

What's in an Instance?

Christopher A. Welty and David A. Ferrucci

RPI Computer Science Dept.
Troy, NY 12180
{weltyc,ferrucci}@cs.rpi.edu

The notion of an instance is ubiquitous in knowledge representations for domain modeling. Most languages used for domain modeling offer syntactic or semantic restrictions on specific language constructs that distinguish individuals and classes in the application domain. The use, however, of instances and classes to represent domain entities has been driven by concerns that range from the strictly practical (e.g. the exploitation of inheritance) to the vaguely philosophical (e.g. intuitive notions of intension and extension). We demonstrate the importance of establishing a clear ontological distinction between instances and classes, and then show modeling scenarios where a single object may best be viewed as a class and an instance. To avoid ambiguous interpretations of such objects, it is necessary to introduce separate universes of discourse in which the same object exists in different forms. We show that a limited facility to support this notion exists in modeling languages like Smalltalk and CLOS, and argue that a more general facility should be made explicit in modeling languages.

1 Introduction

The term *instance* is familiar to members of the Knowledge Representation, Logic, and Object Oriented Communities. The interpretation of representing an object as an instance has always been an implicit assertion that the object *exists* as an individual [Brachman77]. Instances are typically distinguished from *classes*, which are interpreted as descriptions of sets of individuals.

We have found that the distinction between instances and classes is frequently blurred, and this gives rise to undesirable ambiguities and misinterpretations of represented knowledge. A more disciplined approach to the use and interpretation of instance and class is required to eliminate these problems.

We begin by formally stating the restriction that instances and classes must be kept distinct. We discuss the importance of this restriction and show that problems do arise when it is violated. We further discuss a common inference used

when modeling with instances and taxonomic relationships, and show that it is important to modeling domains. We then consider objects whose intuitive conceptualization requires that they be represented as a class *and* an instance. In conventional modeling systems, this requirement seems to call for the dismissal of either our important restriction or our important inference. We show how these objects can be represented without causing such problems by introducing a construct called the *spanning object*. We conclude with previous intuitions of spanning objects and a simple example illustrating their use in an existing domain modeling effort.

2 The Instance Restriction

All modeling languages have a *syntax* and a *semantics*, and these are each subdivided into *pure* and *descriptive* parts [Carnap61]. The descriptive semantics of a language defines how the constructs of the language are interpreted. Disregard for this descriptive semantics can lead

to misinterpretations of representations. In particular, when modeling languages have special constructs for denoting instances, those constructs must have a consistent interpretation within a domain. More formally, we define the *instance restriction*:

I1 If language construct X is used to denote individuals of a set in some universe of discourse Y , then X can not be used to denote a *set* of individuals of Y .

For each of the following categories of modeling languages we describe the language constructs for instances and show that a violation of **I1** is either not permitted syntactically or causes problems in interpretation.

2.1 First-Order Logic

In FOL the language constructs in question are *predicate symbols* and *object symbols*. According to the descriptive semantics of symbolic logic, object symbols may range only over the individuals of the universe of discourse and predicate symbols range over sets of individuals (and other predicates) [Carnap61]. For example, in the universe of discourse Y_1 , $\text{horse}(\text{secretariat})$ is a valid FOL expression in which the predicate symbol horse is interpreted as a set (or class) name, and the object symbol secretariat refers to an individual. Predicate symbols can not themselves have predicates. For example, in Y_1 the expression $\text{hooved}(\text{horse})$ is invalid, as predicates of predicates are, by definition, second-order. The importance of maintaining a first-order representation lies in the absence of a complete deduction system for higher order logics [Gödel31].

There is no *syntactic* restriction in FOL that prevents the formation of expressions that violate **I1**, such as $\text{instance}(\text{secretariat}, \text{horse})$. It is important to note that if the expression is meant as an alternative to $\text{horse}(\text{secretariat})$, it's *interpretation* in Y_1 is second-order; set (or class) membership in symbolic logic is a predicate [Quine64], there-

fore this expression contains a predicate of a predicate. The problem with this kind of violation of **I1** in FOL is that the representation is inconsistent with the interpretation of the object symbol construct, which is supposed to denote only individuals in a universe of discourse.

The importance of maintaining consistent semantic interpretation of the language constructs is exemplified in domain modeling. The main goal of modeling a domain in FOL is to capture some knowledge about the domain in a form that others can understand. Part of that understanding by others is grounded in the interpretation of the basic language constructs. For example, a person seeking to use an FOL domain model may decide that, "all individuals in the domain have a location, so I will give each object symbol a location." This is a fairly intuitive statement that relies on the assumption that *all object symbols denote individuals*. It should be reasonable for a person to make that assumption, since it is part of the semantics of FOL. If the interpretation has been ignored and the class horse is represented as an object symbol, then it will have a location. It clearly should not have a location associated with it, however, since it does not exist as an individual in the same sense that secretariat does.

2.2 Object-Oriented Languages

In Object-Oriented languages the language construct *class* is distinct from the language construct *instance*. Instances are restricted to denoting individuals of the domain, while classes denote sets of individuals that satisfy some set of properties (classes are descriptions that characterize the necessary attributes of sets of individuals). **I1** is enforced by the syntax of an Object-Oriented language since it is not allowable to have an instance act as a description and itself have instances – an instance can not be used as a class.

2.3 Knowledge Representation Systems

An effort to classify all KR systems in use today exists [Patil92], and the purpose of this paper is neither to rival that effort nor exclude any particular system from consideration. We believe it is important to consider **I1** in the context of each KR system, and we specifically discuss two prominent *families* of systems: frame-based derivatives of KL-ONE and semantic networks.

2.3.1 Description Logics

In the KL-ONE family of languages (now called *description logics*), the language constructs *concept* and *individual* denote classes and individuals in the universe of discourse [Brachman85, Brachman89]. Concepts represent descriptions of classes of individuals, typically through the enumeration of the sufficient attributes of the individuals in that class. Description logics do not permit individuals to act as descriptions, and violations of **I1** are thus prevented by the syntax.

2.3.2 Semantic Networks

Semantic Network based representation systems generally have only two language constructs: *nodes* and *links* [Fox85, Maida82]. There is no construct for differentiating between classes and individuals, they are both represented as nodes, and thus **I1** need not apply in such systems. Semantic Network systems are a very low-level representation, however, which make no epistemological commitments [Shapiro87]. Using such systems to represent a domain basically requires defining a language (which may consist of no more than an ontology of special link and node types) on top of the semantic network in which the domain knowledge will be represented.

In most cases the layered language has a special link for an instance relationship. There is still nothing syntactic that prevents the use of chains

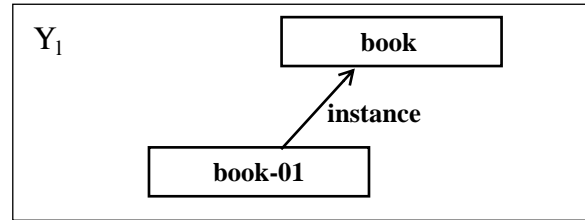


FIGURE 1. An instance relationship in a semantic net.

of instance links which would violate **I1** and give rise to ambiguities. In our experience, users of knowledge representation systems for domain modeling tend to view such unrestricted and rich facility as simply an implementation vehicle. The result is that interpretation of a construct (as e.g. an individual in the domain) may be disregarded in favor of some other attribute of a construct, such as *inheritance*. In Figure 1, for example, we consider a semantic network representation where an instance link between two nodes causes some slots and values to be copied from the node *book* to the node *book-01*. In this universe of discourse (a library card catalog), the modeler has chosen to make *book-01* an instance of *book*. At this point, the interpretation of the instance link is clear, it is an assertion that the domain node *book-01* exists as an individual.

The modeler then discovers a second copy of the same book in the library and, as shown in Figure 2, rather than re-type the information about that book, decides to use an instance link between this new copy, *book-02* and *book-01*, and lets the inheritance associated with the instance link do the work of copying the information into the new node. This may be a clever procedural trick to avoid doing extra work, but it is ontologically invalid; clearly *book-02* is not an instance of *book-01*, and ambiguities immediately arise due to the disregard for the proper interpretation. Consider a user of the model wanting to know how many books there are in the library by counting all the instances of *book*. The count will not include *book-02*.

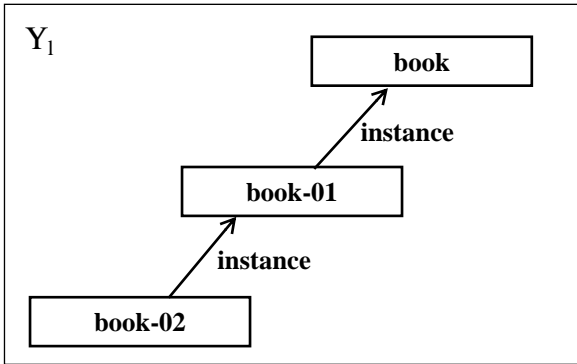


FIGURE 2. Violating the instance restriction.

A more appropriate representation would require either retyping the information for book-02 and making it an instance of book, or (as shown in Figure 3) creating a new link, say copy-of, that has the same inheritance characteristics as instance, yet has a different (and correct) interpretation.

3 The Instance Inference

KR systems and Object-Oriented systems define another language construct for establishing a *subclass* relationship between two classes. The interpretation of this link can vary in subtle ways [Brachman83], but is widely understood to have special implications with respect to the instance relationship. In particular, we define the *instance inference*:

- I2** If in some universe of discourse U , object A is an instance of object B , and B is a subclass of object C , then A is an instance of C .

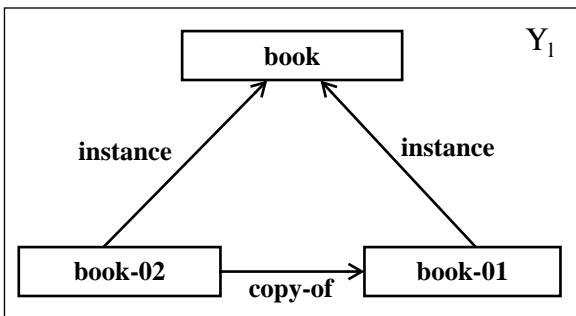


FIGURE 3. A correct representation.

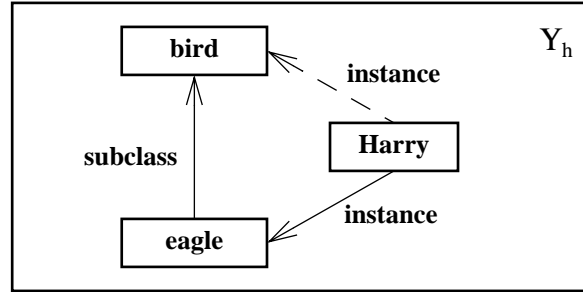


FIGURE 4. An instance of a subclass.

In examples like that shown in Figure 4, it is generally accepted that Harry is an instance of bird by virtue of eagle being a subclass of bird and Harry being an instance of eagle.

I2 is an important part of the pure semantics (which describes the rules of a language [Carnap61]) of Object Oriented and most KR languages. This inference is commonly exploited to establish set membership and inherit properties. A procedure e.g. to determine the cardinality of the set of all birds would use **I2** to count Harry as a member of that set. A property of birds, for example that they have beaks, would be considered true of all the instances of bird, and would, by **I2**, therefore be true of Harry.

4 The Instance Anomaly

We have shown that **I1** is an important restriction because problems arise when it is violated in modeling a domain, and that **I2** is an essential inference in most modeling languages. There are modeling tasks, however, in which the intuitive representation seems to call for the dismissal of either **I1** or **I2**.

Consider adding to the objects shown in Figure 4 the object species, the class of animal species. The intuitive representation of this object would be as a class, of which eagle is an individual, since eagle is a specific species and we can envision properties of the individuals of the class spe-

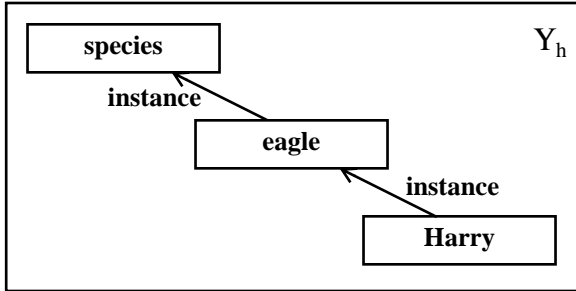


FIGURE 5. Eagle violates the instance restriction.

cies (such as the species population growth rate and whether or not the species is endangered) applying to eagle. For example, it is reasonable to say, “The eagle is endangered,” and this is understood to apply not to individuals of the class of eagles, but to the class itself. The fact that eagles are endangered does not mean that any particular eagle, such as Harry, is endangered. The resulting representation is illustrated in Figure 4, and is a clear violation of **I1** since eagle is both a class (of which Harry is an instance) and an instance of the class species.

To avoid violating **I1** another representation would make species a superclass of eagle. This representation is shown in Figure 6, and while it satisfies **I1**, the application of **I2** produces undesirable results: Harry will be an instance of species. This violates our interpretation of the object Harry; he is an eagle and a bird, but he is not a species. This is more than a minor interpretation problem since, as an individual of the class species, Harry will have the properties associated with individuals of that class. It does not make any sense at all to talk about the rate of population growth for Harry.

A third approach to modeling these objects would be to represent species as a class which has a new object, the eagle, as an instance. This approach is shown in Figure 7, and seems to satisfy **I1** since no object is represented as a class and an instance, and there is no connection between species and Harry. There are serious problems with this approach, however, espe-

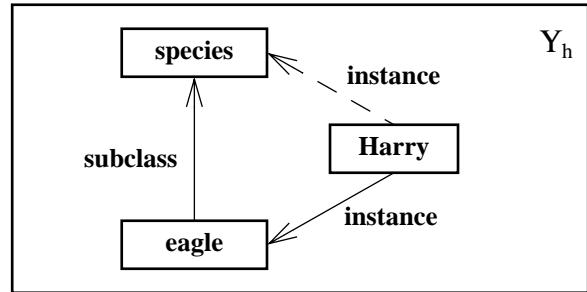


FIGURE 6. An unwanted inference.

cially in the interpretation. The class of eagles is a single object which is broken into two parts, one part the eagle, represents an instance of species and has the properties of such instances. The other part, eagle, represents the class which describes instances such as Harry. It is absolutely unclear what the relationship between the eagle and eagle should be, yet *they represent the same domain object*. In addition, Harry and the eagle are both individuals in this universe, but their interpretation as individuals is different – the eagle does not exist in the same way that Harry does.

5 Spanning Universes of Discourse

The problems in representing the example discussed in the previous section are not problems with **I1** or **I2**. The intuitive approach to representing this example (shown in Figure 4) fails because the object eagle has two interpretations: as an instance of species and as a class that describes Harry. The problems arise because modeling languages are unable to support objects that can be interpreted as classes and

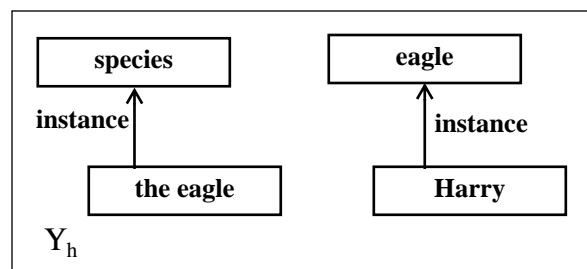


FIGURE 7. Breaking it up into two objects.

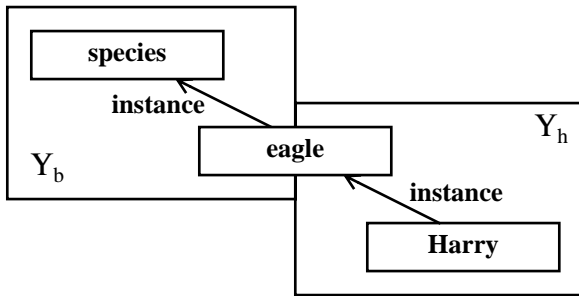


FIGURE 8. The same class as a spanning object.

instances.

This kind of multiple interpretation occurs when a model consists of two or more universes of discourse, where e.g. in universe Y_1 , an object Σ_1 is an instance and in universe Y_2 the same object Σ_1 is a class. We will say that in such cases object Σ_1 spans Y_1 and Y_2 , and that Σ_1 is a *spanning object*.

Figure 10 illustrates the use of spanning objects in the aforementioned example domain. This representation specifies that in universe U_b , eagle is an individual and an instance of species, with all the properties of such instances. In universe U_h eagle is a class and Harry is an individual. The objects Harry and species exist in different universes of discourse, and thus have no relationship to each other, and **I1** holds within each universe.

Many examples of spanning objects can be found in the domain of Computer Science. One common case is that of a language and its *meta-language* [Carnap61]. A metalanguage is essentially a language whose instances are other languages. In Figure 9, for example, Y_g denotes the universe of formal grammars. Y_g contains an object cfg that describes the class of *context-free grammars*. One instance of this class, pascal, is the context-free grammar for the language Pascal. This object also exists in Y_p , the universe of programs, where it is viewed as a description of the syntax of pascal programs. An instance of

that description is a pascal program. In this example, it is clear that pascal has two interpretations: as an instance of a context-free grammar in Y_g , and as a syntactic description of a class of programs in Y_p .

6 Previous Work

The notion of spanning objects has existed implicitly in knowledge representation and in object oriented languages for some time. These previous intuitions were driven by a perceived need to represent properties of class objects in conventional modeling languages which normally allow only instances to have their own properties.

6.1 The Abstraction Relationship

The *abstraction relationship* between a generic and individual exists when a class is abstracted into an individual in order that it may have properties [Brachman83]. Looking back to Figure 7, we had a case where eagle was split into two parts so that it could have properties such as endangered. One of the problems that arose from this approach was the relationship between the two objects eagle and the eagle.

The abstraction relationship was described merely to point out that, in a system where only instances can have properties, if classes were to have properties they would have to be represented as instances as well as classes. When represented in this manner, there would have to be

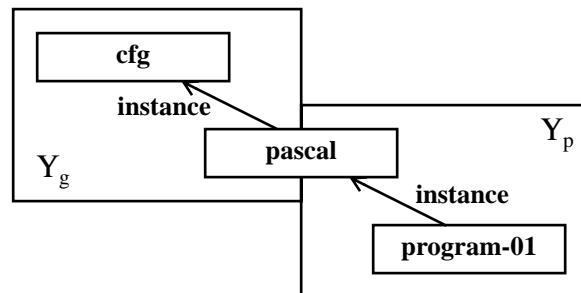


FIGURE 9. An object that spans universes.

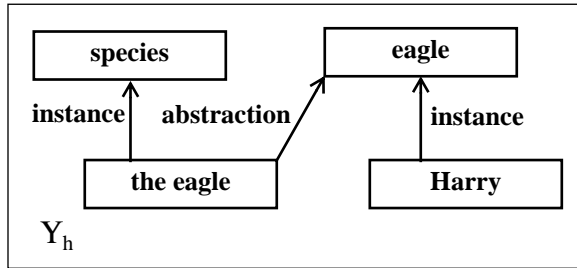


FIGURE 10. A class abstracted into an individual.

some relationship to denote the fact that the two objects should be interpreted as different views of the same object. Figure 10 shows the use of the abstraction relationship between the class and instance parts of eagles.

Although the specifics of the relationship were never defined, its existence to denote special kinds of individuals that should not be interpreted in the same context as other individuals helps avoid problems with **I1**. There are subtle problems with the abstraction relationship, however. First, although the semantics have been defined as different, the eagle and Harry are both individuals in the same universe of discourse and they are not both truly individuals – the eagle does not exist in the same way Harry does. Second, endangered is a property of eagle, not of some other concept, and it is unclear what the relationship between them means in terms of inference. Further, the relationships between the eagle and Harry and between eagle and species are similarly made ambiguous as a result of using two objects (eagle and the eagle) to represent the same domain entity.

6.2 Smalltalk

Another example of the implicit existence of spanning objects can be found in object-oriented languages that offer a meta-class facility like Smalltalk [Goldberg83] or CLOS [Bobrow88]. In Smalltalk, each class is an instance of the class class, which describes the prototypical nature of classes themselves. Class definitions

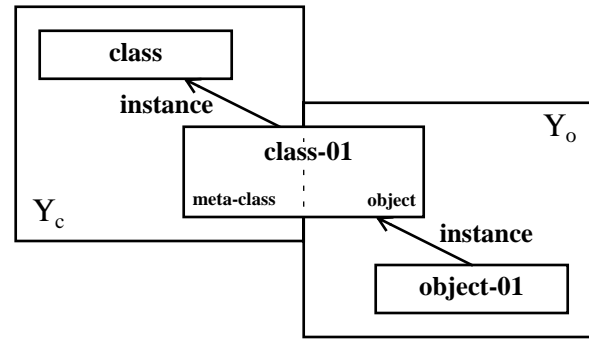


FIGURE 11. The Smalltalk class structure.

are then divided into two parts: the *object description* which describes the nature of the objects which will be instances of the class, and the *meta-class description* which describes attributes of the class which have nothing to do with the objects (like the method for creating new instances, and variables that might hold information like how many instances this class has).

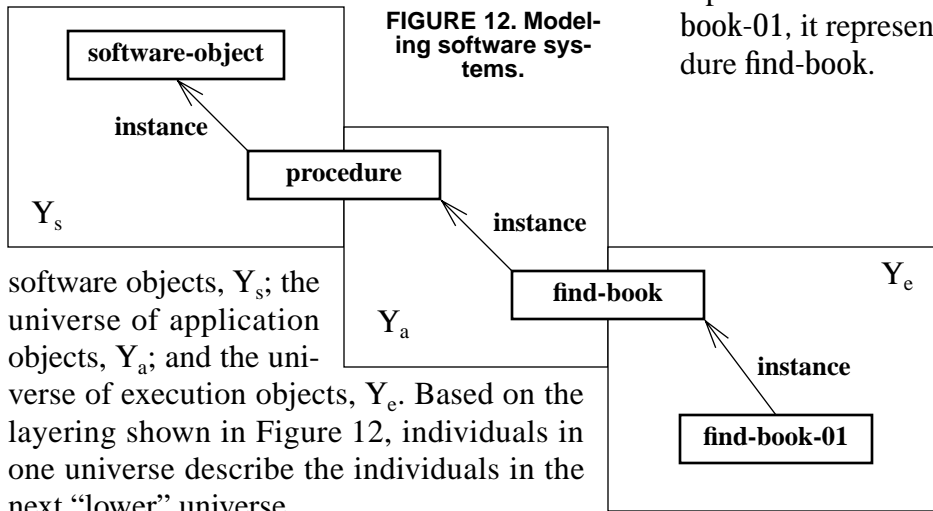
The two parts of the class description are basically a representation of two universes of discourse, as shown in Figure 11. The meta-class description contains information about the class object *as an individual*, and the object description contains information about the object *as a description*. In this sense, **I1** does hold for these languages.

While we recognize the use of spanning objects by Smalltalk, the notion remains implicit and is not provided as a modeling construct that can be applied to an arbitrary layering of universes by the modeler. There are only two universes of discourse allowed – in Figure 11 these are shown as Y_c and Y_o , the universes where classes are instances and where classes are descriptions, respectively. The example involving eagles could not be represented in Smalltalk using the language constructs provided – only instances of class may span universes. KR languages and FOL also provide no such support.

7 Modeling Software Systems

We have found that it is important to make explicit an unambiguous facility for spanning universes of discourse. The need to specify such a facility becomes especially evident when modeling software systems (or any abstract domain) where objects in one domain are themselves representations for objects in other domains [Welty93]. In some sense, modeling using multiple universes of discourse can be likened to the representation levels of semantic networks [Brachman79]. Each universe, or level, is built on objects in the previous universe.

Figure 12 shows a domain model used for modeling software systems which requires three distinct universes of discourse: the universe of



software objects, Y_s ; the universe of application objects, Y_a ; and the universe of execution objects, Y_e . Based on the layering shown in Figure 12, individuals in one universe describe the individuals in the next “lower” universe.

The universe Y_s contains objects that describe software systems. The individuals in this universe are the constructs of programming languages. One such construct, procedure, is a familiar programming language construct used to represent a module in a software system that performs some operation, and exists as an instance of software-object in U_s .

In universe Y_a the object procedure exists as a class, where it describes procedures. The indi-

viduals in this universe are all the parts of the software system being modeled (such as the procedures, data types, files, etc.). In the case where the application being modeled is a software system for a library card catalog, one of the individuals in Y_a would be find-book – the procedure for finding a book in the card catalog database given some key information.

The third universe of discourse, Y_e , contains information about the software system during a particular execution. The individuals in this universe represent various aspects of the running software system and here find-book is a class that describes all calls to the procedure. One of the individuals in this domain, find-book-01, represents the procedure find-book being called to find some book. Note that find-book-01 does not represent what to do to find some object called book-01, it represents an invocation of the procedure find-book.

8 Conclusion

The instance restriction is important to the unambiguous interpretation of represented knowledge; when the same construct in a modeling language is used to represent

instances and classes, undesirable ambiguities result. The constructs for instance and class must be kept distinct. Certain objects, however, act as classes and instances in different contexts, and these special objects must be recognized by the modeling language (and the modeler) as existing in multiple universes of discourse. In each universe, the object is interpreted as either a class or an instance. This approach also avoids undesirable inferences when classes appear in taxonomies.

We believe that clarifying an ambiguity in the use of instances and classes is important for domain modeling and knowledge representation in general. While supporting the notion of spanning objects and multiple universes of discourse in modeling languages is a necessary part of this clarification, it does give rise to some interesting questions that must still be addressed. The most prominent question is on the relationship between the universes in which a spanning object exists. What relationships can be expressed between objects in different universes? Can inferences made about a spanning object in one universe be used in another universe in which the object exists? If such mapping across universes is allowed, it may affect the completeness and tractability of the inference mechanisms, since the essential nature of spanning objects is second-order.

References

- [Bobrow88] Bobrow, D. and Kiczales, G. The Common Lisp Object System metaobject kernel: A status report. In *Proceedings of the 1988 ACM Conference on Lisp and Functional Programming*. ACM SIGART. pp 309-315.
- [Brachman77] Brachman, R. What's in a concept. *Int. J. Man-Machine Studies*. Volume 9, 1977. pp 127-152.
- [Brachman79] Brachman, R. and Levesque, H. On The Epistemological Status of Semantic Networks. In Brachman, R. and Levesque, H., eds., *Readings in Knowledge Representation*, pp 169-189. Morgan Kaufman, 1985.
- [Brachman83] Brachman, R. What is-a is and isn't. *IEEE Computer*. October, 1983. pp 30-36.
- [Brachman85] Brachman, R. and Schmolze, J. An overview of the KL-ONE Knowledge Representation System. *Cognitive Science*. Volume 9, pp 171-216. 1985.
- [Brachman89] Brachman, R., Borgida, A., McGuinness, D., and Resnick, L. The CLASSIC Knowledge Representation System. In *Proceedings of the 11th IJCAI*. August, 1989.
- [Carnap47] Carnap, R. *Meaning and Necessity*. U. of Chicago Press, 1947.
- [Carnap61] Carnap, R. *Introduction to Semantics and Formalization of Logic*. Harvard University Press, 1961.
- [Fox79] Fox, M. *On Inheritance in Knowledge Representation*. CMU Robotics Institute Technical Report number CMU-RI-79024. 1979.
- [Fox85] Fox, M., Wright, J., and Adam, D. Experiences with SRL. In *Expert Database Systems*. Benjamin Cummings. 1985.
- [Gödel31] Gödel, K. Über Formal Unentscheidbare Sätze der Principia Mathematica und Verwandter Systeme I. *Monatshefte für Mathematik und Physik*, Volume 38, pp 173-198. 1931.
- [Goldberg83] Goldberg, A. and Robson, D. *Smalltalk-80: The Language and its Implementation*. Addison-Wesley. 1983.
- [Maida82] Maida, A. and Shapiro, S. Intensional Concepts in Propositional Semantic Networks. In Brachman, R. and Levesque, H., eds., *Readings in Knowledge Representation*, pp 169-189. Morgan Kaufman, 1985.
- [Patil92] Patil, R., Fikes, R., Patel-Schneider, P., McKay, D., Finin, T., Gruber, T., and Neches, R. The DARPA Knowledge Sharing Effort: Progress Report. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann. 1992.
- [Quine64] Quine, W. *From a logical point of view*. Harvard University Press, 1964.
- [Quine69] Quine, W. *Ontological Relativity and Other Essays*. Columbia University Press. 1969.
- [Shapiro87] Shapiro, S. and Rappaport, W. SNePS considered as a fully intensional propositional semantic network. In Cercone, N., and McCalla, G., eds., *The Knowledge Frontier*. pp 262-315. Springer-Verlag, 1987.
- [Welty93] Welty, C. *An Integrated Representation for Software Development and Discovery*. RPI Computer Science Tech Rpt. July, 1993.
- [Woods75] Woods, W. What's in a Link. In Brachman, R. and Levesque, H., eds., *Readings in Knowledge Representation*, pp 217-241. Morgan Kaufman, 1985.