Int. J. Human–Computer Studies (1995) **43**, 907–928

Toward principles for the design of ontologies used for knowledge sharing†

THOMAS R. GRUBER

Stanford Knowledge Systems Laboratory, 701 Welch Road, Building C, Palo Alto, CA 94304, USA. email: Gruber@ksl.stanford.edu

Recent work in Artificial Intelligence (AI) is exploring the use of formal ontologies as a way of specifying content-specific agreements for the sharing and reuse of knowledge among software entities. We take an engineering perspective on the development of such ontologies. Formal ontologies are viewed as designed artifacts, formulated for specific purposes and evaluated against objective design criteria. We describe the role of ontologies in supporting knowledge sharing activities, and then present a set of criteria to guide the development of ontologies for these purposes. We show how these criteria are applied in case studies from the design of ontologies for engineering mathematics and bibliographic data. Selected design decisions are discussed, and alternative representation choices are evaluated against the design criteria.

1. Introduction

Several technical problems stand in the way of shared, reusable knowledge-based software. Like conventional applications, knowledge-based systems are based on heterogeneous hardware platforms, programming languages, and network protocols. However, knowledge-based systems pose special requirements for interoperability. Such systems operate on and communicate using statements in a formal knowledge representation. They ask queries and give answers. They take "background knowledge" as an input. And as agents in a distributed Artificial Intelligence (AI) environment, they negotiate and exchange knowledge. For such knowledge-level communication, we need conventions at three levels: representation language format, agent communication protocol, and specification of the content of shared knowledge. Proposals for standard knowledge representation formats (Morik, Causse, & Boswell, 1991; Fulton, 1992; Genesereth & Fikes, 1992) and agent communication languages (Finin, Weber, Wiederhold, Genesereth, Fritzson, McKay, McGuire, Pelavin, Shapiro & Beck, 1992) are independent of the content of knowledge being exchanged or communicated. For establishing agreements about knowledge, such as shared assumptions and models of the world, ontologies can play a software specification role (Gruber, 1991).

Current research is exploring the use of formal ontologies for specifying content-specific agreements for a variety of knowledge-sharing activities (Gruber & van Baalen, 1991; Neches, Fikes, Finin, Gruber, Patil, Senator & Swartout, 1991;

† Substantial revision of paper presented at the International Workshop on Formal Ontology, March 1993, Padova, Italy. Available as Technical Report KSL 93-04, Knowledge Systems Laboratory, Stanford University.

1071-5819/95/120907 + 22$12.00/0

Genesereth, 1992; Allen & Lehrer, 1992; Gruber, Tenenbaum, & Weber, 1992; Walther, Eriksson, & Musen, 1992; Patil, Fikes, Patel-Schneider, McKay, Finin, Gruber & Neches, 1992; Cutkosky, Engelmore, Gruber, Genesereth, Mark, Tenenbaum & Weber, 1993). A long-term objective of such work is to enable libraries of reusable knowledge components and knowledge-based services that can be invoked over networks. We believe the success of these efforts depends on the development of an engineering discipline for ontology design, akin to software engineering for conventional software.

This paper is an analysis of design requirements for shared ontologies and a proposal for design criteria to guide the development of ontologies for knowledge-sharing purposes. Section 2 describes a usage model for ontologies in knowledge sharing. Section 3 proposes some design criteria based on the requirements of this usage model. Section 4 shows how these criteria are applied in ontologies designed explicitly for the purpose of knowledge sharing.

2. Ontologies as a specification mechanism

A body of formally represented knowledge is based on a *conceptualization*: the objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them (Genesereth & Nilsson, 1987). A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly.

An ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an Ontology is a systematic account of Existence. For AI systems, what "exists" is that which can be represented. When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of discourse. This set of objects, and the describable relationships among them, are reflected in the representional vocabulary with which a knowledge-based program represents knowledge. Thus, in the context of AI, we can describe the ontology of a program by defining a set of representational terms. In such an ontology, definitions associate the names of entities in the universe of discourse (e.g. classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms. Formally, an ontology is the statement of a logical theory.†

We use common ontologies to describe *ontological commitments* for a set of agents so that they can communicate about a domain of discourse without necessarily operating on a globally shared theory. We say that an agent *commits* to an ontology if its observable actions are consistent with the definitions in the

† Ontologies are often equated with taxonomic hierarchies of classes, class definitions, and the subsumption relation, but ontologies need not be limited to these forms. Ontologies are also not limited to *conservative definitions*, that is, definitions in the traditional logic sense that only introduce terminology and do not add any knowledge about the world (Enderton, 1972). To specify a conceptualization one needs to state axioms that *do* constrain the possible interpretations for the defined terms.

ontology. The idea of ontological commitments is based on the *Knowledge-Level* perspective (Newell, 1982). The Knowledge Level is a level of description of the knowledge of an agent that is independent of the symbol-level representation used internally by the agent. Knowledge is attributed to agents by observing their actions; an agent "knows" something if it acts *as if* it had the information and is acting rationally to achieve its goals. The "actions" of agents—including knowledge base servers and knowledge-based systems—can be seen through a *tell and ask* functional interface (Levesque, 1984), where a client interacts with an agent by making logical assertions (tell), and posing queries (ask).

Pragmatically, a common ontology defines the vocabulary with which queries and assertions are exchanged among agents. Ontological commitments are agreements to use the shared vocabulary in a coherent and consistent manner. The agents sharing a vocabulary need not share a knowledge base; each knows things the other does not, and an agent that commits to an ontology is not required to answer all queries that can be formulated in the shared vocabulary.

In short, a commitment to a common ontology is a guarantee of consistency, but not completeness, with respect to queries and assertions using the vocabulary defined in the ontology.

3. Design criteria for ontologies

Formal ontologies are *designed.* When we choose how to represent something in an ontology, we are making design decisions. To guide and evaluate our designs, we need objective criteria that are founded on the purpose of the resulting artifact, rather than based on *a priori* notions of naturalness or Truth. Here we propose a preliminary set of design criteria for ontologies whose purpose is knowledge sharing and interoperation among programs based on a shared conceptualization.

(1) Clarity: An ontology should effectively communicate the intended meaning of defined terms. Definitions should be *objective.* While the motivation for defining a concept might arise from social situations or computational requirements, the definition should be independent of social or computational context. *Formalism* is a means to this end. When a definition can be stated in logical axioms, it should be. Where possible, a *complete* definition (a predicate defined by necessary and sufficient conditions) is preferred over a partial definition (defined by only necessary or sufficient conditions). All definitions should be documented with natural language.

(2) Coherence: An ontology should be coherent: that is, it should sanction inferences that are consistent with the definitions. At the least, the defining axioms should be logically consistent. Coherence should also apply to the concepts that are defined informally, such as those described in natural language documentation and examples. If a sentence that can be inferred from the axioms contradicts a definition or example given informally, then the ontology is incoherent.

(3) Extendibility: An ontology should be designed to anticipate the uses of the shared vocabulary. It should offer a conceptual foundation for a range of

anticipated tasks, and the representation should be crafted so that one can extend and specialize the ontology *monotonically*. In other words, one should be able to define new terms for special uses based on the existing vocabulary, in a way that does not require the revision of the existing definitions.

(4) *Minimal encoding bias:* The *conceptualization* should be specified at the knowledge level without depending on a particular symbol-level encoding. An *encoding bias* results when a representation choices are made purely for the convenience of notation or implementation. Encoding bias should be minimized, because knowledge-sharing agents may be implemented in different representation systems and styles of representation.

(5) Minimal ontological commitment: An ontology should require the minimal ontological commitment sufficient to support the intended knowledge sharing activities. An ontology should make as few claims as possible about the world being modeled, allowing the parties committed to the ontology freedom to specialize and instantiate the ontology as needed. Since ontological commitment is based on *consistent* use of *vocabulary*, ontological commitment can be minimized by specifying the weakest theory (allowing the most models) and defining only those terms that are essential to the communication of knowledge consistent with that theory.

3.1. TRADEOFFS

Ontology design, like most design problems, will require making tradeoffs among the criteria. However, the criteria are not inherently at odds. For example, in the interest of clarity, definitions should restrict the possible interpretations of terms. Minimizing ontological commitment, however, means specifying a weak theory, admitting many possible models. These two goals are not in opposition. The clarity criterion talks about definitions of terms, whereas ontological commitments is about the conceptualization being described. Having decided that a distinction is worth making, one should give the tightest possible definition of it.

Another apparent contradiction is between extendibility and ontological commitment. An ontology that anticipates a range of tasks need not include vocabulary sufficient to express all the knowledge relevant to those tasks (requiring an increased commitment to that larger vocabulary). An extensible ontology may specify a very general theory, but include the representational machinery to define the required specializations.

Extendibility and ontological commitment both include a notion of sufficiency or adequacy. Since an ontology serves a different purpose than a knowledge base, the concept of *representational adequacy* (McCarthy & Hayes, 1969) differs. A shared ontology need only describe a vocabulary for talking about a domain, whereas a knowledge base may include the knowledge needed to solve a problem or answer arbitrary queries about a domain.

To see how these abstract criteria can be used guide ontology design, we will consider two case studies. In each, we will discuss selected design decisions, and evaluate alternative representation choices against the proposed criteria.

4. Case studies in ontology design

In this section we discuss the design of two ontologies. In conventional data modeling, one would define ontologies with data-type declarations or a database schema. Since we wish to write knowledge-level specifications, independent of particular data or programming languages, we use the knowledge interchange format KIF (Genesereth & Fikes, 1992).† Each ontology defines a set of classes, relations, functions, and object constants for some domain of discourse, and includes an axiomatization to constrain the interpretation. The resulting language (the basic logic from KIF + the vocabulary and theory from the ontologies) is a domain-specific specification of a conceptualization.

4.1. A QUICK OVERVIEW OF KIF

KIF is a prefix notation for predicate calculus with functional terms and equality. Free variables, which start with the prefix ?, are universally quantified. Material implication is indicated with the $=\!\!>$, $<\!\!=$, and $<\!\!=\!\!>$ operators. Equality between terms is denoted by the $=$ relation. The `member` relation indicates set-membership, and `setof` is the set construction operator. Relations are first-class objects in the universe of discourse, defined as sets of tuples. Relations are denoted by constants that serve as both predicate symbols and as terms denoting the relations as objects. Functions are a special case of relations, where a function of N arguments is equivalent to a relation of $N+1$ arguments whose last argument is the value of the function on the first N arguments. Classes are represented with unary relations. For example, the sentence `(C ?q)` means `?q` is an instance of the class C. Definitions are given by the KIF definitional operators `defrelation`, `deffunction`, and `defobject`, which associate a relation, function, or object constant with the set of axioms that follow. In the style of Lisp syntax, case is not significant in constants and the text following a semicolon on the same line is ignored.

4.2. CASE 1: PHYSICAL QUANTITIES, UNITS OF MEASURE, AND ALGEBRA FOR ENGINEERING MODELS

In the first case study, we consider the problem of representing mathematical models of engineering systems. Engineers use mathematical models, such as sets of equations, to analyse the behavior of physical systems. The conventional notations for formatting mathematical expressions in textbooks and the engineering literature usually leave implicit many of the details required to understand the equations. For instance, it is not clear from the expression $f = kx + c$ which symbols are variables or constants; whether they represent numbers or quantities (e.g. forces, lengths); whether the numbers are reals or vectors; whether the quantities are static values, functions of time, or functions of time and space; and the units of measure assumed. The reader must interpret these notations using background knowledge and context.

† To support development, we used a set of KIF-based analysis and translation tools provided by the Ontolingua system (Gruber, 1992; Gruber, 1993).

To build libraries of reusable engineering models, it is important to have a notation based on a formal language with a well-defined, context-independent interpretation. For this purpose, Greg Olsen and I have developed a family of formal ontologies for engineering modeling. Our goal was to define the vocabulary and conceptual foundation necessary for sharing mathematical models of device behavior among computer programs. These ontologies can then be used as a language for communication among engineering tools in a distributed, heterogeneous environment (Gruber *et al.,* 1992; Kuokka, Pelavin, Weber, Tenebaum, Gruber & Olsen, 1993).

The most important concepts in the ontologies are *physical-quantity* (e.g. 3 m, 80 km/h), *physical-dimension* (length, length/time), *unit-of-measure* (m, km/h), *magnitudes* of various orders (scalars, vectors, tensors, and functions thereof), *algebras* for describing mathematical constraints (e.g. operators for products, sums, exponentiations, and derivatives), and the concepts at a *metalinguistic* level to describe mathematical expressions as objects in the universe of discourse (e.g. to reason about whether an expression is in closed form or to describe the dependent variables in a system of equations).

We now focus on the representation of physical quantities in engineering models. Engineers often use the word "quantity" to refer to both the thing-in-the-world, such as the length of some segment of railway track (about 3 m), and the thing-in-the-equation, such as the parameter X in an equation model denoting the position of a car along the track. In the engineering ontologies, we distinguish these two notions. A physical-quantity is the extent of some property in the world independent of how it is represented or measured (e.g. the length that is 3 m long). The symbol X is an expression that denotes a physical quantity. A representation of X is at the metalinguistic level, where parameters and equations in an engineering model are treated as special cases of terms and sentences in the logic. The treatment of the metalinguistic level is beyond the scope of this paper, so we will concentrate on the object level representation of physical quantity.

4.2.1. Version 1

Consider a straw-man proposal, which we will criticize and revise. We will start with a straightforward definition of a physical quantity: a pair comprising a number and a unit of measure. The KIF form below defines a `physical-quantity` as an object consisting of a magnitude and a unit, which are given by the unary functions `quantity.magnitude` and `quantity.unit`. The definition says that the magnitude of a physical quantity must exist† and be of type `double-float` and its unit must be a member of the standard set of units. These constraints are analogous to slot value restrictions in object-centered languages. Since the definition is an if-and-only-if ((=)) condition, it also says that every pair of such magnitudes and units defines a quantity.

† In KIF, functions may be partial (only defined for some arguments). The predicate defined is used to indicate that a function has a value for a particular sequence of arguments. In relational terminology, `(defined (f x))` implies that there exists a y such that, for the function F viewed as a binary relation, $F(x, y)$ holds. In slot-value terminology, it means that slot f has exactly one value on object x.

```
(defrelation PHYSICAL-QUANTITY
  ((=>) (PHYSICAL-QUANTITY ?q)
        (and (defined (quantity.magnitude ?q))
             (double-float (quantity.magnitude ?q))
             (defined (quantity.unit ?q))
             (member (quantity.unit ?q)
                     (setof meter second kilogram
                            ampere kelvin mole candela)))
```

`Physical-quantity` is a class (i.e. a unary relation that holds over instances of the class). For describing individual instances of the class, we define a constructor function called `the-quantity`. The term expression `(the-quantity ?m ?u)` denotes a physical quantity `?q` whose magnitude is `?m` and unit is `?u`.

```
(deffunction THE-QUANTITY
  ((=>)(and (defined (THE-QUANTITY ?m ?u))
            (= (THE-QUANTITY ?m ?u) ?q))
       (and (physical-quantity ?q)
            (= (quantity.magnitude ?q) ?m)
            (= (quantity.unit ?q) ?u))))
```

The definition of `physical-quantity` already stated that all quantities are determined by the values of their magnitudes and units; this definition simply adds vocabulary to be able to denote a specific quantity with a term. For example, the following states that X is the quantity 3 m, where `meter` is one of the possible units of measure.

```
(= X (the-quantity 3 meter))
```

Analysis of version 1. Our original proposal satisfies some of the design criteria for ontologies. The definition of `physical-quantity` is specified declaratively—its meaning is independent of any program. It is internally consistent, and simple enough to be clear. However, it falls short on some of the other criteria. First, the `double-float` constraint, while familiar to data definitions, is a specification of the *precision* of the encoding of numbers rather than the concept of physical quantities. This is an instance of *encoding bias* because it reflects an implementation detail (bits of precision) rather than the knowledge-level commitments of parties to the ontology. It would be better to say that the magnitude of such a quantity is a real number, acknowledging that all computer programs must approximate the reals.†

Second, the concept of units of measure is defined as a set of possible values for the `quantity.unit` function. This is also a specification of the *encoding* rather than the world, although it is not as obvious as the double-float vs. real-number example. What we mean by physical quantity does not depend on a commitment to a particular set of units. Therefore, the set of possible units should not be part of the definition of physical-quantity.

Third, fixing the set of possible units is a limit on *extendibility*. The world has many standards for units of measure, and the purposes of this ontology (sharing

† For sharing knowledge about discrete approximation, such as theories of error bounds on numeric computation, axioms about precision could be written without encoding bias.

enginering models and theories across people, domains, and tools) do not sanction a preference for one standard. Therefore, the ontology should allow for the definition of alternate sets of units, and a way to relate them to existing units.

4.2.2. Version 2

Consider how we could modify the initial proposal to remove these inadequacies. First, we can reformulate the definition of physical-quantity so that the concepts of magnitude and unit of measure are made explicit as independent classes, `magnitude` and `unit-of-measure`.

```
(defrelation PHYSICAL-QUANTITY
  (⟨=⟩ (PHYSICAL-QUANTITY ?q)
       (and (defined (quantity.magnitude ?q))
       (magnitude (quantity.magnitude ?q))
       (defined (quantity.unit ?q))
       (unit-of-measure (quantity.unit ?q))))))
```

Now we can define the class `magnitude` to include the class of all real numbers, rather than the class of numbers encodable in floating point format. The class `real-number` comes from KIF's number ontology.

```
(defrelation MAGNITUDE)
  (⟨= (MAGNITUDE ?x)
    (real-number ?x)))
```

Note that the definition above is *incomplete.* A complete definition of a relation includes necessary and sufficient conditions for the relation to hold. The definition above gives sufficient conditions for being a member of the class—that all real numbers are magnitudes—but not necessary conditions. This is done in anticipation that there will be other sorts of magnitudes in engineering models.

The complete family of engineering math ontologies includes separate theories for vector and tensor quantities. These specialized theories inherit from parent theories all the above axioms about quantitites and magnitudes, and add axioms stating that vectors and tensors are also magnitudes.

Decoupling the ontologies in this way helps to minimize *ontological commitment.* An agent can commit to the basic theory of real-valued quantities without any commitment to higher order magnitudes. The rationale for minimizing commitment is the assumption that greater commitment implies a more complex agent.

To accommodate alterate sets of units, we define a class for units of measure, and provide vocabulary with which to define new units. We start with a class called `unit-of-measure`, which is a *primitive.*†

```
(defrelation UNIT-OF-MEASURE
  (class (UNIT-OF-MEASURE))
```

† A primitive term is one for which we are not able to give a complete axiomatic definition. We must rely on textual documentation and a background of knowledge shared with the reader to convey the meanings of primitives. Technically, all terms with incomplete definitions are primitives. Some, like `magnitude` can be strengthened with further conditions to become complete (e.g. `magnitude` could be made equivalent to `real-number`). Others, like `unit-of-measure`, must get their meaning from human interpretation (the meter is a unit of measure purely by convention).

To allow the user to extend the set of units we define a basis set called basic-unit (a sub-class of unit-of-measure) and operators called unit* and unit^ for building new units from existing one. Unit* is analogous to multiplication and unit^ is analogous to exponentiation to a real power.‡ In an engineering equation, if two quantities are multiplied, their product can be expressed as the quantity whose magnitude is the arithmetic product of the magnitudes of the quantities and whose unit of measure is the unit* of the units of the two quantities. The analogous relationship holds for exponentiation, which introduces division through negative exponents.

```
(defrelation BASIC-UNIT
  (=⟩ (BASIC-UNIT ?u)        ;  basic units are distinguished
    (unit-of-measure ?u))) ;  units of measure

(deffunction UNIT*
  ; Unit* maps all pairs of units to units
  (=⟩ (and (unit-of-measure ?u1)
           (unit-of-measure ?us))
    (and (defined (UNIT* ?u1 ?u2))
         (unit-of-measure (UNIT* ?u1 ?u2))))
  ; It is commutative
  (= (UNIT* ?u1 ?u2) (UNIT* ?u2 ?u1))
  ; It is associative
  (= (UNIT* ?u1 (UNIT* ?u2 ?u3))
    (UNIT* (UNIT* ?u1 ?u2) ?u3)))

(deffunction UNIT^
  ; Unit^ maps all units and reals to units
  (=⟩ (and (unit-of-measure ?u)
           (real-number ?r))
    (and (defined (UNIT^ ?u ?r))
         (unit-of-measure (UNIT^ ?u ?r))))
  ; It has the algebraic properties of exponentiation
  (= (UNIT^ ?u 1) ?u)
  (= (unit* (UNIT^ ?u ?r1) (UNIT^ ?u ?r2))
    (UNIT^ ?u (+ ?r1 ?r2)))
  (= (UNIT^ (unit* ?u1 ?u2) ?r)
    (unit* (UNIT^ ?u1 ?r) (UNIT^ ?u2 ?r)))
```

For example, one can define meters as a basic-unit for length and seconds as a basic unit for time. We then define a unit for velocity called m/s using the combination operators. We can use this new unit to denote a specific quantity of velocity.

‡ In a later revision of the physical quantities theory, we defined units as special cases of quantities. We also included real numbers as quantities with an identity dimension. Since quantities became a generalization of numbers, we extended KIF's * and expt functions with polymorphic definitions for each sub-class of quantities, including units of measure. This allowed us to eliminate unit* and unit^, and substitute * and expt throughout. In this formulation, the-quantity reduces to *.

```
(defobject METER
  (basic-unit METER))

(defobject SECOND
  (basic-unit SECOND))

(defobject METER/SECOND
  (= METER/SECOND
    (unit* meter (unit^ second -1))))

(= REAL-FAST (the-quantity 10000 meter/second))
```

Analysis of version 2. This new axiomatization of quantities, magnitudes, and units is extensible, but it turns out to be *incoherent*. Six feet and two yards are both descriptions of the same length, but if we use the definition above for physical quantities, treating them as pairs of magnitudes and units, then ⟨6 feet⟩ and ⟨2 yards⟩ are *not* the same quantity! This is easily proved: If

```
(= (quantity.unit (the-quantity 6 foot)) foot)
(= (quantity.unit (the-quantity 2 yard) yard)
```

and

```
(not (= foot yard))
```

then

```
(not (= the-quantity 6 foot) (the-quantity 2 yard))).
```

This conclusion is consistent with (and follows from) the axiomatic definitions. However, in our conceptualization, these quantities are lengths in the world, independent of how they are measured. Thus 6 feet and 2 yards *should* be equal, and our proof has revealed a problem. If the representation allows one to infer something that is not true in the conceptualization then the ontology is incoherent.†

4.2.3. Version 3

The original representation for quantities as aggregates with magnitude and unit components reflects a style encouraged by object-oriented data modeling techniques. Objects are entities whose state is given by instance variables or methods. One can imagine implementing quantities as tuples of numbers and units. But in this domain, this image led us astray. If we abstract away from the implementation, we can solve the problem of equivalent quantities by changing the definitions of the functions that relate quantities to magnitudes and units. In the next version, we say that

† Of course we have no means of mechanically verifying coherence with the conceptualization. We do have means of deriving consequences of the axioms, which we must interpret with respect to the intended conceptualization.

the magnitude of a quantity depends on the unit in which it is requested. This dependency is reflected in the definition below, in which `quantity.magnitude` becomes a binary function that maps quantities *and* units to magnitudes. With this definition, we avoid the incoherence; it is possible for two quantities specified in different units to be equal.

```
(deffunction QUANTITY.MAGNITUDE
  (=>) (and (defined (QUANTITY.MAGNITUDE ?q ?u))
            (= (QUANTITY.MAGNITUDE ?q ?u) ?m))
    (and (physical-quantity ?q)
         (unit-of-measure ?u)
         (magnitude ?m))))
```

So far we have provided a formal mechanism to describe new units and physical quantities, but have not said enough about the semantics of quantities with respect to units and other quantities. This is an issue of *clarity*. Can any combination of quantities be compared on magnitudes, or added? Which units make sense for which quantities? How is 1000 m/s like 1 mile/h but unlike 6 m? The missing link is the notion of *physical dimension*. Length, time, and velocity are physical dimensions. Both units and physical quantities are associated with physical dimensions, and the units used to specify a quantity must be of the same dimension as the quantity. Only quantities of the same dimension can be added or compared.

We can define a primitive class called physical-dimension (e.g. length, velocity), and say that a unit-of-measure must be associated with a single physical dimension. The definition of `unit-of-measure` below states the latter constraint with a total functional mapping (via the function `unit.dimension`) from units to physical dimensions.

```
(defrelation PHYSICAL-DIMENSION
  (class PHYSICAL-DIMENSION))

(defrelation UNIT-OF-MEASURE
  (=>) (UNIT-OF-MEASURE ?u))
    (and (defined (unit.dimension ?u))
         (physical-dimension (unit.dimension ?u)))))
```

Similarly, we can define the function `quantity.dimension` mapping all quantities to their dimensions.

```
(deffunction QUANTITY.DIMENSION
  (=>) (physical-quantity ?q)
    (and (defined (QUANTITY.DIMENSION ?q))
         (physical-dimension (QUANTITY.DIMENSION ?q)))))
```

We now have a notion of "compatibility" among quantities and units (i.e. equal dimensions). We can specify this by strengthening the definition of the function `quantity.magnitude`, adding the constraint (underlined below) that the dimensions of the quantity and unit must be the same.

```
(deffunction QUANTITY.MAGNITUDE
  (=>) (and (defined (QUANTITY.MAGNITUDE ?q ?u))
```

```
        (= (QUANTITY.MAGNITUDE ?q ?u) ?m))
    (and (physical-quantity ?q)
         (unit-of-measure ?u)
         (magnitude ?m)
         (= (quantity.dimension ?q)
            (unit.dimension ?u))))
```

By introducing the concept of physical dimension, we have been able to add more axiomatic constraints to the definitions (increasing clarity) without changing the ontological commitment. It doesn't make any sense to ask for the magnitude of a length in seconds; these constraints just make this understanding explicit.

One final extension of the ontology *does* increase the ontological commitment, but is consistent with the existing definitions and makes the theory more useful for sharing engineering models. In the extension, we say that the magnitudes of any two quantities of the same dimension can be compared, regardless of the units in which they were originally specified. We can state this formally by making the function `quantity.magnitude` total for all quantities and units of the same dimension, adding another axiom to the definition of quantity.magnitude

```
(deffunction QUANTITY.MAGNITUDE
  (=>) (and (defined (QUANTITY.MAGNITUDE ?q ?u))
            (= (QUANTITY.MAGNITUDE ?q ?u) ?m))
       (and (physical-quantity ?q)
            (unit-of-measure ?u)
            (magnitude ?m)
            (= (quantity.dimension ?q)
               (unit.dimension ?u)))))
  (=>) (and (quantity ?q)
            (unit-of-measure ?u)
            (= (quantity.dimension ?q)
               (unit.dimension ?u)))
       (defined (QUANTITY.MAGNITUDE ?q ?u)))
```

Without this axiom, the definition of `quantity.magnitude` stipulates necessary conditions—domain and range constraints that must hold whenever the function is defined.

The practical consequence of this commitment is that unit conversion is now possible. Given any quantity `?q`, originally specified in unit `?u`, there is a magnitude `?m = (quantity.magnitude ?q ?u)`. From the previous axiom, for any other unit `?u2` of the same dimension we can expect there to exist a magnitude `?m2 = (quantity.magnitude ?q ?u2)`. Thus, `quantity.magnitude` provides the vocabulary for unit conversion.

Based on this ontology, Yves Peligry of the Stanford Knowledge Systems Laboratory has written an agent that can perform unit conversion operations. It commits to the physical quantities ontology, and answers queries in KIF using the vocabulary from the ontology. It gets its information about the dimensions and relative magnitudes of various units by reading other ontologies containing unit definitions. The unit ontologies describe *systems* of units in which some dimensions are fundamental, with standard units, and the rest are composed using

multiplication and exponentiation operators. For example, one such ontology specifies the standard SI system of units (Halliday & Resnick, 1978). Given such an ontology, the agent can determine whether the provided unit definitions are sufficient to compute the magnitude of any quantity given in one unit in terms of another. Then, given a query of the form

```
(evaluate (quantity.magnitude quantity-expression unit-expression))
```

the agent can return the appropriate magnitude.

This is an example of using an ontology for agent interoperation. The ontology provides the *vocabulary* from which to construct such queries, and the semantics so that two agents can agree on what makes sense in a given vocabulary. In this case, the agents can agree about which quantity expressions and term expressions denote quantities and units, and when they are given as arguments to the `quantity.magnitude` function, that the magnitude exists. These agreements establish a basis for agent discourse.

Separating the core ontology about quantities and units from the specific conventions for systems of units minimizes the *ontological commitment* of participating agents. While they all need to commit to the core theory, they can commit to differing standards of measure. Since commitment to an ontology does not require completeness of inference, agents can "understand" the conditions under which a value exists (e.g. a magnitude in some unknown unit) without knowing how to compute the value. This makes it possible to capitalize on the services of "mediator" agents (Wiederhold, 1992) to translate between agents. In this case, the mediator is the unit conversion agent.

4.2.4. Summary of case 1

We have seen how a formal ontology for engineering models can be designed with respect to the general design criteria we outlined in the Section 3. Criterion 1 asks for *clarity*. In the engineering math ontologies, clarity is achieved from a formal axiomatization, which forced us to make explicit the distinctions and assumptions that are often left ambiguous in textbook notations. Since the resulting ontology is machine interpretable, the "reader" can use a theorem prover to help answer the kinds of questions one finds at the end of textbook chapters (questions that probe the student's understanding of the defined concepts). The second criterion, *coherence* guided the design of the ontology throughout. In the case study we found that the definitions in an early version implied a conclusion that was inconsistent with our conceptualization (equivalent quantities expressed in different units were not equal). Criterion 3, *extendibility*, also suggested a design change: instead of fixing a set of units, we added vocabulary to define new units and dimensions, and to allow for quantities to be expressed in any compatible unit. We saw an instance of *encoding bias* (criterion 4) in the assumption about the precision of magnitudes. To help minimize *ontological commitment* (criterion 5) we decomposed a comprehensive theory of engineering mathematics into modular theories, so that agents that who could reason about scalar quantities but not higher-order magnitudes, or agents that work in different standard units, could commit to the smallest theory needed to share models with other agents.

We will now consider a domain in which the application of the same design criteria leads to different choices in the representation.

4.3. CASE 2: AN ONTOLOGY FOR SHARING BIBLIOGRAPHIC DATA

The second case study involves an ontology for sharing bibliographic data, such as the information in a library's card catalog and the reference list at the end of a scholarly paper. This example will clarify some of the design issues relating ontologies and shared data.

The bibliography ontology is designed to facilitate automatic translation among existing bibliographic databases, to support the specification of reference-formatting styles independently of database or program, and to provide a knowledge-level vocabulary for interacting with network-based services such as bibliography database search.

Any such ontology describes a conceptualization, a view of a world from a particular perspective. In this ontology, the world includes information. We say that a bibliography is made up of *references*. A reference contains the information needed to identify and retrieve a publication. References contain *data* about publications; references are not the publications themselves. *Documents* are the things created by authors that can be read, viewed, listened to, etc. Books and journals are documents. There may be references for several publications per document, as in edited collections. Documents are created by *authors*, who are *people* or other *agents*. Documents are published by *publishers* at publication dates, which are *time points*. Authors and publishers have *names*, which are strings.

The bibliography ontology defines a class for each of the concepts italicized above. For example, bibliographic references are represented by instances of the class called `reference`. The definition of this class includes the necessary condition that a reference be associated with a document and a title. The document associated with a reference is represented by the `ref.document` function and the title with `ref.title`.

```
(defrelation REFERENCE
; A bibliographic reference is a description of some publication
; that uniquely identifies it, providing the information needed
; to retrieve the associated document. A reference is
; distinguished. . .
  (=> (REFERENCE ?ref)
      (and (defined (ref.document ?ref))
           (defined (ref.title ?ref)))))

(deffunction REF.DOCUMENT
; ref.document maps references to documents
  (=>)] (and (defined (REF.DOCUMENT ?ref)
              (= (REF.DOCUMENT ?ref) ?doc))
      (and (reference ?ref)
           (document ?doc))))

(deffunction REF.TITLE
```

```
; ref.title maps references to title strings
  (⟸) (and (defined (REF.TITLE ?ref))
           (= (REF.TITLE ?ref) ?title))
       (and (reference ?ref)
            (title-name ?title))))
```

In a database, an entity such as a reference might be encoded as a record or relation tuple. The vocabulary for representing the data associated with these entities—the fields of a database record—are provided by unary functions and binary relations. For example, the `ref.title` function represents the title field of a reference, whose value is a title-name (a string). The sentence `(defined (ref.title ?ref))` in the definition of `reference` states that a valid reference *must* have an associated title. Another field is the `ref.year`, which will be defined below. It is an "optional field"—not all references will include the year information.

The `ref.document` function is different from the "data field" functions. `Ref.document` represents the relationship between a reference and a document. The document is an element of our conceptualization, but is not a data item like a name string or number. It is used to help specify the relationship of other data. For example, in an edited collection, all the papers share the same publication data. This is because, in the world, the papers are published in a single document. In each reference, the fields for the publisher and year of publication are determined by the publisher and year of publication of the document. This is true whether or not the document is represented in a database.

The relationships between the data in a reference and the facts about documents and authors is provided by unary functions and binary relations. Let us consider the `ref.year` function in detail. The function `ref.year` maps a reference to an integer representing the year in which the associated document was published. If the associated document was not published, then the `ref.year` function is undefined for that reference. The relationship between references, documents, and years is specified below: The `ref.year` of a reference is an integer that is the `timepoint.year` of the time point that is the `doc.publication-date` of the document associated with the reference.

```
(deffunction REF.YEAR)
  (= (ref.year ?ref)
     (timepoint.year (doc.publication-date (ref.document ?ref)))))

(deffunction DOC.PUBLICATION-DATE
; the timepoint at which the document was published
  (⇒ (and (defined (DOC.PUBLICATION-DATE ?DOC))
          (= (DOC.PUBLICATION-DATE ?doc) ?date))
     (and (document ?doc)
          (timepoint ?date))))
```

Notice that the mapping from references to year numbers involves an intermediate entity called a `timepoint`. A `timepoint` is a single point in real, historical time—the same for all observers and contexts. It is continuous, and its meaning is independent of the granularity at which we wish to approximate it. The date of birth of a person and the date of publication of a document are timepoints.

```
(defrelation TIMEPOINT
; A timepoint is a point in real, historical time—the same for all
; observers and contexts. It is independent of measurement
; resolution.
  (class TIMEPOINT))
```

`Timepoint.year` is a function from time points to integers representing the number of years since a fixed reference time point.

```
(deffunction TIMEPOINT.YEAR
; the number of years between a timepoint and the origin timepoint
; of the Christian calendar. The value is an integer, rounded up.
  (=> (and (defined (TIMEPOINT.YEAR ?t))
           (= (TIMEPOINT.YEAR ?t) ?y))
      (and (timepoint ?t)
           (integer ?y))))
```

4.3.1. Analysis

At this point we have enough context to examine the rationale for the central design decision in this ontology: to define conceptual entities described by the data, rather than just specifying the data. Why are documents distinguished from references? Why introduce publishers and authors as independent entities when only their names appear in references? Why is a theory of time points included, when dates in references always appear as numbers and strings? Do the anticipated applications need to reason about dates, publication events, authorship, and so forth? If not, why ask them to commit to these concepts in an ontology?

One reason for introducing these entities is to be able to state *integrity constraints* about the data in the reference object. For instance, if two papers are published in the same edited collection, then one would like to ensure that their publication data (date, publisher, city, etc.) are the same. By representing documents independently of references, we are able to write such constraints. Database agents can use integrity constraints to guarantee coherence and detect data processing errors.

Representing the conceptual entities in addition to the data also provides some independence from application-specific encoding and resolution. We distinguish timepoints, rather than just talking about year numbers, for this reason. We say that documents are published at points in historical time, which, when mentioned in references, may be approximated at various levels of precision. If one reference claims that a journal article was published in "1993," and another that the article was published in "March 1993," then we have the representational machinery to determine that these two references are consistent. We could, instead, insist that all dates be represented in some canonical form, such as day/month/year. However, this would be specifying a precision that may not be available in the data, and would vary over implemented systems. Since this commitment would follow from a particular format rather than from the conceptualization, it would be a kind of *encoding bias.*

On the other hand, one might also question the rationale for describing the data (strings, numbers) at all. Why represent the name of an author separately from the author, or the integer encoding of years separately from time points? One could

instead limit the ontology to the conceptual entities such as documents, authors, and time points, and leave it to the implementations to decide how a particular author might be named or date might be stored in a database. More generally, the issue is whether *identifiers* for objects are appropriate to represent in knowledge-level specifications.

Typically, the choice of identifiers for objects in a knowledge base is a symbol-level issue. For example, a database may identify employees with unique integers that serve as keys in employee records. If the identifier is an artifact of the database design, and not in the world, then it is not appropriate to include in a shared ontology. Defining employees in terms of numbers is an example of encoding bias.

In the domain of the bibliography ontology, however, symbolic identifiers such as references and names are properly in the domain of discourse. The reference object, and its data fields such as the year of publication, are in the conceptualization of the bibliographic domain. It is part of the convention of bibliographic references that years are encoded as integers. This is independent of *format*, but it is an encoding. Similarly, the names of authors are important elements of the domain. For example, an early version of the bibliography ontology was revised to accommodate the concept of pen names: multiple names for the same author. That authors sometime use pen names, and that these names are the official author names in publications, is a property of the world, not a database schema.

Nonetheless, we need to distinguish the identifiers from the entities being identified to preserve coherence. Names don't write; authors do!

The identification of dates is a more subtle case. The notion of dates in the bibliography domain assumes a coordinate system and unit of measure for historical time. The domain also has standards for the precision in which time is measured (integral years, months, and days). Why is this not an *ecoding bias*? The commitment to a specific unit of measure might also seem to limit *extendibility*. Would an agent that works from a Chinese calendar be able to commit to an ontology based on the Western calendar? Compare this situation to the physical quantities ontology (Section 4.2), in which we were careful to keep the concept of a quantity independent of the unit in which it is known, and to provide a mechanism for supporting multiple, equivalent units.

The specification of a standard measurement or identification scheme does not *inherently* impose an encoding bias or limit extendibility. The bibliography ontology does not *equate* the specification of dates with the concept of points in historical time. The notion of timepoint is independent of units, just as the concept of thermodynamic temperature is independent of whether it is measured on the Kelvin or Rankine scale. The unit and coordinate system are introduced by the mapping from time points to the surface encoding of the data via functions such as `timepoint.year`.

The agent working on the Chinese calendar can read the date specified using the Western calendar, and convert it into the appropriate internal format. In addition, because encodings are distinguished from the conceptual entities, one can extend the existing ontology to handle unforeseen units. For example, some publication dates use the name of a season, as in "Summer 1993". A function from time points to seasons could be defined without contradicting the underlying ontology.

5. Summary and discussion

We have described five general design criteria and have given detailed examples of the design of formal ontologies for knowledge sharing. In these case studies, we found several instances of encoding bias, ranging from prescriptions of numerical precision to implicit assumptions resulting from viewing a quantity as a magnitude/unit pair. We showed that when an encoding is intrinsic to the conceptualization (integers for years), encoding bias can be avoided by representing both the underlying concept (time points) and the units of measure (timepoint years). We showed how to extend the representational vocabulary (e.g. for units of measure in the engineering ontology) without overcommitment (e.g. to particular units as an exclusive standard).

We found that the evaluation of design decisions against the criteria depends on the knowledge available and the applications anticipated for a domain. In the engineering domain, we have a strong theory with which to relate the concepts of physical quantity, unit of measure, dimension, and magnitude. The ontology is specified with great clarity, since most of the concepts can be defined axiomatically. The resulting specification imposes real constraints on the implementation of agents (e.g. they are prohibited from making implicit assumptions about units), allows for program- and notation-independent knowledge bases (libraries of engineering models), and provides the basis for useful inferential services (e.g. behavior prediction and unit conversion). The ontological commitment to a strong theory is justified for the sharing of valuable mathematical models. In the bibliography domain, the theory relating references, documents, people, publishers, and dates is weak. We could have imposed a stronger theory, for instance, describing a world where all authors have permanent, unique names and the month is always known for a conference paper. That would have been imposing more ontological commitment than is necessary to share the information. It would also be incoherent, since we know that these constraints do not hold in the world being modeled.

5.1. RELATED WORK

The design criteria proposed in this paper are primarily for evaluation of design decisions in choosing among representations, rather than as guidelines for generating theory. Choosing appropriate ways of conceptualizing a domain is a constructive modeling task, a topic of active research (e.g. Ford & Bradshaw, 1993).

The LILOG project followed a set of software engineering principles, such as modularity and abstraction, in the development of its ontology (Pirlein, 1993). These principles guide the form and organization of knowledge specifications, and are complementary to those proposed here, which pertain to choice of representation. The design of the Penman ontology (Bateman, Kasper, Moore, & Whitney, 1990) draws from knowledge of the structure and content of natural language (including the work of linguists, philosophers, psychologists, and lexicographers). Skuce and Monarch (1990) recommend a similar strategy for general knowledge acquisition. In a workshop on the LILOG ontology, Simmons (Simmons, 1991) discussed content-independent ontology design criteria, including the software engineering principles and some criteria similar to those proposed (independently) in this paper. Reporting on experience in formalizing theories of objects and their properties, he concludes

that these content-independent criteria are less important than the validity of the theory itself (i.e. as a scientific theory of cognition). For both LILOG and Penman, the purpose of the ontology is to organize knowledge used for natural language processing, and so the ontology must account for distinctions found in language. In contrast, for the ontologies discussed in this paper, which specify a common conceptualization for knowledge sharing among programs, whether the conceptualization is a good model of the world was not the dominant criterion. In any case, an ontology is only a specification, and the utility of an ontology ultimately depends on the utility of the theory it represents.

Several examples of ontologies are being developed to enable knowledge sharing and reuse. The Cyc project is generating a wealth of examples (Lenat & Guha, 1990; Lenat, Guha, Pittman, Pratt, & Shepherd, 1990) and methods for partitioning knowledge bases into modular theories (Guha, 1991). Guarino (1992), Sowa (1993) and others offer content-independent guidelines for organizing concepts, relations, and individuals in an ontology. Comprehensive "top level" frameworks such as the Penman Upper Model (Bateman *et al.*, 1990), the upper structure of LILOG, Skuce's ontology, Takagaki's adaptation of Mario Bunge's ontology (Takagaki, 1990), the Ontek ontology, and the upper reaches of Cyc's ontology offer partial designs that can be extended and instantiated for particular needs. There are a number of ontologies focusing on special representation problems, such as varieties of time (Allen, 1984; Ladkin, 1986; Shoham, 1987), space (Cohn, 1995), part–whole structure (Eschenbach & Heydrich, 1995; Gerstl & Pribbenow, 1995), causality and change (Hobbs, 1995; Terenziani, 1995). Formal specifications of domain or task specific conceptualizations are beginning to appear in the research literature. Ontologies for the sharing and reuse of knowledge about engineering models (Kiriyama, Yamamoto, Tomiyama, & Yoshikawa, 1989; Alberts, 1993), planning and scheduling (Allen & Lehrer, 1992; Hama, Hori, & Nakamura, 1993), manufacturing enterprises (Fox, 1993) and problem-solving tasks and methods (Steels, 1990; Musen, 1992; Walther *et al.*, 1992; Ericksson, Puerta, & Musen, 1994) can be viewed as modular building blocks for reusable knowledge bases and ontology-specific software. Of particular relevance is the KADS methodology for expert system development, which is based on sharing and reusing "models of expertise" (theories about tasks, domains, and solution strategies) (Wielinga, Schreiber, & Breuker, 1992; Wielinga, Velde, Schreiber, & Akkermans, 1992). These models are essentially ontologies, and recent work has begun to produce formal specifications (Akkermans, van Harmelen, Schreibner, & Wielinga, 1993; Aben, 1993; van Harmelen & Balder, 1992; Angele, Fensel, & Landes, 1992; Fensel & Studer, 1993).

We have a start on a technology to put such ontologies in portable form, for comparison and exchange (Genesereth & Fikes, 1992; Gruber, 1993). With this we can begin to accumulate a corpus of examples of ontology design. As we learn more about the design of ontologies for knowledge sharing, we may be able to evolve today's preliminary design criteria into working design principles.

The intellectual foundation for the ontology work is the product of a rewarding collaboration with colleagues on the ARPA Knowledge Sharing Effort, particularly Richard Fikes, Mike Genesereth, Bill Mark, Bob Neches, Ramesh Patil, Marty Tenenbaum, and Jay Weber. Thanks to Danny Bobrow, Richard Fikes, Hania Gajewska, Pat Hayes, James McGuire, and

Greg Olsen for their thoughts on ontology design and thoughtful reviews of this paper. The work is supported by ARPA prime contract DAAA15-91-C-0104 through Lockheed subcontract SQ70A3030R, and NASA Grants NCC 2-537 and NAG 2-581 (under ARPA Order 6822).

References

ABEN, M. (1993). Formally specifying re-usable knowledge model components. *Knowledge Acquisition*, **5**, 119–141.

AKKERMAN, H., VAN HARMELEN, F., SCHREIBER, G. & WIELINGA, B. (1993). A formalization of knowledge-level models for knowledge acquisition. *International Journal of Intelligent Systems*, **8**, 169–208.

ALBERTS, L. K. (1993). *YMIR: an ontology for engineering design*. Ph.D. dissertation, University of Twente.

ALLEN, J. & LEHRER, N. (1992). *DARPA/Rome Laboratory Planning and Scheduling Initiative Knowledge Representation Specification Language (KRSL), Version 2.0.1 Reference Manual*. ISX Corporation.

ALLEN, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence*, **23**, 123–154.

ANGELE, J., FENSEL, D. & LANDES, D. (1992). *Conceptual modelling with KARL: four applications*. Technical Report 231, University of Karlsruhe.

BATEMAN, J. A., KASPER, R. T., MOORE, J. D. & WHITNEY, R. A. (1990). *A general organization of knowledge for natural language processing: the penman upper model*. Technical report, USC/Information Sciences Institute, Marina del Rey, CA.

COHN, T. (1995). A taxonomy of logically defined qualitative spatial relations. *International Journal of Human–Computer Studies*, **43**, 831–846.

CUTKOSKY, M., ENGELMORE, R. S., FIKES, R. E., GRUBER, T. R., GENESERETH, M. R., MARK, W. S., TENENBAUM, J. M. & WEBER, J. C. (1993). PACT: An experiment in integrating concurrent engineering systems. *IEEE Computer*, **26**, 28–37.

ENDERTON, H. B. (1972). *A Mathematical Introduction to Logic*. San Diego, CA: Academic Press.

ERICKSSON, H., PUERTA, A. & MUSEN, M. A. (1994). Generation of knowledge-acquisition tools from domain ontologies. *International Journal of Human–Computer*, **41**, 425–453.

ESCHENBACH, C. & HEYDRICH, W. (1995). Classical mereology and restricted domains. *International Journal of Human–Computer Studies*, **43**, 723–740.

FENSEL, D. & STUDER, R. (1993). *An analysis of languages which operationalize and formalize KADS models of expertise*. Technical report 237, University of Karlsruhe.

FIKES, R., CUTKOSKY, M., GRUBER, T. & VAN BAALEN, J. (1991). *Knowledge sharing technology project overview*. Technical Report KSL 91–71, Stanford University, Knowledge Systems Laboratory.

FININ, T., WEBER, J., WIEDERHOLD, G., GENESERETH, M., FRITZSON, R., MCKAY, D., MCGUIRE, J., PELAVIN, P., SHAPIRO, S. & BECK, C. (1992). *Specification of the KQML agent-communication language*. Technical Report EIT TR 92-04, Enterprise Integration Technologies, Palo Alto, CA.

FORD, K. M. & BRADSHAW, J. M. (Eds.). (1993). *Knowledge Acquisition as Modeling*. New York, NY: John Wiley & Sons. Special issue of *International Journal of Intelligent Systems*, Volume 8, 1993.

FOX, M. (1993). A common-sense model of the enterprise. *Proceedings of the Industrial Engineering Research Conference*. Longer version available as report from Department of Industrial Engineering, University of Toronto.

FULTON, J. A. (1992). *Technical report on the semantic unification meta-model*. Standards working document ISO TC184/SC4/WG3 N103, IGES/PDES Organization, Dictionary/Methodology Committee. Contact James Fulton, Boeing Computer Services, P.O. Box 24346, MS 7L-64, Seattle, WA 98124-0346.

GENESERETH, M. R. (1992). An Agent-Based Framework for Software Interoperability.

Proceedings of the DARPA Software Technology Conference. Meridian Corporation, Arlington VA, p. 359–366. Also Report Logic-92-2, Computer Science Department, Stanford University, June 1992.

GENESERETH, M. R. & FIKES, R. E. (1992). *Knowledge interchange format, version 3.0 reference manual.* Technical Report Logic-92-1, Computer Science Department, Stanford University.

GENESERETH, M. R. & NILSSON, N. J. (1987). *Logical Foundations of Artificial Intelligence.* San Mateo, CA: Morgan Kaufmann.

GERSTL, P. & PRIBBENOW, S. (1995). Midwinters, end games, and bodyparts: a classification of part–whole relations. *International Journal of Human–Computer Studies,* **43,** 865–889.

GRUBER, T. R. (1991). The role of common ontology in achieving sharable, reusable knowledge bases. In J. A. ALLEN, R. FIKES, & E. SANDEWALL, (Eds.) *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference.* pp. 601–602 Cambridge, MA: Morgan Kaufmann.

GRUBER, T. R. (1992). *Ontolingua: a mechanism to support portable ontologies.* Technical Report KSL 91-66, Stanford University, Knowledge Systems Laboratory. Revision.

GRUBER, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition,* **5,** 199–220.

GRUBER, T. R., TENENBAUM, J. M. & WEBER, J. C. (1992). Toward a knowledge medium for collaborative product development. In J. S. GERO, Eds. *Artificial Intelligence in Design '92.* Boston, MA: Kluwer.

GUARINO, N. (1992). Concepts, attributes, and arbitrary relations: some linguistic and ontological criteria for structuring knowledge bases. *Data and Knowledge Engineering,* **8,** 249–261.

GUHA, R. V. (1991). *Contexts: a formalization and some applications.* Ph.D. dissertation, Stanford University, USA.

HALLIDAY, D. & RESNICK, R. (1978). *Physics.* New York, NY: John Wiley and Sons.

HAMA, T., HORI, M. & NAKAMURA, Y. (1993). *Task-specific language constructs for describing constraints in job assignment problems.* Technical Report, IBM Japan.

HOBBS, J. (1995). Sketch of a proposed ontology that underlies the way we talk about the world. *International Journal of Human–Computer Studies,* **43,** 819–830.

KIRIYAMA, T., YAMAMOTO, F., TOMIYAMA, T. & YOSHIKAWA, H. (1989). Metamodel: an integrated modeling framework for intelligent CAD. In J. S. GERO, Eds. *Artificial Intelligence in Design.* Southampton: Computational Mechanics Publications.

LADKIN, P. (1986). Time representation: a taxonomy of interval relations. *Proceedings of the Sixth National Conference on Artificial Intelligence.* p. 354–359, Philadelphia.

LENAT, D. B. & GUHA, R. V. (1990). *Building Large Knowledge-based Systems: Representation and Inference in the Cyc Project.* Menlo Park, CA: Addison-Wesley.

LENAT, D. B., GUHA, R. V., PITTMANN, K., PRATT, D. & SHEPHERD, M. (1990). Cyc: toward programs with common sense. *Communications of the ACM,* **33,** 30–49.

LEVESQUE, H. J. (1984). Foundations of a functional approach to knowledge representation. *Artificial Intelligence,* **23,** 155–212.

MCCARTHY, J. & HAYES, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In B. MELTZER & D. MICHIE, Eds. *Machine Intelligence 4.* Edinburgh: Edinburgh University Press.

MCGUIRE, M J., PELAVIN, R., WEBER, J. C., TENENBAUM, J. M., GRUBER, T. R. & OLSEN, G. (1992). SHADE: Technology for knowledge-based collaborative engineering. *Journal of Concurrent Engineering: Applications and Research (CERA),* **1,** 137–146.

MORIK, K., CAUSSE, K. & BOSWELL, R. (1991). *A common knowledge representation integrating learning tools.* Technical Report, GMD.

MUSEN, M. A. (1992). Dimensions of knowledge sharing and reuse. *Computers and Biomedical Research,* **25,** 435–467.

NECHES, R., FIKES, R. E., FININ, T., GRUBER, T. R., PATIL, R., SENATOR, T. & SWARTOUT, W. R. (1991). Enabling technology for knowledge sharing. *AI Magazine,* **12,** 16–36.

NEWELL, A. (1982). The knowledge level. *Artificial Intelligence,* **18,** 87–127.

PATIL, R. S., FIKES, R. E., PATEL-SCHNEIDER, P. F., MCKAY, D., FININ, T., GRUBER, T. R.,

& NECHES, R. (1992). The DARPA knowledge sharing effort: progress report. In C. RICH, B. NEBEL, & W. SWARTOUT, Eds. *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference.* Cambridge, MA: Morgan Kaufmann.

PIRLEIN, T. (1993). Reusing a large domain-independent knowledge base. *Proceedings of the Fifth International Conference on Software Engineering and Knowledge Engineering,* San Francisco, CA, USA.

SHOHAM, Y. (1987). Temporal logics in AI: Semantical and ontological considerations. *Artificial Intelligence,* **33,** 89–104.

SIMMONS, G. (1991). Empirical methods for "ontological engineering"—case study: objects—. In G. KLOSE, E. LANG, & T. PIRLEIN, Eds. *Ontologie und Axiomatik der Wissensbasis von LILOG.* Berlin: Springer-Verlag.

SKUCE, D. & MONARCH, I. (1990). Ontological issues in knowledge base design: some problems and suggestions. *Fifth Knowledge Acquisition for Knowledge Based Systems Workshop.* Banff.

SOWA, J. F. (1995). Top-level ontological categories. *International Journal of Human–Computer Studies*, **43,** 669–685.

STEELS, L. (1990). Components of expertise. *AI Magazine*, Summer 1990, 27–49.

TAKAGAKI, K. (1990). *A formalism for object-based information systems development.* Ph.D. dissertation, The University of British Columbia.

TERENZIANI, P. (1995). Towards a causal ontology coping with the temporal constraints between causes and effects. *International Journal of Human–Computer Studies,* **43,** 847–863.

VAN HARMELEN, F. & BALDER, J. (1992). $(ML)^2$: a formal language for KADS conceptual models. *Knowledge Acquisition,* **4,** 127–161.

WALTHER, E., ERIKSSON, H. & MUSEN, M. A. (1992). *Plug and play: construction of task-specific expert-system shells using sharable context ontologies.* Technical Report KSI-92-40, Knowledge Systems Laboratory, Stanford University.

WIEDERHOLD, G. (1992). Mediators in the architecture of future information systems. *IEEE Computer*, **25,** 38–49.

WIELINGA, B., VELDE, W. V. D., SCHREIBER, G. & AKKERMANS, H. (1992). The CommonKADS framework for knowledge modelling. *Knowledge Acquisition for Knowledge Based Systems Workshop*, Banff.

WIELINGA, B. J., SCHREIBER, A. T. & BREUKER, J. A. (1992). KADS: a modelling approach to knowledge engineering. *Knowledge Acquisition,* **4,** 5–53.