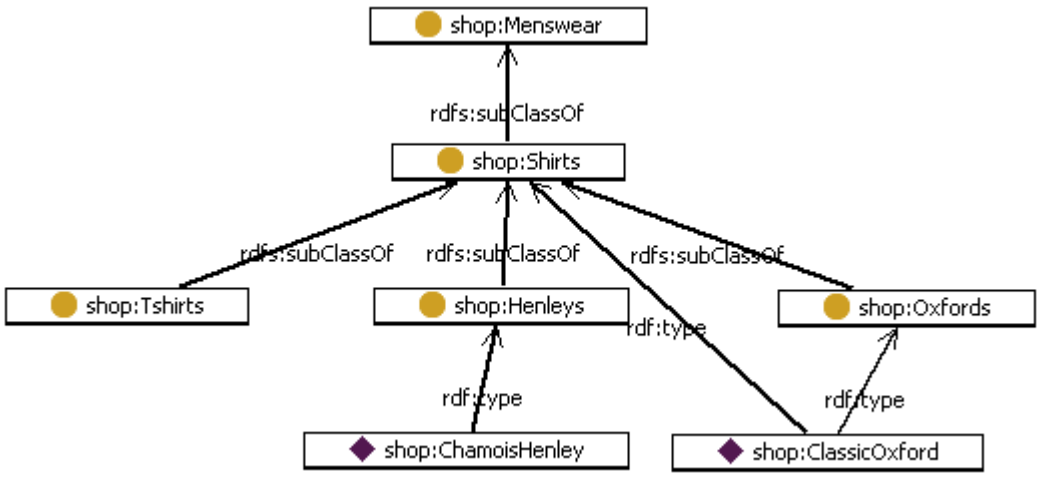

Figure 5-2. Asserted triples in the catalogue model.

An inferencing query engine that enforces just the type propagation rule will draw the following inferences

```
shop:ChamoisHenley rdf:type shop:Shirts .
shop:ChamoisHenley rdf:type shop:MensWear .
shop:ClassicOxford rdf:type shop:Shirts .
shop:ClassicOxford rdf:type shop:MensWear .
shop:BikerT rdf:type shop:Shirts .
shop:BikerT rdf:type shop:MensWear .
```

Some of these triples were also asserted; the complete set of triples over which queries will take place is as follows, with inferred triples in italics:

```
shop:Henleys rdfs:subClassOf shop:Shirts .
shop:Shirts rdfs:subClassOf shop:Menswear .
shop:Blouses rdfs:subClassOf shop:WomensWear .
shop:Oxfords rdfs:subClassOf shop:Shirts .
shop:Tshirts rdfs:subClassOf shop:Shirts .
shop:ChamoisHenley rdf:type shop:Henleys .
shop:ChamoisHenley rdf:type shop:Shirts .
shop:ChamoisHenley rdf:type shop:MensWear .
shop:ClassicOxford rdf:type shop:Oxfords .
shop:ClassicOxford rdf:type shop:Shirts .
shop:ClassicOxford rdf:type shop:MensWear .
shop:BikerT rdf:type shop:TShirts .
shop:BikerT rdf:type shop:Shirts .
shop:BikerT rdf:type shop:MensWear .
```

All triples in the model, both asserted and inferred, are shown in Figure 5-3.

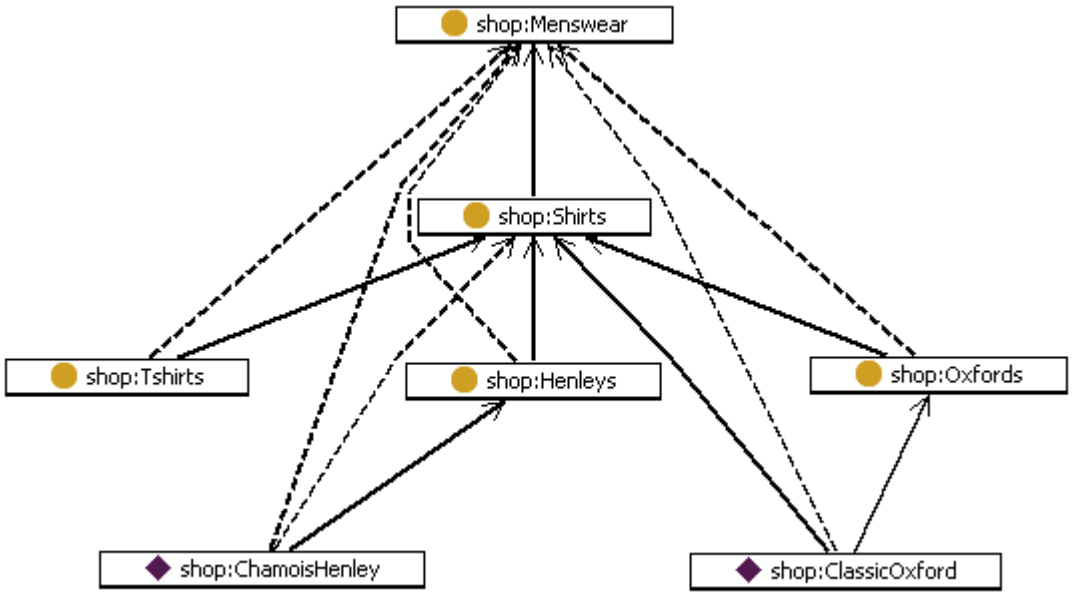

Figure 5-3. All triples in the Catalogue model. Inferred triples are shown as dashed lines.

The situation can become a bit more subtle when we begin to merge information from multiple sources, in which each source itself is a system that includes an inference engine. Most RDF implementations provide a capability by which new triples can be asserted directly in the triple store. This makes it quite straightforward for an application to assert any or all inferred triples. If those triples are then serialized (say, in RDF/XML) and shared on the web, another application could merge them with other sources, and draw further inferences. In complex situations like this, the simple distinction of *asserted* vs. *inferred* might be too coarse to be a useful description of what is happening in the system.

5.1.4 When does inferencing happen?

The RDFS and OWL standards define what inferences are valid, given certain patterns of triples. But when does inferencing happen? Where and how are inferred triples stored, if at all? How many of them are there?

These questions are properly outside the range of the definitions of RDFS and OWL, but are clearly important for any implementation that conforms to these standards. It should, therefore, come as no surprise that the answers to these questions can differ from one implementation to another. The simplest approach is to store all triples in a single store, regardless of whether they are asserted or inferred. As soon as pattern is identified, any inferred triples are inserted into the store. This approach is quite simple to describe and implement, but risks an explosion of triples in the triple store. At the other extreme, an implementation could instead never actually store any inferred triples in any persistent store at all. Inferencing is done in response to queries only. The query responses are produced in such a way as to respect all the appropriate inferences, but no

inferred triple is retained. This method risks duplicating inference work, but is parsimonious in terms of persistent storage.

These different approaches have an important impact in terms of change management. What happens if a data source changes, that is, a new triple is added to some data store, or a triple is removed? A strategy that persistently saves inferences will have to decide which inferred triples must also be removed. This presents a difficult problem, since it is possible that there could be many ways for a triple to be inferred. Just because one inference has been undermined by the removal of a triple, does that mean that it is appropriate to remove that triple? An approach that recomputes all inferences whenever a query is made need not face this issue.

Chapter 12XXX discusses the trade-off between these and other hybrid approaches. For the purposes of the next chapters, we will speak in terms of what triples can be inferred, without any commitment to the implementation choices for how they will be represented or stored.

5.1.5 Inferencing as glue

Inferencing is the glue that holds the semantic web together. When anyone says anything about any topic, and someone else says something else, inferencing is the way we make these two pieces of information fit together to provide conclusions that go beyond the facts expressed by the individuals. Even with the single inferencing pattern we have seen so far, we can see an example of how this can work.

Suppose that one source of information provides a list of members of the class *Henleys*, and another source provides a list of members of the class *Oxfords*. Suppose

further that we know that *Henleys* and *Oxfords* are both types of *Shirts*. How can we find a list of all *Shirts*?

As we have seen in section 5.1.3, we can determine that all *Henleys* are also *Shirts* by using a simple inference rule for *subClassOf*. Using this rule again for *Oxfords*, we can determine that all *Oxfords* are also *Shirts*. The merged graph now looks exactly the same as the one shown in Figure 5-3, and the inferencing determines that both the *ClassicOxford* and the *ChamoisHenley* are in fact members of the class *Shirts*.

There are two fundamental components in this simple data integration example. First, there is a *model* that expresses the relationship between the two data sources; in this case, the model consists simply of a single class (*Shirts*) that is has both of the classes to be integrated as subclasses; this is represented with the single concept of *subClassOf*. Second is the notion of inferencing. It is the process of inferencing that applies the model to the two data sources to produce a single, integrated answer. In the subsequent chapters, we will see a variety of ways in which the inferencing standards of the Semantic Web can be used to integrate data. But the two components of model and inferencing are the same for all the examples.

When data seems disconnected it is because some apparently simple consistency is conspicuous by its absence. This is why simple inferences are important; the simpler the missing connection, the dumber the data seems. The inference systems of the Semantic Web seem quite simple (even simplistic) from the point of view of problem solving, but they are very useful for making data more consistent and connected.

5.1.6 Chapter Summary

RDF provides a consistent way to represent data so that information from multiple sources can be brought together and treated as if they came from a single source. But when we want to use that data, the differences in those sources comes out. For instance, we'd like to be able to write a single query that can fetch related data from all the integrated data sources.

The Semantic Web approach to this problem uses a modeling language in which the relationship between the sources can be described. The meaning of the modeling language is specified by inferencing. A modeling constructs meaning is given by the pattern of inferences that can be drawn from it. Information integration is achieved by invoking inferencing before or during the query process; a query returns not only the asserted data, but also inferred information. This inferred information can draw on more than one data source.

We have seen how even very simple inferencing can provide value for data integration. But just exactly what kind of inferencing is needed? There isn't a single universal answer to this question. The Semantic Web standards identify a number of different inferencing modes, intended for differing levels of sophistication of data integration over the Semantic Web.

In the following chapters, we will explore three particular inferencing modes, which were introduced in section **Error! Reference source not found.**, RDFS, OWL-FAST and OWL. They differ only in terms of the inferences that each of them languages allow. RDFS (chapter 6) is a recommendation defined and maintained by the W3C. It operates on a small number of inference rules that deal mostly with relating classes to

subclasses and properties to classes. OWL-FAST (chapter 7) is a mode that we have defined for this book. We have found a particular set of inference patterns to be helpful both pedagogically (as a gentle introduction to the more complex inference patterns of OWL) and practically (as a useful integration tool in its own right). OWL-FAST builds on top of RDFS to include constraints on properties and notions of equality. OWL (chapters 9 and 10) is a recommendation defined and maintained by the W3C which builds further to include rules for describing classes based on allowed values for properties. All of these standards use the notion of inferencing to describe the meaning of a model; they differ in the inferencing that they support.

5.2 Fundamental Concepts

The following fundamental concepts were introduced in this chapter:

Inferencing: The process by which new triples are systematically added to a graph based on patterns in existing triples.

Asserted triples: The triples in a graph that were provided by some data source.

Inferred triples: Triples that were added to a model based on systematic inference patterns.

Inference rules: Systematic patterns defining which triples should be inferred.

Inference engine: a program that performs inferences according to some inference rules. It is often integrated with a query engine.