Frames and OWL Side by Side

Hai H. Wang[1] Natasha Noy[2] Alan Rector[1] Mark Musen[2] Timothy Redmond[2] Daniel Rubin[2]
Samson Tu[2] Tania Tudorache[2] Nick Drummond[1] Matthew Horridge[1] Julian Seidenberg[1]

[1]Bio-Health Informatics Group
The University of Manchester,

[2]Stanford Medical Informatics
Stanford University

1 Introduction

Understanding the relation of the two most widely-used ontology modeling paradigms, OWL (and other DL paradigm) to frame paradigms is an important issue. Because the Protégé tool supports both paradigms within a single overall framework, it provides an opportunity to compare the two paradigms both in theory and practice. In this paper, we focus on the relationship between the Frames paradigm, as defined in the OKBC specification [1] and implemented in core Protégé, and OWL and its implementation in Protégé-OWL.

OWL provides three increasingly expressive sublanguages, OWL Lite, OWL DL, and OWL Full, for different communities of users. In this paper, we will primarily use OWL DL as the basis for comparison, as OWL DL provides the description-logic reasoning capabilities that are the distinctive features of OWL. Therefore, unless otherwise noted, when we use "OWL", we are actually referring to OWL DL. Contrastingly, there is not a single standard frame representation language analogous to OWL. The past standardization effort has produced the Open Knowledge-Base Connectivity (OKBC) protocol, which is a generic access and manipulation layer for knowledge representation services. For the purpose of this paper, we will primarily use the Protégé-Frames implementation of the OKBC Knowledge Model. When appropriate, we comment on features of frame representations that may not be available in Protégé. Unless otherwise noted, when we use "Frames," we are actually referring to Protégé-Frames.

2 Similarities and Differences

The two paradigms have many similar modeling constructs: both are built around the notion of classes, representing concepts in the domain of discourse; classes have instances; properties (slots) describe attributes of those classes and relationships between them; restrictions and facets express constraints on the values of properties and slots. There are however major differences in the semantics of these constructs and in the way these constructs are used to infer new facts in the ontology or to determine if the ontology is consistent. As a result, the way that the modeling constructs are used in the two paradigms and the implications of definitions are different.

2.1 Semantics and implication

The following represents some of the major differences between Frames and OWL:

Unique name assumption In Frames, if two objects have different names, they are assumed to be different, unless explicitly stated otherwise. In OWL, no such assumption is made.

Closed World Assumption vs Open World Assumption In Frames, everything is prohibited until it is permitted; in OWL, everything is permitted until it is prohibited. Nothing can be entered into a Frames KB until there is a place for it in the corresponding template. Anything can be entered into an OWL KB unless it violates one of the constraints.

Single vs Multiple models A Frames ontology only has one model which is the minimal model that satisfies each of the assertions of the Frames ontology. This means that models for a Frames ontology can only contain instances that are explicitly specified. In general an OWL ontology will have many models consisting of all possible interpretations that satisfy each of the assertions in the OWL ontology.

These differences have direct implications on how inference is performed, and therefore, implications for the use of modeling constructs in the two paradigms. The foremost of these differences are:

[1]http://www.ai.sri.com/okbc/spec.html

Assertion vs Classification In Frames, defining facets on a slot at a class, or defining a constraint on a slot at the top level, makes a statement about all instances of that class (except for possible exceptions provided by default values), describing *necessary* conditions for instances of that class. In OWL, there are effectively two kinds of statements about classes: a) those that, as in Frames are true of all individuals in a class, and b) those that are collectively *necessary and sufficient* to recognize members of a class. A OWL classifier can use the sufficient conditions to infer which classes are subclasses of the defined class. There is no equivalent feature in Frames.

Constraint checking vs Consistency checking The same reasoner that checks the classification also checks that an OWL KB is consistent. The classifier tries to build a model that satisfies all the axioms in the ontology. If no such model can be built, the ontology is inconsistent. When building a model that satisfies all the assertions, a classifier may assign new types to ontology instances, in addition to the types explicitly asserted by a modeler. By contrast, a Frames reasoner checks if the constraints are satified by the property values on instances; if they are not, the instance is non-conformant. In Frames, the inference cannot assign a new type to an instance.

Other differences include association of slots and properties to classes and individuals: In OWL, properties can be used with any class or individual unless such use violates explicitly specified constraints. In Frames, slots must be explicitly attached to classes before we can add facets to them or use them to assign slot values for the instance of the class. Treatment of multiple domains and ranges for slots and properties also works differently: In OWL, multiple domains and ranges are treated as an intersection, and in Frames—as a union.

In practice, these major differences in the sanctioned inference lead to differences in modeling style. A developer of an OWL ontology thinks in terms of necessary and sufficient conditions to define a class, building new concepts from existing ones by fitting them together in definitions like blocks of Lego and determining what conditions sufficiently define something as an instance of a class. A developer of a Frames ontology addresses the problem from another angle, deciding what are the implications of being a member of a particular class.

2.2 Expressive Power

2.2.1 Frames

Meta-modelling: Metaclasses are classes whose instances are themselves classes and constitute an integral part of the Frames formalism. In OWL DL, the sets of classes and instances are disjoint, and therefore metaclasses are not available[2].

Classes as property values: Another implication of the requirement that classes and instances are disjoint in OWL DL is the the prohibition on using classes as property values. There are a number of ways to approximate this meaning in OWL DL , using classes directly as property values could be much more intuitive in many cases.

Default Information and Exceptions: Default reasoning has always been one of the great strengths of frames systems. Frames are designed around the notion of "prototypes" in which defaults are used to fill partial knowledge in our perceptions. Many domains, such as Biology and Medicine, strongly rely on exceptions. Because OWL is a logic formalism. It leaves no room for exceptions. Currently, there are some attempts to use design patterns to represent default information in OWL . However, the proposed approaches are far from perfect and impractical in all but the simplest cases. Default reasoning remains one of the key strengths of Frames over OWL.

Concrete Domains and User Defined Datatypes: One of the omissions in the current OWL language that prevents many potential users from adopting OWL is poor representation of numeric expressions in order to be able to express quantitative relations. Issues preventing this support from being in the current version of OWL include how to refer to an XML schema user defined datatype with a URI and the denotational semantics of the XML data type etc. However, these are being resolved and OWL 1.1 will support user-defined datatypes.

2.2.2 OWL DL

Defined Classes: As discussed before, OWL allows us to 'define' classes, allowing inferred subclass and type information to be calculated. There is no equivalent to defined classes in Frames.

Embedding Class Definition: OWL allows complex expressions to be built up without having to name each intermediate concept before use. This embedding of 'anonymous concepts' allows more expressive representations to be built easily and naturally. To some extents, the embedding of concepts in expressions can be considered as a special case of definition. This can result in a dramatic reduction in the number of facts that have to be maintained explicitly. Frames does not have a notion of anonymous classes.

[2]Metaclasses are allowed in OWL Full.

Set Combination on Classes: OWL allows users to apply the standard set operators (Intersection, Union and Complement) on class descriptions. With the combination of the OWL class axioms, like equivalent class, we can model statements like covering, disjointness etc.

Characters of Properties: Frames allows uses to say that a slot is *functional* (by setting the maximum cardinality to be one) or *symmetric* (by setting the invert slot to be itself). In OWL, apart from *functional* and *symmetric*, the properties can be set as *transitive* as well. Statements like "If A is part of B and B is part of C, then A is part of C as well" which could not be modeled in Frames could be model in OWL. The transitive property is important for some application domains like anatomy and geography.

3 Tool Support

3.1 OWL DL

3.1.1 OWL reasoner

As we have discussed, one of the most important distinguishing features of logic based ontology languages like OWL from other ontology formalisms is that it has formal semantics. Staying within OWL DL, allows us to build reasoners which can make automatic inferences over our knowledge base. Reasoners can be used at development time to help users to build and manage their ontologies more easily. We believe that building large, reusable ontologies with rich multiple inheritance by manual effort unaided by formal checks is virtually impossible. This is doubly so in any collaborative environment where work from several authors must be reconciled and integrated. However, many of our users develop draft versions of ontologies without benefit of classifiers to which we propose "enrichments" with the help of classifiers and then return to the original authors for checking. At the end of the development process, we deliver most of the ontologies we build to applications in forms that do not require the applications to have classifiers or even know that classifiers have been used. In general, the use of OWL reasoners includes but is not limited to the following:

Diagnosis An OWL ontology is an engineered artifact which may inevitably contain flaws which may include logical inconsistency, unexpected subsumptions inference and unexpected type coercion for individuals. Using OWL reasoners directly the first kind of flaws can be automatically and efficiently identified. With tool support and additional interaction from users, the causes of the last two flaws can normally be discovered as well.

Composition OWL allows us to build more complex concepts out of simpler concepts – this is sometimes referred to as "Conceptual Lego". The fundamental advantage of using classifiers is that they allow composition i.e. they allow new concepts to be defined systematically from existing concepts. This can result in a dramatic reduction in the number of facts that have to be maintained explicitly.

Normalisation and Managing Polyhierarchies Managing and maintaining a complex polyhierarchy of concepts is hard. Changes often need to be made in several different places, and keeping everything in step is error prone and laborious. Using a classifier, we have developed a mechanism for 'normalisation' of ontologies. In normalised ontologies, even the most complex polyhierarchies are built out of simple non-overlapping trees. Concepts bridging the trees are created by definitions. The relationships amongst defined concepts are maintained by the classifier. Changes are always made in exactly one place.

Pre- and Post- Coordination and the Ontology Life Cycle Querying a knowledge base is one of the fundamental tasks needed in an ontology-driven application. Depending on the type of ontology and the type of queries that need to be asked, we can determine whether the reasoner is needed at run-time, or if it can be simply used at "publish" time. Many applications need an ontology primarily as a fixed vocabulary with fixed relations. They do not expect to define new concepts in terms of existing concepts "on the fly" at run time. What is required at run time is a "pre-coordinated" ontology in which all concepts needed and the relations between them are explicitly pre-defined. That the pre-coordinated ontology was, or was not, derived through normalisation and classification is irrelevant at run time. What is required is a hierarchy (or polyhierarchy) of concepts to "fill the boxes" in the information or decision support model. In this case the life cycle of the ontology is that it is authored, maintained and demonstrated correct in OWL with the help of the classifier, but then "published" as a pre-coordinated form in RDF(S), OWL Lite, CLIPS, or some proprietary form. On the other hand, some applications require more concepts than can be conveniently enumerated. Indeed the number of concepts that can be defined from the elements of an ontology may be indefinitely large or even demonstrably infinite. If it cannot be predicted in advance which concepts are needed, then the classifier is needed at run time to create and classify the concepts found.

3.2 Frames

3.2.1 Knowledge-Acquisition Forms

Class definitions in Protégé-Frames provide constraints on instances of those classes. Tools like Protégé-Frames use these constraints to guide and verify the knowledge-acquistion process. Protégé-Frames, for example, generates knowledge-acquisition forms based on Slot widgets in Protégé-Frames (the user- interface components for acquiring slot values) limit the number of choices that users get and flag the violation of constraints: for instance, Protégé-Frames will put a red border around a slot with more values than the maximum cardinality constraints allows. We can simulate similar knowledge-acquisition support for OWL, but this does not reflect the semantics of the language.

3.2.2 Constraint languages

Most frame-based systems lack the ability to express disjunction, existential quantification about frames, or relationships between properties of the same frame or different frames. For example, one cannot restrict a value of one property based on that of another property. A remedy for this limitation is the introduction of a more expressive constraint language based on First Order Logic (FOL) developed specifically to express these more complex relationships. In Protégé the Protégé Axiom Language (PAL) serves this purpose. PAL is a subset of first-order logic that allows users to express constraints on slot values while making the unique-name and closed-world assumptions.A plugin to Protégé enables users to find the instances in the knowledge base that violate the PAL constraints.

3.2.3 Query tab

Protégé provides different ways in which the user can query the content of an ontology. The **QueryTab** is part of the Protégé system and allows the user to retrieve instances in the ontology that match certain criteria. The queries are defined following the pattern *(class, slot, operator, value)* and can be composed in a conjunction or a disjunction of queries. The operator can be set according to the type of values that the slot may take. For instance, a slot of type *String* allows besides the *equals*, other lexical operators as well, like *contains* or *begins with*. The query engine is built-in Protégé-Frames and uses the closed world assumption, i.e. it will only return as true the things that have been explicitly asserted in the knowledge base. A query like, "Give me all instances of *Pizza* that have the value of *hasCountryOfOrigin* slot *Italy* " will only return the instances of pizzas that have this assertion explicitly made.

4 Frames or OWL: Guidelines

The many differences between frames and OWL could prove daunting to users who need to select an approach for their applications. As might be expected, in some cases, the frame representation with its closed-world semantics is a good fit for the application, while in other cases, the capabilities and expressiveness of OWL are needed to deliver the functional requirements.

In general Frames is particularly useful when the application has the following requirements:

- Creating ontologies for domains in which the closed-world assumption is appropriate.
- The application focuses on data acquisition.
- The application domain requires constraints on slot values.
- The model relates classes to other classes.

Likewise, the following requirements of applications may make OWL a preferred choice:

- Creating robust terminologies in which classes are defined.
- Need for DL reasoning to ensure logical consistency of ontologies.
- Controlled terminologies are published on the Semantic Web and accessed by other applications.
- Applications in which classification is a paradigm for reasoning.

5 Conclusion

In this paper, we compared the two most important ontology modeling paradigms – Frames and DL both in theory and practice. We hope that from this comparison users can understand the major difference between these two paradigms and be able to choose the most suitable modeling languages for their applications.