15 OWL Levels and Logic

This book is about modeling in the context of the semantic web, in particular, using the W3C languages RDF, RDFS and OWL to build and distribute those models. The meaning of these models is given by the inferences that each of these languages define for the models. RDFS provides rudimentary inferencing about types based on class membership and properties. OWL provides a wide array of more advanced modeling features to describe how data can be related.

In **Error! Reference source not found.**, we introduced a subset of OWL that we called RDFS-Plus. There are a number of reasons why someone might define a subset of a language like OWL. In the case of RDFS-Plus, we were interested in a subset of the language that has considerable utility for semantic modeling, but does not place a large burden on either a modeler or someone trying to understand a model. RDFS-Plus includes features that are similar to what can be found in familiar data representation systems like relational database and object-oriented systems. Researchers, implementers and product developers have defined a number of subsets based on modeling expressivity, computational complexity, and often, based on what parts of the OWL language can best be handled by whatever inferencing system they already have.

In the initial OWL specification, the W3C identified three particular variants (or "species") of OWL, which they called OWL-Lite, OWL-DL and OWL-Full. The distinction between OWL-DL and OWL-Full is particularly subtle, and is the topic of much of this chapter. We will examine the motivations behind these variants, and the ramifications these motivations have in terms of technology and modeling style.

Any language will grow as it is used. A semantic web language, even more so. Realizing this, the W3C processes encourage the evolution of languages to provide new functionality while mainting backward backward compatibility. As we shall see, there are a number of useful modeling idioms that are clumsy or impossible in the current definintion of OWL. This chapter outlines the particular features that are being considered in the ongoing process in the W3C for the OWL Recommendation.[1]

15.1 OWL dialects and Modeling philosophy

The original OWL standard defined three subsets of OWL: OWL-Lite, OWL-DL and OWL-Full. The most subtle distinction is that between OWL-Full and OWL-DL. The distinction is primarily one of modeling philosophy – what do we expect of our models?

Normally when we refer to different subsets of a language, we can list the language structures in one subset that are not found in the other. For instance, RDFS has *rdfs:domain*, *rdfs:range*, *rdfs:subPropertyOf*, etc., while RDFS-Plus has all of those, plus some new language features like *owl:inverseOf* and *owl:TransitiveProperty*. We can define how these two languages are similar or different, based on which language terms are available in each one.

In the case of OWL-Full and OWL-DL, the situation is more subtle. Both OWL-Full and OWL-DL use exactly the same set of modeling constructs. That is, if we were to list all the properties and classes that make up OWL-Full, and make the same list for OWL-DL, it would be the same list. In fact, that list is just the list of OWL features you have

[1] Progress and status of the OWL Recommendation is documented at http://www.w3.org/2004/OWL

read about in this book. Everything you have learned applies equally well to OWL-Full and to OWL-DL.

So what is the difference? What was so important that the W3C saw fit to make two distinct standards, if they have the same language constructs, with the same meaning? The distinction between these two variants – or "species" as they are often called – of OWL has to do with how the language constructs are used. The differences in allowed usage are motivated by a difference in the basic philosophy of why one builds models for the semantic web. We will outline these two basic philosophies – one in which emphasis is placed on having models that are provable, and the other in which emphasis is placed on making models that are executable. We examine each of these in turn, along with the intuitions that motivate them.

15.1.1 Provable models

An important motivation for formal modeling (as opposed to informal modeling) is to be precise about what our models mean. In the context of the Semantic Web, this allows us to know precisely and without doubt when concepts coming from two different sources refer to the same thing. Does my notion of James Dean movie correspond to yours? A formal description can help us determine whether or not this is the case. My definition of James Dean movie is one that stars James Dean; yours might include movies *about* James Dean, or maybe movies with the words "James Dean" in the title. How can we tell, if we just have the name "James Dean Movie"? A formal model makes these things clearer. Then it becomes a simple matter of automation to decide whether two classes are the same, or if one subsumes the other, or if they are unrelated.

It is this aspect of modeling that motivates a logical definition of OWL. Each construct in OWL is a statement in a formal logic. The particular logical system of OWL-DL is called *Description Logic*. As the name suggests, Description Logic is a logical system with which formal descriptions of classes, individuals and the relationships between them can be made. The inferences in OWL that have formed the basis of the bulk of this book are formally defined by a model theory based on Description Logic.

Using logic as the foundation of a modeling language makes perfect sense; we can draw upon decades or even centuries of development work in logical formalism. The properties of various logical structures are well-understood. Logic provides a framework for defining all of the inferences that our modeling language will need. But there is one fly in the ointment. In a computational setting like the web, we would like our logic to be processed automatically by a computer. Specifically, we want a computer to be able to determine all of the inferences that any given model entails. That is, we want to be able to automatically determine whether my notion of James Dean movie is the same, more general, or less general than yours.

It is at this point that the details of the logic become important. What does it mean for our modeling formalism if we base it on a logic for which this kind of automation cannot, in principle, exist? That is, what happens if we can't determine whether my notion of James Dean movie is the same as yours? If we view this sort of provable connection as essential to the nature of modeling, then we have failed. We simply cannot tolerate a logic in which this kind of question cannot be answered by automated means in some finite amount of time.

In the study of formal logic, this question is called *decidability*. A logical system is *decidable* just if there in fact does exist an algorithm (i.e., a computer program) that can answer all questions of this sort in a finite amount of time. If not, then the system is *undecidable*. It is not our intention in this book to go into any detail about the mathematical notion of decidability, but a few comments on its relevance for modeling are in order.

The first thing to understand about decidability is also the most surprising – how easy it is for a formal system to be undecidable. Given the formal nature of logic, it might seem that with enough patience and engineering, that a program could be developed to correctly and completely process any formal logic. One of the most influential theorems that established the importance of the notion of decidability shows that even very simple logical systems (basically, any system that can do ordinary integer arithmetic) are undecidable. It is actually quite challenging to come up with a logical system that can represent anything useful that is also decidable.

This bit of tightrope walking is the impetus behind the OWL-DL standard. OWL-DL is based on a particular formulation of Description Logic. This means that there is an algorithm that can take as input any model expressed in OWL-DL, and determine which classes are equivalent to other classes, which classes are subclasses of other classes, and which individuals are members of which classes. The most commonly used algorithm for this problem is called the *Tableau Algorithm*. It works basically by keeping track of all the possible relations between classes, ruling out those that are inconsistent with the logical statements made in the model. The Tableau Algorithm is guaranteed to find all entailments of a model in OWL-DL in a finite (but possibly quite long!) time.

Furthermore, it is possible to determine automatically whether a model is in fact in OWL-DL, so that a program can even signal when the guarantees cannot be met.

Modeling in OWL-DL supports the intuition that a model must be clear, unambiguous, and machine-processable. The Tableau Algorithm provides the machinery by which a computer system can make determinations about equivalence of classes.

15.1.2 Executable models

Another important motivation for formal modeling in the semantic web is to form a complete, integrated picture by federating information from multiple sources. If one source provides information about the places where hotel chains have hotels, and another describes what hotels appear at a particular place, a formal model can tell us that we can merge these two sources together by treating them as inverses of one another. The model provides a recipe for adding new information to incomplete information, so that it can be federated with other sources.

Seen from this point of view, a model is similar to a program. It provides a concise description of how data can be transformed for use in other situations. What is the impact of decidability in such a situation? Standard programming languages like Fortran and Java are undecidable in this sense. The undecidability of these languages is often demonstration with reference to the *Halting Problem*; it is impossible in principle to write a FORTRAN program that can take another arbitrary FORTRAN program as input, along with input for *that* program, and determine whether *that* program will halt on that input. Even though these languages are undecidable, they have proven nevertheless to be useful engineering languages. How can we write programs in these languages, if we can't automatically determine their correctness, or in some sense, even their meaning? The

answer to this question in these cases is a discipline called Software Engineering. Even though it is not possible *in general* to determine whether any program will terminate, it is usually possible to determine that *some particular* program will terminate, and indeed, with what answer. The craft of engineering good FORTRAN programs is to write programs that not only will terminate on all input, but will actually perform well on particularly interesting input.

Seen from this point of view, decidability is not a primary concern. Models are engineered in much the same way as programs are. If a model behaves poorly in some situation, then an engineer debugs the model until it behaves well. Since we are not concerned with decidability, we don't need the guarantee that any algorithm will find all possible inferences. This opens up the choice of processor for OWL to a much wider range of algorithms, including algorithms like Forgy's RETE algorithm that have enjoyed considerable popularity as processors for rule-based languages.

This executable style of modeling is the motivation behind the OWL-Full standard. The meaning of a modeling construct in OWL-Full is given in much the same way as the meaning of a construct in a programming language is given. Just as the meaning of a statement in a procedural programming language is given by the operation(s) that a machine will carry out when executing that statement, the meaning of an executable model is given by the operation(s) that a program (i.e., an inference engine) carries out when processing the model. Information federation is accomplished because the model describes how information can be transformed into a uniform structure.

15.1.3 OWL-Full vs. OWL-DL

So far, we have described the motivation behind OWL-Full and OWL-DL without actually describing what the differences are in terms of the actual language.

The first thing to note about OWL-DL and OWL-Full is that they use exactly the same constructs; every modeling construct you have learned in this book can be used both in OWL-Full and OWL-DL. The inferences that you can draw from them are also the same, with the understanding that in the case of OWL-Full it might not be possible for an automated system to draw all correct conclusions.

The difference between the languages lies in the usage. But describing these differences is also problematic, since the determination of the precise boundary between OWL-Full and OWL-DL is a popular topic for Description Logic researchers. Many of the restrictions that were originally defined for OWL-DL have since been proved to have been too harsh. Inclusion of these usages has been shown not to damage the decidability of the model. For this reason, it is more important to understand the decidability-based motivation of the distinction than any particular usage distinction. Here we will outline the major kinds of restrictions on the modeling language that are enforced by OWL-DL. The details of these continue to change as research in description logic proceeds.

Class/Individual separation. In OWL-DL, Classes and Individuals are completely separate. That is, a model cannot specify that some resource is both a class and a member of a class. Recalling an example from **Error! Reference source not found.**, we defined a number of ranks as classes:

```
ship:Captain rdfs:subClassOf ship:Officer .
ship:Commander rdfs:subClassOf ship:Officer .
ship:LieutenantCommander rdfs:subClassOf ship:Officer .
ship:Lieutenant rdfs:subClassOf ship:Officer .
ship:Ensign rdfs:subClassOf ship:Officer .
```

We can specify the rank of an individual using membership in one of these classes, e.g.,

```
:Warwick rdf:type ship:Captain .
```

By virtue of their use in *rdfs:subClassOf* triples, all of the entities mentioned here are classes. But in another context, we might want to express what we know about these ranks. For instance, there is an ordering to these ranks, by which *Captain* outranks *Commander*, which in turn outranks *LieutenantCommander*, etc. We could express this relationship in RDF using a series of triples:

```
ship:Captain ship:outranks ship:Commander .
ship:Commander ship:outranks ship:LieutenantCommander .
ship:LieutenantCommander ship:outranks ship:Lieutenant
.
ship:Lieutenant ship:outranks ship:Ensign .
```

We represent the usage of *ship:outranks* with domain and range specifications as well:

```
ship:outranks rdfs:domain ship:Rank .
ship:outranks rdfs:range ship:Rank .
```

While this seems like a natural thing to do, it violates the separation of Class and Individual in OWL-DL. Each rank is both a class (with members who hold that rank). But the domain and range information of *ship:outranks* makes each rank a member of the class Rank, and hence they are individuals. In OWL-Full, there is no condition forbidding this usage.

Property/Individual separation. In a similar vein, a property cannot also be an individual. We'll consider a hypothetical example of metadata management to illustrate this. Suppose we use a property to represent ownership metadata about information artifacts.

```
comp:creator rdfs:domain comp:Asset .
```

That is, each company asset has a creator. In this scenario, when the property *comp:creator* was introduced, the idea was that company assets would include things like documents, web pages, books, etc., and that this information would be described using the property *comp:creator*.

Suppose that someone then were to decide to take this step one further, and use *comp:creator* to describe parts of the model. It would be tempting to use it to describe the creator of all the parts of the model, thus:

```
ship:Captain comp:creator :Wenger .
ship:outranks comp:creator :Polk .
```

This says that *Wenger* is the creator of the class *ship:Captain*, and that *Polk* is the creator of the property *ship:outranks*. This information about the source of information could be very useful during model maintenance. Who do I go to, to find out about the design decision in the *Captain* class? Or the intended usage of the *outranks* property?

This is a perfectly sensible usage of a property like *comp:creator*, but it violates the rules of composition in OWL-DL, since the *domain* information about *comp:creator* allows us to infer that *ship:Captain* and *ship:outranks* are members of the class *Asset*, and hence individuals. Early versions of SKOS (**Error! Reference source not found.**) had this error. SKOS used certain annotation properties to describe SKOS entities, thereby making all SKOS properties and classes become individuals. Not only was this

confusing for people evaluating the inferences from a SKOS model, it also prevented any model that includes SKOS from satisfying the conditions to be in OWL-DL.

InverseFunctional on datatypes. In **Error! Reference source not found.**, we learned that *owl:InverseFunctionalProperty* is an important construct for data federation. Whenever two individuals share a value for an *InverseFunctionalProperty*, we can infer that they are the same individual. Things like social security number, employee number, driver's license number, serial number, etc. are commonly used in this way. In **Error! Reference source not found.** we saw that FOAF uses *foaf:email* in this way.

Unfortunately, OWL-DL has a condition that outlaws exactly these uses. It stipulates that an *InverseFunctionalProperty* must not also be a *DatatypeProperty*, that is, it cannot refer to a string, date, number, etc. That is, exactly the things that make up social security numbers, email addresses, etc. are forbidden from *InverseFunctionalProperties*. This is a stringent restriction, and one that is quite often responsible for placing a model into OWL-Full instead of OWL-DL.

15.1.4 OWL-Full together with OWL-DL

The distinction between OWL-Full and OWL-DL is technical, having to do with advanced mathematical topics like decidability. Add to that the fact that the utility of a decidable language is also controversial, and the result is that many beginning modelers find the prospect of deciding between them to be quite daunting. How do I decide between OWL-DL and OWL-Full? Do I have to decide now? What are the ramifications of making the wrong decision? Do I have to understand advanced mathematics to make this decision? If this were the case, then there would be a serious barrier to entry for semantic modeling.

Fortunately, the data merging mechanism of RDF provides a means by which a modeler can follow good engineering practice and postpone this decision for as long as possible. Since OWL-DL is a subset of OWL-Full, any model (or model fragment) built in OWL-DL will also be a valid OWL-Full model.

In **Error! Reference source not found.**, we saw how to use *owl:imports* to include one model into another. The interpretation of such an import is that the importing model includes all triples from both models.

Just because one model imports another, doesn't mean that both of them have to be in the same dialect of OWL. In particular, an OWL-Full model can import an OWL-DL model. Let's suppose that *model:A* is an OWL-Full model, and *model:B* is an OWL-DL model, and that *A* imports *B* thus:

```
model:A owl:imports model:B .
```

Since *A* imports *B*, all the triples from *B* can also be considered to be in *A*. But since OWL-DL is a subset of OWL-Full, no new constructs were added into the model by the import. The import does not change the status of *A*; it is still OWL-Full.

On the other hand, *B* does not import anything, so no foreign triples are included in it. So *B* is still in OWL-DL.

For purposes of merging with other models, advertising as a reusable ontology, or being able to make comprehensive decidable proofs, *B* has all the advantages of an OWL-DL model. For the purposes of data manipulation and treating the model as a program, A has all the information that *B* has.

When starting a model from scratch, it isn't necessary to decide whether to model in OWL-DL or OWL-Full. In any case, best practice suggests keeping the model in (at

least) two separate pieces, where one imports the other. The imported model can be kept in OWL-DL. If it is important to you or your audience that some aspect of your model be published in OWL-DL, then that aspect should be modeled in the imported ontology. Aspects that are of use to an OWL-Full application should be modeled in the importing ontology.

This practice was used in FEARMO (**Error! Reference source not found.**); the published FEARMO models are in OWL-DL, but certain FEARMO applications make use of OWL-Full features. These models import the published OWL-DL FEARMO models, which contain all the reusable content. Any other party who wishes to include FEARMO into their model can import the OWL-DL version.

15.1.5 OWL-Lite

Along with OWL-Full and OWL-DL, the original OWL specification identified a subset of OWL-DL with the intention that it would be easier to implement, and speed adoption of OWL. As OWL implementations mature, the significance of OWL-Lite is fading. Many implementations have skipped over OWL-Lite entirely, and gone directly into support of OWL-Full or OWL-DL, or, more commonly, supporting a proprietary subset of OWL. The simplifications in OWL-Lite include:

Limited Cardinality Restrictions. Cardinality restrictions are limited in OWL-Lite to the integers 0 and 1. But as we have seen in **Error! Reference source not found.**, cardinality restrictions to 0 or 1 have natural and common interpretations. Most cardinality restrictions in real models use 0 or 1 anyway.

No oneOf constructs. OWL-Lite does not include *owl:oneOf* constructs. This is in line with a simplified model of cardinality.

No hasValue restrictions. OWL-Lite does not include *owl:hasValue* restrictions.

15.2 Beyond OWL

Throughout this book, we have seen examples of the usefulness of the constructs of OWL for information management and data integration on a web-wide scale. But we have not argued for their completeness – that is, are there any other modeling constructs that would be useful in a Semantic Web setting?. In fact, there are a number of useful modeling capabilities that go beyond the capabilities of OWL. The Semantic Web, like the Web itself, is an ever-developing system, with new components and capabilities being developed all the time. In this chapter, we explore some of the directions in which the Semantic Web is being developed, either within the OWL recommendation itself, or beyond it, typically in proposals for rule languages for the web.

15.2.1 Metamodeling

"Metamodeling" is the name commonly given to the practice of using a model to describe another model as an instance. One feature of metamodeling is that it must be possible to describe properties of classes in the model. But as we have seen above, describing classes typically violates the separation of class and individual that allows a model to be described in OWL-DL.

There are a number of motivations for metamodeling. One such motivation is that a model needs to play more than one role in an application. In one role, a particular concept should be viewed as a class; in another role, as an instance. If we are modeling animals, we might say that *BaldEagle* is an endangered species, thereby referencing *BaldEagle* as an individual. But in another application, we could view *BaldEagle* as a class, whose

members are the particular eagles in the zoo. Examples of this sort abound; wine connoisseurs speak of individual wines in terms of vintage. For them, the vintage is an individual. But for a wine merchant who is keeping stock of how many bottles the shop has sold, the bottles themselves are individual members of the class, which is indicated by the vintage.

We have already seen a number of examples of this kind of metamodeling in this book. In **Error! Reference source not found.**, we saw how *a foaf:Group* is an individual that corresponds to a class of all the members of the group. In **Error! Reference source not found.**, we saw how the Class-Individual Mirror pattern allowed us to view a line of business either as an individual, or as a class of all the subfunctions that comprise it. In 15, we saw how military ranks can be seen as both classes and individuals.

Other reasons for metamodeling are to imitate capabilities of other modeling systems (like object-oriented modeling) in which the value for some property can be specified for all members of a class at once.

Metamodeling itself is not an issue in OWL-Full, since there is no restriction against using the same resource as an individual and as a class. The formal issues really arise only when trying to achieve the results of metamodeling in OWL-DL. But even though there is no formal issue with overloading a single resource to refer to a class and an individual, we recommend as a best practice to keep these things separate, even in OWL-Full. There really is a difference between a species and the set of animals of that species; there is a difference between Shakespeare's family and the set of people in it. These distinctions could be important to someone who wants to reuse a model. Keeping them distinct in the first place will enhance the model's utility.

Fortunately, there are a number of possible approaches to doing metamodeling in OWL (either OWL-DL or OWL-Full). For most situations, we recommend the Relationship Transfer pattern from **Error! Reference source not found.**, or the Class-Individual Mirror pattern from **Error! Reference source not found.**.

Recent Description Logic research has determined that in certain cases, the Class/Individual separation constraint can be relaxed without any danger to the decidability of the logic. Thus it is possible to have a new version of OWL-DL in which metamodeling of the sort we have described here can be done as easily in OWL-DL as in OWL-Full. Whether such a proposal reaches fruition in the OWL standard or not, we still recommend using one of the patterns in this book whenever possible, instead of resorting to overloading resource usage.

15.2.2 Multipart Properties

In RDFS, we have seen how properties can relate to one another using *rdfs:subPropertyOf*. This establishes a hierarchy of properties; any relations that hold lower in the hierarchy also hold higher in the hierarchy. There are other ways in which properties can relate to one another. A common example is the notion of *uncle* – A is the *uncle* of B just if A is the *brother* of someone who is the *parent* of B. This is called a "multipart property" – that is, the property *uncle* is made up of two parts (in order): *parent* and *brother*.

When multipart predicates are used with other RDFS and OWL constructs, they provide some powerful modeling facilities. For instance, we can model the constraint "a child should have the same species as its parent" by stating that the multipart predicate

made up of *hasParent* followed by *hasSpecies* (which we denote as *hasParent* + *hasSpecies*) is *rdfs:subPropertyOf hasSpecies*. Let's watch how this work.

Suppose we have the following triples

```
Elsie hasParent Lulu .
Lulu hasSpecies Cow .
```

Now we can infer

```
Elsie hasParent+hasSpecies Cow .
```

But since the multipart predicate *hasParent+hasSpecies* is *a rdfs:subPropertyOf hasSpecies*, we can infer

```
Elsie hasSpecies Cow .
```

There are some proposals for how to represent multipart properties in OWL, in such a way that they do not endanger the decidability of OWL-DL.

15.2.3 Qualified Cardinality

Cardinality restrictions in OWL allow us to say how many distinct values a property can have for any given subject. Other restrictions tell us about the classes that those values can or must be members of. But these restrictions work independently of one another; we cannot say how many values from a particular class a particular subject can have. A simple example of qualified cardinality is a model of a hand. A hand has five fingers, one of which is a thumb.

Qualified cardinalities may seem like a needless modeling detail, and in fact, a large number of models get by quite fine without them. But models that want to take advantage of detailed cardinality information often find themselves in need of such detailed modeling. This happens especially when modeling the structure of complex objects.

For example, when modeling an automobile, it might be useful to say that a properly equipped automobile includes five tires, four of which must be regular road-worthy tires, and a fifth that is a designated spare tire, that might not have all the properties of a regular tire. Structural models of this sort often make extensive use of qualified cardinalities.

There is a proposal for defining qualified cardinalities that will allow them to work within the decidability constraints of OWL-DL.

15.2.4 Multiple Inverse Functional Properties

Inverse Functional Properties can be used to determine the identity of individuals based upon the values of the properties that describe them. If two people share the same social security number, then we can infer that they are actually the same person. This kind of unique identifier is indispensable when merging information from multiple sources.

Unfortunately, anyone who has done a lot of such integration knows that this kind of merging only scrapes the surface of what needs to be done. Far more common is the situation in which some combination of properties together implies the identity of two or more individuals. For instance, two people residing at the same residence with the same first and last names should be considered to be the same person. Two people born in the same hospital at the same time of day should be considered to be the same people. Examples of this kind of multiple identifiers are much easier to come by than single identifies, as required by inverse functional properties.

To further complicate matters, in real information federation situations, it is often the case that even these combinations of properties cannot guarantee the identity of the

individuals. Two people at the same address with the same name are very likely to be the same person (but not certain; a father could live with his son of the same name). OWL has no facility to deal with uncertainty, so there is no way to express this sort of information. Extending OWL to deal with uncertainty is a topic of current research and standardization efforts in the Semantic Web.

A few proposals have been made for how to deal with multiple inverse functional properties in OWL. The problems include syntactic ones (how to express a relation including an arbitrary number of properties) as well as logical ones (what are the logical properties of the resulting system?). It is even possible to achieve multiple inverse functional property definitions while staying within the inferencing bounds of OWL-Full. The solution is far too convoluted for clear and understandable semantic models.

15.2.5 Rules

OWL is the most inclusive inferencing system currently defined for the Semantic Web, but it isn't likely to be the last. Many of the limitations in OWL can be addressed, for the purposes of data management, using Rules.

Rule-based systems have a venerable tradition starting in the days of Expert Systems, and are in common use in business logic applications to this day. A number of useful algorithms for processing data with rules have been known for many years, and many of them have been streamlined.

Many of the issues with OWL presented in this chapter can be addressed with rules. Multipart properties (like the definition of *uncle*) are easily expressed in rules. Multiple inverse functional properties can be expressed in rules as well. There are even a number of approaches to uncertainty in rules. While none of them has emerged as the final word

in rule-based uncertainty, many of them have considerable research and practical examples behind them, making uncertainty in rules a relatively well-understood issue.

Given all these virtues of rules and rule-based systems, why don't they play a bigger role in modeling on the Semantic Web than they do? In fact, one could even ask why there is a need for a modeling language like OWL, when there exists a mature, well-understood rules technology. One could even ask this question in greater generality. Why aren't more software systems in general written in rules?

We cannot treat this issue in full detail in this book, but we can outline the answer, as it relates to OWL and the Semantic Web. One of the lessons learned from the history of rule-based systems is that software engineering in such systems is more difficult than it is in modular, procedural languages. While it is unclear whether this is an essential feature of rule-based systems or not, it is undeniable that rule-based programmers have not achieved the levels of productivity of their more conventional counterparts. This has particular ramifications in the Semantic Web; one defense for using OWL-Full vs. OWL-DL was that software engineering discipline makes the notion of decidability basically irrelevant for model design. In the case of rule-based systems, software engineering cannot provide this same support. But rule-based systems are just as undecidable as general-purpose languages like FORTRAN and Java.

Is there a way to get the best of both worlds? In section 15.1.4, we saw how we could combine OWL-DL and OWL-Full models, and in some sense get the advantages of both modeling languages. Can this be done with rules as well? These, and other similar questions, are the current focus of research and development into Semantic Web rules languages.

15.3 Chapter Summary

OWL should be considered as a living language, growing in the context of the ways it is being used on the web and in commerce. As shortcomings in the language are identified, the system grows to accommodate. Sometimes that growth takes the form of additional constructs in the language (e.g., multipart properties), sometimes as connections to other systems (rules), but sometimes progress in a language comes from specifying limitations to the language (as is the case for OWL-DL and OWL-Full). All of these processes are moving in parallel for the Semantic Web.

15.4 Fundamental Concepts

OWL-Full – unrestricted dialect of OWL, with all constructs used in any combination.

OWL-DL – dialect of OWL restricted to ensure decidability – all constructs allowed, but certain restrictions on their use.

OWL-Lite – subset of OWL-DL designed to encourage early adoption. Significance wanes as implementations reach OWL-DL and OWL-Full levels.

Metamodeling – models that describe models, usually requires that classes be treated as individuals.

Multipart properties – daisy-chain composition of properties.

Multiple Inverse Functional Properties – uniquely identify an individual based upon matching values for several properties.

Qualified Cardinality – cardinality restriction whereby the class of the value being counted is specified as well as the number of distinct values.