3

Ontological Commitments in Knowledge-Based Design Software: A Progress Report

Filippo A. Salustri, P.Eng.
Department of Mechanical Engineering, Ryerson Polytechnic Institute, 350 Victoria Street, Toronto, ON, M5B 2K3, Canada

Keywords: logic, ontological commitment, knowledge representation, design

Abstract: The increased sensitivity of engineered products to external forces requires new computer-based design tools that can express the richness and complexity of product knowledge. This paper is a progress report of the author's research towards the development of such a knowledge-based design tool, called the Design Knowledge Specification Language (DKSL). A key goal is to ensure the maximum possible logical rigor. In order to do this, *ontological commitments* are constructed to map logical structures to the domain of design knowledge. The first part of the paper discusses a number of ontological commitments the author has discovered for design. The second part of the paper presents the current, incomplete implementation of DKSL. An example of the structural and steady-state thermal analysis of a wall is used to present DKSL's capabilities. Although much work remains to be done, it appears that DKSL may be able to accurately and rigorously describe any design knowledge.

1. INTRODUCTION

As engineered products and engineering processes become more sensitive to economic, social, and technological forces, CAE tools must be made to express the richness and complexity of the information used in engineering environments. To treat the complexity, CAE systems are moving towards a knowledge-based approach, that is distributed transparently over intranets and

S. Finger et al. (eds.), *Knowledge Intensive Computer Aided Design*

the Internet. One of the author's on-going research projects is the design and implementation of a network-centric knowledge-based system (KBS) for engineering design applications.

There are many other research efforts aiming at the same or similar goals. For general knowledge representation (KR) applications, KIF, Ontolingua, and KQML (Genesereth and Fikes, 1992; Gruber, 1992), developed at Stanford University, and the KL-ONE family of languages (Brachman et al., 1991) are the most well-developed. These are very large systems able to represent general knowledge in a number of domains. In engineering domains, KIF has found some applications (Hurst, 1991; Hakim and James H. Garrett, 1993). Systems specific to engineering have also been developed (Alberts, 1994; Eastman et al., 1991b).

In the author's view, there are two problems with these approaches. Those efforts aimed at general KR (such as KIF) have to manage "common sense" knowledge, which is substantially different than the technical knowledge typical in engineering environments. On the other hand, those efforts specific to engineering tend to have relatively informal foundations. The author believes that it is possible to develop logical systems for CAE tools that are more rigorous than those currently available, yet are targeted specifically to engineering. The key to achieving this is in developing a proper set of *ontological commitments*, which formalize the correspondence between logical structures and the domain of engineering design.

This paper gives an overview of the current status of the project, starting with a discussion of the ontological commitments that have been established to date. The second part of the paper covers the implementation of DKSL (Design Knowledge Specification Language), which embeds those commitments in a frame-based KR system.

2. BACKGROUND

The author's research involves the development of logical theories of the various aspects of engineering design. The author's current focus is in the development of a language for the description of products. In the previous KIC workshop, the author presented a formal theory of product description, called AIM-D (Salustri, 1996). This paper will discuss the ongoing development of a specification language, DKSL, which implements AIM-D in the form of a programming language environment with knowledge base (KB) capabilities. Because DKSL depends on AIM-D, a brief summary of the theory is presented here.

AIM-D is an interpretation of ZF^1 axiomatic set theory (Copi, 1979). Set theory is a basic tool of logic, used in fields like number theory to prove the existence of the integers (Bernays, 1968). Though its validity is not provable

due to Godel's Theorems, it is quite robust – robust enough, the author contends, to provide a degree of rigor for design theories that has heretofore been lacking.

Specifically, AIM-D uses the ZF axioms to define formally the information needed to model products. Currently, AIM-D covers quantities, features, parts, and assemblies, as well as sub-assemblies and systems, and types of all these entities. It does so by imposing a fixed semantics on the ZF axioms. This kind of interpretation amounts to making *ontological commitments* about the nature of designed products. Insofar as the author is ultimately interested in implementing a KBS for designed products, it is not so much the axioms as the ontological commitments that must be embedded in the KBS; the axiomatic theory demonstrates the (degree of) validity and rigor.

In closing, there are two particular points of interest about AIM-D that are noteworthy here. First, not all the axioms of ZF were used in developing AIM-D. This opens the interesting possibility that there is a logic, simpler than set theory and perhaps even demonstrably valid, that may be sufficient for design purposes.

The second point regards the ease with which ZF can formalize otherwise intuitive notions universal to design, leading to more robust computable algorithms. For example, the ZF Axiom of Foundation limits the notion of a set to those entities for which set membership is antisymmetric. That is, if A contains a set B, then A can only be a set if B does not contain A. This has a clear correspondence to the intuition that an assembly a cannot have b as a subassembly if b already contains a as a subassembly.

3. ONTOLOGICAL COMMITMENTS

An *ontological commitment* is a mapping between a language and a structure that systematically axiomatizes the forms and modes of being in a domain; this allows only certain *intended* meanings of models to be captured (Guarino et al., 1994). In other words, an ontological commitment is a decision to adhere to a certain interpretation of a language in a some domain; it is a mechanism to help ensure that a given model written in a given language communicates exactly and only what was intended by the model developer. For example, mathematical algebra includes ontological commitments regarding what variables such as x and y mean, what operators such as $+$ do, etc. Meaning can be ascribed to algebraic statements only when there is agreement on its underlying ontological commitments. Similarly, in design, there exist ontological commitments regarding the meaning of the various symbols used in blueprints. The meaning of the blueprint is lost if ontological commitments are missing or inconsistent.

The general problem of KR is that it admits a domain so broad that it is considered inappropriate to make ontological commitments about it. This is

especially difficult in representing "common sense" knowledge held by the average individual, which is often incomplete, inconsistent, or even incorrect. This means that such KR systems must be able to treat the incompleteness, inconsistencies, and outright errors, which in turn greatly complicates the whole problem.

On the other hand, in the highly technical and relatively restricted domain of engineering design, we strive to minimize these problems. Here, some ontological commitments can be used as *simplifying assumptions* to improve the robustness, complexity, and computability of knowledge representations of designed products.

Though this occurs at the expense of expressiveness, a restricted solution today is in some ways better than the promise of a more general solution tomorrow; also, such a solution for design may provide a stepping stone to more general solutions by providing experience needed to develop more powerful KBSs in the future.

In developing any KBS, some ontological commitments must be made to limit the models possible in the system to only those models *intended* by the developers of the models and the users of the KBS. Often, these commitments are only implied, opening the possibility of *mis*interpretation. A fundamental goal of the author's work is to find the basic ontological commitments needed to define product models. To this end, the ontological commitments made in the development of DKSL are discussed in this section. So far, only some arise directly from AIM-D, which is a work-in-progress; some other, more tentative commitments must be made to allow continued development of DKSL while the theory is being developed. These other commitments deal specifically with modeling aspects of product function, of the various kinds of part-whole relationship (*mereology*), and of contexts.

3.1 COMMITMENTS ARISING FROM AIM-D

AIM-D maps ZF set theory to the domain of product modeling via ontological commitments that constrain DKSL. These ontological commitments result in a hierarchy of fundamental types. At the most primitive level are *quantities*, which are tuples of a value and a dimensional metric; *5 ft*, *100 N* are examples of quantities. A *feature* defines a geometrically and functionally relevant entity that is not necessarily realizable (e.g. a hole, or a fillet); features are compositions of interrelated quantities. *Parts* are aggregates of interrelated features, that are realizable through non-assembly manufacturing processes (casting, machining, etc.). Finally, an *assembly* is an aggregate of interrelated parts that are realizable only through assembly processes. These four domains of entities are disjoint, and are fundamental for product modeling because each domain covers a unique and distinctive class of entity in a designed product.

Additionally, AIM-D supports the concepts of both sub-assemblies and systems, but as entity domains existing outside the hierarchy described above. While sub-assemblies are essential *conceptual* entities in any design or manufacturing process, the author does not believe they constitute "real-world" entities. For example, an automobile engine may be considered an assembly by the engineers that design the engine, but as a subassembly by the engineers that design the whole automobile. But whether the engine is actually in an automobile (i.e. a subassembly) or not (i.e. an assembly) does nothing to alter the essential nature of the engine. If the notion of subassembly is context-dependent, then it cannot be fundamental to AIM-D, which is intended to capture the essential nature of the product. Thus, assemblies in AIM-D are composed of manufactured (non-assembled) parts; sub-assemblies are useful, perhaps even essential, constructs for both *designing* and *manufacturing* processes, but are only *ancillary* with respect to description of products as real objects.

This approach contradicts the relatively common intuition exemplified by statements such as *"The automobile is an assembly of the following sub-assemblies. . . "*, which suggests that sub-assemblies have a substantive nature. However, once assembled, the distinction between sub-assemblies disappears without prior knowledge of the assembly process. This argument should not be taken as one diminishing the importance of sub-assemblies in engineering; it is intended only to distinguish the notion of subassembly as an artificial one, and to incorporate and formalize that distinction into AIM-D.

The more generic notion of a *system* is also formalized in AIM-D, due to its relevance in engineering. But, as with sub-assemblies, systems are ancillary. Indeed, it was found during the development of AIM-D that the only substantive difference between sub-assemblies and subsystems was that system components need not be in direct contact with each other, whereas direct contact is required for sub-assemblies; otherwise, the formalizations for systems and for assemblies are the same.

Another aspect of the ontological commitments of AIM-D regards the representation of *type* information. An extremely popular commitment is to the existence of explicit *classes* (as in object-oriented languages) or *concepts* (as in description logics such as CLASSIC). Classes and concepts are meta-level information units; this raises issues of reflection and its computational counterpart, recursion.

While there are modeling problems that can be solved efficiently with classes, there is also evidence that the human mind works more "by example" than by abstraction (Jaynes, 1976; Damasio and Damasio, 1992), and that design is one area where the generally accepted semantics of classes and types can impede the development of accurate, flexible, and robust models (Johnson and Zweig, 1991; Eastman and Fereshetian, 1994). For example, there is a tendency in

automotive engineering to regard a blueprint of an automobile as a model of a *typical* automobile and not as the *set or class* of automobiles of a certain make and model. Also, the tolerancing of dimensions can be viewed as the definition of a "vaguely" defined prototype: any item whose dimensions fit within the limits described by the tolerances can be thought of as a specific version of the more general prototype. That is, designers tend to use an exemplar-based, rather than a class-based, approach to model products. In keeping with this observation, AIM-D admits no notion of class or concept, but rather uses the notion of an *exemplar*, an entity typical of items in a collection. Collections themselves exist only intentionally. (It is noted that it is possible to develop class-based systems from prototype-based ones.)

In order to structure collections of entities, AIM-D uses notions of generalization of entity attributes. Generalization occurs in AIM-D by *ignoring* certain aspects of entity attributes. Specifically, three kinds of generalization are defined within AIM-D: ignoring whole attributes, ignoring the values of attributes, and ignoring the number of values of attributes.

3.2 COMMITMENTS DUE TO PRODUCT FUNCTION AND BEHAVIOR

There is a difference of opinion in the research community regarding the meaning *function* and *behavior*; no standardized, consistent model of these terms exists. Some researchers consider function as a description of the actions a product can perform (e.g. (Qian and Gero, 1996)), while others treat it as a description of a subset of *behaviors* (i.e. intended behavior or "purpose", as in (Sturges et al., 1996)). Various other definitions are given in (Chittaro et al., 1994; Chakrabarti, 1993).

The current author defines *behavior* as the response of a system to predefined inputs which are not necessarily quantified; it describes the *role* played by a product in a larger system. The behavioral perspective takes the product being designed to be a "black box" whose internal function is not visible (or even known); the inputs, outputs, and operational environment of the product, on the other hand, are "transparent" (see Figure 3.2b). Insofar as behavior describes the response of a product, it is seen as answering the question *"What does the product do?"* Behavior is described without commitment to the form of the product.

Function, on the other hand, is a description of *how* a product works rather than what it does (Figure 3.2a), where the environment is now opaque, and the product is "transparent," and is composed of a series of black box subsystems whose interaction describes how the product comes to exhibit a certain behavior, but without necessarily making commitments about product form.

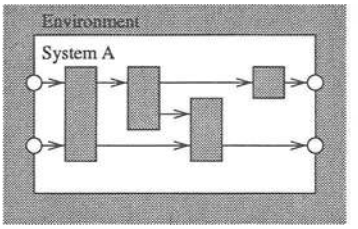

Figure 3.1a Example of the functional perspective.

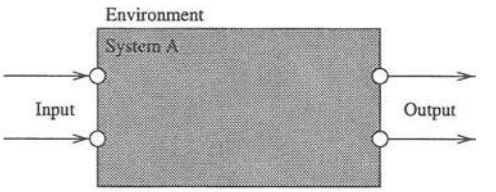

Figure 3.1b Example of the behavioral perspective.

These perspectives are meant to demark the different kinds of tasks that can be performed with functional/behavioral information. At the behavioral level, design is systems-based, concerned with identifying functional requirements to be met by a product. At the functional level, on the other hand, design consists of a configuring components and their interrelationships. So-called "top-down" design proceeds by alternating between behavioral and functional perspectives at ever increasing levels of details. Identifying these perspectives is used below to demonstrate that behavioral and functional descriptions are essentially the same; they are just viewed in different contexts.

It is noted here that the definitions of function and behavior adopted by the other are the opposite of those commonly used in the literature. While the need for terminological consistency is aknowledged, the author believes his definitions are more in keeping with those conventionally implied by practicing engineers. In any event, it matters little in the long run since the distinction between function and behavior is shown (below) to be an artificial one only.

In order to explore this matter further, consider the following three statements:

1. The refrigerator keeps food cold.
2. The refrigerator keeps things cold.
3. The refrigerator preserves food.

Any of these statements in isolation can be considered a *behavior* of a refrigerator. If statement 1 is considered a behavior, then we may ask *How is this behavior achieved?* The answer involves the functions of a refrigerator (isolating a region of space, transferring heat from that space by some means, etc.). However, we may also ask *Why does the refrigerator keep food cold?* One answer to this question is statement 3. Now, considering statements 1 and 3 together, statement 1 is a *function* rather than a behavior. Thus ,whether the statements are taken as functional or behavioral, *context* plays a crucial role

in (a) providing terminological information about the words appearing in the statements, and (b) implying information about the operating environment. For example, in statement 1, the terms "cold" and "keeps" are relative to the context of refrigerators.

Function and behavior are thus *relative to the reference frame of an agent* making assertions about a product; that is, they are not intrinsic properties of designed products. Nonetheless, functional and behavioral information about a product is very important, especially during the product's design. Therefore it is essential that it be representable in the author's system.

It is often possible to represent both functions and behaviors in single natural language clauses that seem quite intuitive to humans (e.g. "...to support a load in bending..."); both behavior ("to support a load") and function ("in bending") are intimately connected in a single phrase. The fact that both the behavior and function can be described in a single natural language statement only obscures their distinction. This constitutes, in the author's opinion, a significant problem with the use of natural language, or any other *in*formal language, to precisely define the nature of designed products. Natural language is used herein only for expository purposes; the author intends this research to lead eventually to a more formal specification of functional/behavioral information.

In order to avoid this confusion while maintaining a sense of connection between them, the author uses the term *predicative description* to include both function and behavior descriptions. This term captures the sense of activity, as well as the complexity of the concepts.

The basic relation that connects function and behavior is the "how/why" relation: given a function, the *why* relation describes its behavior; and given a behavior, the *how* relation describes the function that results in it. The how/why relations are disjoint and intransitive with other relations, particularly with respect specialization. In figure 3.2 four predicative statements about a refrigerator are given, and both the how/why and specialization/generalization relations are shown. It may be argued that the *why* relation is a kind of generalization: in comparing statements 1 and 3 in the figure, it is sensible to think of "preserving food" as a generalization of "keeping food cold". But this is a generalization based on the intent the statement as a whole, rather than one associated with the *components* of the statements. A similar argument can be made about the specialization and *how* relations. The key differences between function and behavior in the author's work can be summarized as in the table in Table 3.1.

Predicative descriptions are *complex* in that their expression tends to be formed as *predicate* clauses consisting of verb/object pairs (VOPs). That is, they describe actions performed by an entity upon some other entity, independent of the phrasing in natural language.

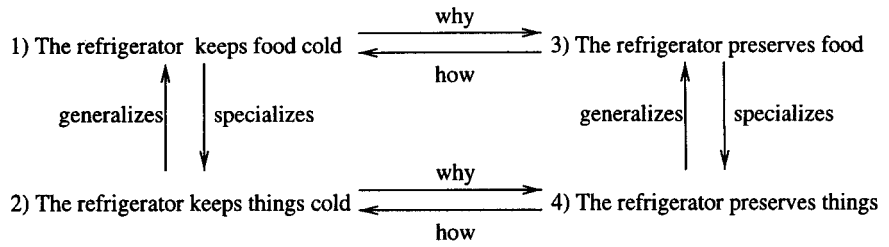

Figure 3.2 Example of abstraction of predicative descriptions. Numbered statements are assertions about a product. Labeled arrows indicate predicative and taxonomic relations between assertions.

This apparent verb/object structure of both functions and behaviors has been used as the root of various formalizations, such as in (Umeda et al., 1996); the current author also employs this approach. Consider again the statements in Figure 3.2: statements 1 and 3 are related through how/why relations. Statement 2 is related to statement 1 by generalization on the object of the VOP. A similar generalization carried out on statement 2 (yielding statement 4) is virtually meaningless. The author believes that generalizations will not generally transfer through how/why relations; in other words, abstraction relations and predicative relations are not transitive. Also, that the abstraction occurred only on the object part of the VOP suggests that in the general case, abstraction can occur on either the verb or the object parts independently.

Table 3.1 Differences between Function and Behavior

Behavior	**Function**
role-dependent	operational
goal-oriented	process-oriented
what a system does	how a system does it
based on purpose/usage	based on physical properties

The key to abstraction of predicative relations is the verb part of the VOPs. But since a VOP can be *both* a functional and a behavioral description (depending on the context), any generalization rules for the verb parts of VOPs must be based on the definition of the verb term itself rather than on its functional or behavioral connotations.

Finally, these definitions of function and behavior deal only with the reactions expected of products for given sets of inputs; that is, no notion of *intended*, or designed-in, function or behavior, or of *purpose* is implied.

3.3 COMMITMENTS DUE TO MEREOLOGY

Mereology is a branch of logic that uses part-whole relationships to describe entities. For example, in a conventional ontology, one might model an automobile as having the property of having four wheels; in a mereological perspective, a relation is defined between the automobile and the wheels themselves. Obviously, in engineering design, both the conventional and the mereological approaches are relevant. Surprisingly, little work appears in the recent literature outside Europe on mereology in AI, and there is almost no work on mereology as such in the engineering literature.

The basic problem of mereology as a field of study is that there appear to be various, often inconsistent, semantics associated with the term "part of". For example, consider the following three statements.

1. A piston is a part of an engine.
2. An engine is a part of an automobile.
3. An automobile is a part of a fleet.

Each statement, on its own, is perfectly reasonable. Furthermore, from the first two statements, we can reasonably deduce that *"A piston is a part of an automobile."* But from all three statements, can we reasonably deduce that *"A piston is a part of a fleet"*? The problem is that there are two different meanings of the term "part of," and that transitivity is not preserved between them. Mereology's main concern is establishing an overall structure to reason reliably with all the possible part-whole relationships.

There appear to be two schools of thought regarding the treatment of mereology. One school advocates a single, universal, and transitive part-of relation, based on the assumption that all distinctions about types of parts are really conceptualizations and are not rooted in reality. In order to address the paradoxes that result, first-order predicate calculus is used to introduce sufficient predicates to distinguish between kinds things. This approach is taken by the developers of Ontolingua and KIF (Gruber, 1992) and the logics of Lesniewski (Srzednicki et al., 1984).

The other school of thought contends the cognitive distinctions must be represented; in other words, a proper mereology must handle the transitivity problem directly by admitting distinctions between different part-of relations. This approach is supported by the work of Artale et al. (Artale et al., 1996b; Artale et al., 1996a) and Simons (Simons, 1987). In this approach, different part-of relations are explicitly defined to handle different conceptualizations (e.g. assembly/component versus space/region), and transitivity is not preserved across them. Also, the part-of relation is seen as complex, rather than primitive, which requires the development of specialized logics that integrate mereology with topology and morphology as in, for example, (Borgo et al., 1996).

A fundamental problem with this approach is that there is no way to enumerate all the different "primitive" part-of relations. For example, in (Winston et al., 1987) six primitive part-of relations are defined; they are summarized in Table 3.2. It has been shown ((Artale et al., 1996b)) that (a) it is impossible to decide if these constitute a complete set of part-of primitive relations, and (b) some of these relations (such as stuff/object) are more linguistic artifacts than actual cognitive or other constructs of knowledge.

Table 3.2 Summary of part-of relations in (Winston et al., 1987).

RELATION	EXAMPLE
component/integral-object	"wheels are parts of cars"
member/collection	"a product is part of a batch"
portion/mass	"3 ft. of stock rod"
stuff/object	"a car is partly aluminum"
feature/activity	"grasping is part of carrying"
place/area	"the front of the car"

Rather than siding with one school or the other, the author proposes a new mereological framework, wherein the part-of relation is a well-defined function mapping triplets of arguments to the boolean values. That is:

$$P(p, W, \pi) \Rightarrow \{\mathcal{T}, \mathcal{F}\} \tag{3.1}$$

where p is a part, W is a whole, and π is a property or properties used by the P part-of function.

This approach is based on the observation that parthood is related to some sort of *overlapping* between the values of at least one property of a part on the values of the same properties of a whole. For example, to establish a part-of relation for regions of a space, π is the set of properties defining the size and position of a spatial region. P, then, asserts that p is wholly contained by W if its volume is contained in W's volume.

This approach addresses a variety of open issues. First, by "deferring" uniqueness of different part-of relations to the properties π, P itself remains a single universal, ternary predicate, which is logically elegant. Second, primitive part-of relations can be defined as those whose properties π are fundamental in AIM-D (e.g. properties of length, mass, time, etc.); complex part-of relations are constructed by composing primitive ones; this suggests an abstraction hierarchy of part-of relations which would be useful for automated reasoning

processes, such as case-based reasoning and decision support. Third, transitivity is preserved to the degree that different properties are used in different part-of relations.

This last point deserves some explanation. Transitivity is preserved entirely in reasoning processes where different instances of P use the same properties π. On the other hand, different instances of P that use properties that have no commonality in the abstraction hierarchy are not transitive at all. These two cases correspond to the typical behavior of other approaches. For example, transitivity is preserved over different instances of the group/member relation, but not between a group/member relation and an assembly/component relation.

However, the author's approach allows *partial* transitivity to be recognized. In the example at the beginning of this section, it was shown that one may reason that a piston is a part of a rental fleet of automobiles if there is only one part-of relation. While there is clearly something wrong with such a conclusion for most conventional uses, there is still a certain sense in which it is reasonable. The author believes that this "partial" sense of the conclusion results from the partial subsumption of the properties with which part-of is used in the example. Being able to represent this kind of partial parthood opens the possibility of substantially different reasoning processes that can be automated in a KBS, and should allow for a richer representation of design knowledge.

Specific mereological axioms using the formalism presented above are currently under development for the next "version" of AIM-D. The current version of AIM-D (Salustri, 1996) contains only an implied notion of mereology as captured by the four levels of product composition defined therein. That is, AIM-D has specific axioms for the construction of assemblies from parts, parts from features, and features from quantities. Equation 3.2 gives an example: the axiom relating parts and assemblies. It states that a part p consists of features f that are in the set of all features F, and that satisfy a predicate ϕ, which is taken to be any possible mereological relation between features f.

$$\exists p \left[\forall f \left[(f \in p) \equiv (f \in F) \bullet \phi(f)\right]\right] \tag{3.2}$$

Each axiom implies a different parthood relation between the whole (e.g. assemblies) and its parts (e.g. parts). The next version of AIM-D will have a more explicit formulation of parthood relations based on the material presented herein.

3.4 COMMITMENTS FOR CONTEXT-SENSITIVITY

Engineering terms can often have different meanings depending on the various *contexts* in which they are used. A context is essentially a mapping between terms and denotations. An assertion may be found true in one context but false

in another. Contexts may also include special rules for carrying out those reasoning processes. For example, consider: *"the block deck height of the engine includes the thickness of the engine gasket."* If the designer in charge of the engine block believes this assertion, but the designer in charge of the cylinder head does not believe it, the designers will disagree about the answer to the question *"What is the total height of the engine?"*; the two designers are working in two slightly different contexts. Contexts have various uses ranging from encapsulation of parts of a KB, to providing a shorthand notation for omitting common arguments (such as location, time, etc.) and separating meta levels of languages (Sowa, 1992). Contexts are currently a topic of significant interest, especially in the KR community, where the issues raised in their treatment impacts on distributed computing and AI. Some of the possible uses of contexts include (Sowa, 1992):

- partitioning a knowledge base into more manageable modules;
- encapsulating parts of a knowledge base, as in so-called object-oriented systems;
- providing a shorthand for omitting common arguments, such as location, time, etc.;
- providing a way to resolve indexical referents, such as "this", "I", and definite noun phrases beginning with "the;"
- representing environments whose modality, level of certainty, or hypothetical existence is different from that of other environments;
- supporting propositional attitude verbs, such as "believe;" and
- separating a meta level of language that is used to talk about the language in a nested context.

There are also several ongoing efforts to formalize notions of context. Akman and Surav (Akman and Surav, 1996) give an excellent overview of the various approaches.

Contexts are clearly relevant to product modeling, since typically many designers are involved in concurrently developing a single product. In this kind of environment, an unintentional contextual difference can lead to disastrous results.

The author is currently working to incorporate contexts into AIM-D. The general approach is most similar to that of McCarthy and Buvac (McCarthy and Buvac, 1994), wherein contexts are essentially namespaces binding terms to semantics. Contexts can be nested, and various predicates are provided to test the truth value of a statement in a particular context, and to "lift" terms commonly defined in different contexts to higher, more universal contexts.

One very important aspect of contexts in AIM-D regards the terms used to name entity attributes. This is best illustrated with a simple example.

Consider the statements: (a) *"The color of the car is green,"* and (b) *"Green is a color."* Linguistic idiosyncrasies aside, the term *color* is used in two distinct, but related, ways: as the name of an entity attribute, and as a generalized relation entity. It is possible to capture the relation between the various uses of a term without recourse to linguistic constructs.

In AIM-D, the names of entity attributes are the names of relations between entities, and are themselves terms defined in some context. This means that a term such as *color* must be used consistently throughout a given context where it is defined, and in all its sub-contexts. This is different from the approach taken in object-oriented modeling, where the semantics of an instance variable are consistent only across the instances of a given class. By making the universality of attribute name definitions explicit, (semi-)automated reasoning about attributes, and the relations they represent, is now possible.

Contexts also matter in terms of managing mereological relations. For example, (Gerstl and Pribbenow, 1996) suggest that a primary characteristic of item that leads to different mereological relations is whether the item is homogeneous (having no parts), uniform (consisting of like parts only), or heterogeneous (having various different parts). However, it depends on the context of a particular task how a particular item will be regarded. For example, in the context of engine assembly, an aluminum part may be regarded as a homogeneous item; but from a context of materials engineering, aluminum is at least uniform if not heterogeneous. Clearly, the interactions between context and mereology still need further exploration.

3.5 SUMMARY

Ontological commitments can be regarded as decisions about the interpretation of statements in a given language. A variety of commitments have been presented in this section that pertain to the description of designed products. Clearly, significant work remains to be done; there are many other commitments about designed products that can be found or deduced from other research efforts. However, to help ensure rigor, the only commitments currently part of AIM-D are those that apparently allow a consistent logic to exist.

4. IMPLEMENTATION OF DKSL

In this section, the design and implementation of DKSL is discussed, including how the ontological commitments made thus far have been, or are being, embedded within it. Generally, the commitments amount rules that DKSL must satisfy in order to preserve logical rigor: no model should be representable in DKSL if the model violates the ontological commitments.

For example, AIM-D defines sub-assemblies in terms of subsets of the set of all parts of a product, rather than as parts of those products. This implies that DKSL must be able to distinguish automatically between parts, sub-assemblies, and product assemblies. Furthermore, DKSL has restrictions on how assemblies are formed through the merging of defined sub-assemblies.

The author has not yet investigated the changes that would have to occur in DKSL if different ontological commitments were made; this issue remains an open one for future research.

4.1 UNDERLYING KNOWLEDGE REPRESENTATION

The current implementation of DKSL is as a small, stand-alone program with a text-based user interface. It is a "concept-proving" implementation, to allow the author to study internal structures and algorithms needed to represent and manipulate design information effectively.

SCM, an implementation of the Scheme programming language by Aubrey Jaffer, is currently used to implement DKSL. SCM is small, robust, and true to the IEEE standard for Scheme (IEEE, 1991); it also has a number of extensions that facilitate rapid software prototyping (e.g. POSIX-compliant file I/O operations).

DKSL is implemented using a *frame-based* KR scheme. Frame systems are similar to object-based systems, but introduce a finer level of representation. Object systems are based on object-attribute-value triplets: objects contain attributes which have values. Frames, on the other hand, use frame-slot-facet-value quadruplets. Slots are composed of possibly many facets, which allows a richer representation of attributes. Furthermore, functions called *procedural attachments* can be associated with slot facets. These functions may be triggered automatically or at a user's request to carry out various management tasks such as constraint checking and inverse relation maintenance. A procedural attachment that fires automatically is called a *demon*. Generally, demons execute in three cases: when a new value is (a) added to or (b) removed from a slot, and (c) when a slot's value needs to be calculated rather than retrieved.

Frame systems as described above are common in KR systems such as CLASSIC (Brachman et al., 1991). However, the system implemented in DKSL is substantially different from these others. These differences arise from the ontological commitments, and are presented here.

Context-sensitivity. Conventional frame systems make no particular commitments about the contexts in which terms are resolved into frames. However, DKSL supports the notion of a context as a "dictionary" mapping terms to frames, implementing the ontological commitments regarding context sensitivity (Section 3.4). Contexts in DKSL may be created by the user, and may be

nested. A System Context contains basic definitions needed by DKSL, and a User Context, which is a sub-context of the System Context stores user-defined frames. Other application-specific contexts are under development. Lifting of terms (per (McCarthy and Buvac, 1994)) is achieved by a simple comparison of different frames with the same name in different contexts. Furthermore, a slot is viewed as a relation having a uniform semantics over a whole context.

No Classes. There are no explicit classes or "meta-frames" in DKSL. Rather, a prototype-based approach is used, wherein any entity can be an *exemplar* with which other frames can be *cloned.* The use of prototypes is consistent with the ontological commitments made in Section 3.1: AIM-D entity types are not explicit, and neither are those in DKSL.

Inheritance through specialization. Without classes, a different kind of inheritance mechanism is needed. *Specialization* of individual frames is used: an exemplar generalizes its clones. Specialization information is used only during frame construction; no specialization information is kept in frames. The most important reason for this is that it allows an exemplar and its clones to change with time without requiring complex change management to preserve that relation beyond a frame's construction.

Calculated type compatibility. Type similarity between frames is calculated as needed. Two frames are type-compatible if there are some slots in one frame with the same names as some slots in the other frame. Since slot names have uniform semantics in a given context, no checks are needed once a value is added to a slot. A frame is a *specialization* of another if the one has at least as many slots as the other, and if every slot in the other has a type-compatible correspondent in the one. Generalization is just the converse of specialization.

Although this clearly imposes a heavier computational load than conventional type systems, it also increases the expressiveness of the language to represent *varying degrees* of type similarity. For example, it is possible to determine if one frame *could be* a specialization of another. This allows the system to "guess" type compatibility of frames, which could be very useful for exploratory algorithms such as case-based reasoning. Also, it is possible to develop automatically *normalized* abstraction hierarchies of arbitrary collections of frames. This raises the possibility of transmitting KBs between systems or agents in forms that are reliably re-constructible in different environments. This kind of type-compatibility is consistent with the ontological commitments in AIM-D.

Location and Names of Demons. DKSL supports two kinds of `if-added` demons, called `pre/put` and `post/put` demons. The `pre/put` demons are predicates that check the validity of the new values before they are added to slots. If a `pre/put` demon fails, either a warning message is displayed and the assignment continues, or an error is triggered and the assignment does not occur; which of these actions occurs depends on whether the demon is *hard* or

soft (discussed below). The `post/put` demons are run after new information is inserted into a KB; these demons perform conventional management tasks such as maintenance of inverse relationships.

The location of demons is standardized in DKSL, and takes advantage of the uniform semantics rule for slots. This allows a well-defined, yet extensible technique for searching, storing, and checking the validity of data, as well as maintaining interrelationships between the data. Demons may be stored in the frame that defines the semantics of a slot, in which case they will be triggered wherever that slot is used; they may alternatively be stored in facets of a particular instance of a slot, in which case they are only triggered for that slot instance.

Hard and soft constraints. Demons can implement constraints on the knowledge stored in the system. These constraints can be either *hard*, violation of which causes an error to be triggered, or *soft*, in which case a warning message is displayed for the user only. This distinction accounts for the different implications of a constraint violation with regards to KB integrity. Hard constraints preserve the basic integrity of the KB, whereas soft constraints indicate an inconsistency in the product being modeled. Since design can be regarded as driven by the need to eliminate such inconsistencies, it is important to represent them differently than those arising from the KBS itself.

Partial meta-information. DKSL also allows some kinds of *meta*-level information about entities to be stored. For example, one constrain the number of values that a slot can have, and the types of values that a slot can contain by associating exemplars representative of those types with particular facets. This kind of information can be used by `pre/put` and `post/put` demons to perform a variety of checks and other operations automatically.

There is other useful meta-information that could be stored, but one must be careful when adding meta-level information to a KBS: one may create *terminological cycles* and other semantic artifacts for which even simple computations are intractable. Work on DKSL is intentionally proceeding slowly in this regard, so that sufficient care is taken to avoid these problems.

5. DKSL FOR ENGINEERING DESIGN

An example of a simple DKSL product model is introduced to demonstrate its applicability. The example involves the structural and steady-state thermal modeling of a wall. It is based on one presented in (Eastman et al., 1991a), but is lacking in some details. Its use will facilitate a comparison of DKSL to various other modeling schemes discussed in (Eastman et al., 1991a).

We begin with an overview of the model's structure; this will be followed by a description of some of the frames needed to describe the model in DKSL, and some of the operations possible on that model.

5.1 STRUCTURAL MODELING CONSIDERATIONS

The reader is referred to Figure 3.3, which shows schematically a wall, labeled with the major structural components as described below.

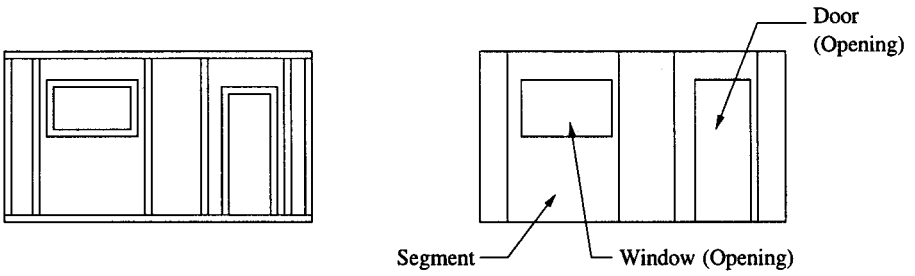

Figure 3.3 Structure of a wall.

In order to model a wall's structure and steady-state thermal behavior, both the geometry and composition of the wall must be considered. Since the example only models an isolated wall, the width and height of the wall are arbitrarily defined values. Had the model included many attached walls, width and height would have been determined by the overall structures of which they were a part. Wall thickness, however, depends on the wall's composition. A wall is composed of various *layers*, each serving a specific purpose – load bearing, insulation, covering, and so on. Each layer consists of a single material, is of constant thickness, and contributes to the overall thickness of the wall.

Complex wall shapes can be described as compositions of area-wise *segments*; a small collection of regular planar shapes can be combined to produce quite complex geometries. Each segment is composed of layers, and all layers in a segment have the same surface area.

Finally, a wall may contain various kinds of *openings* or passages. In this model, openings lie within single wall segments and must pass entirely through the wall. For simplicity, only windows and doors are considered. An opening is a kind of segment: it occupies an area and may be composed of many layers (e.g. multi-paned windows); however, openings are not allowed to contain other openings.

5.2 STEADY-STATE THERMAL MODELING CONSIDERATIONS

The following mathematical model of the steady-state heat behavior of a wall is assumed. The physical relationships are drawn from (Eastman et al., 1991a) and a standard thermodynamics text (Reynolds and Perkins, 1977).

The steady-state heat flow through a wall is given approximately by the following equations, where ΔT is the change in temperature through the wall,

t_w and A_w are the thickness and area of the wall respectively, and k_w is the overall of coefficient thermal conductivity of the wall (calculated by analogy with electrical systems). The thermal resistance of the ith layer of the wall is by r_i.

$$\dot{Q}_w = \frac{k_w A_w}{t_w} \Delta T, \quad k_w = \frac{1}{\sum_i r_i}, \quad r_i = \frac{t_i}{k_i}. \tag{3.3}$$

5.3 DEFINITION OF WALL EXEMPLAR OBJECTS

Given this conceptual model, DKSL frames can now be constructed to represent it. The frames defined here are simplified versions, intended to focus only on the particular example being presented. In a "real" implementation, many more slots and facets would be defined. All quantities are in SI units.

First, we establish a context for this application; this is shown in Figure 3.4. The `simple-walls` context is defined as a sub-context of contexts for 2.5D geometry, SI units for physical systems, and physical assemblies. Setting the context to `simple-walls` ensures that all subsequent assertions are made in that context; this information becomes persistent so that future queries to the information defined in this context will be evaluated within it. (It is assumed that a user intends for knowledge to be used always in the same context, unless that knowledge is intentionally "lifted" into other contexts.) Setting the value of `coordinate-type` establishes a term in the new context that will be used by other frames related to geometry to determine the kind of coordinates to be used in this context.

```
(define-context simple-walls
  (2.5d-geometry
   si-physical-units
   physical-assemblies))
(set-context!  simple-walls)
(set!  coordinate-type
       cartesian-coordinate-system)
```

Figure 3.4 A context for the example wall model.

A `material` exemplar is defined in Figure 3.5; for brevity, only one necessary property, thermal conductivity, and one instance is included. Thermal conductivity data is from (Reynolds and Perkins, 1977).

Wall layers are modeled in Figure 3.6.

A derived attribute (one whose value is calculated from other attributes) for thermal resistance is also included. The `region` exemplar, defined in

```
(define-frame material
  (new (the-frame)
       (thermal-conductivity
        (watt/meter-degree 1.0))))
(define-frame glass,window
  (new (material)
       (thermal-conductivity
        (watt/meter-degree 0.78))))
```

Figure 3.5 Material exemplar and instances.

the `physical-attributes` context, indicates that `layer` can take part in region/space mereological relationships.

```
(define-frame layer
  (new (region)
       (material (new material))
       (thickness (centimeter 1.0))
       (thermal-resistance
         (derived-from (thickness material)
           (/ thickness
              (material 'thermal-conductivity))))))
```

Figure 3.6 Layer objects.

A `wall-atom` frame is defined to model the kinds of properties common to both openings and wall segments (see Figure 3.7). It too is a kind of region. The `(all layer)` construct returns all slot values in the current frame that are of type `layer`. The `thermal-conductivity` slot models the overall coefficient of thermal conductivity per unit area, calculated according to the mathematical model in Section 5.2.

Now `wall-atom` can be used to define exemplars for openings and segments (Figure 3.8). The *specific heat flow* of an opening is the rate of heat flow per degree of temperature difference at steady-state. Also, a distinction is made between the solid wall area of the segment and the area of any openings in the segment. The specific heat flow of a segment is the sum of the specific heat flows of the openings and of the rest of the segment. Finally, the `heat-flow` slot models the heat flow through a wall segment for a given temperature difference (`delta-t`).

No geometric (shape) information has been embedded in `segment` and `opening`; shape information will be provided when specific segments and

```
(define-frame wall-atom
  (new (region)
       (thickness
         (derived-from ((layers (all layer)))
           (apply + (for ((l layers))
                         (l 'thickness)))))
       (thermal-conductivity
         (derived-from ((layers (all layer)))
           (/ 1.0
              (apply +
                (for ((l layers))
                  (l 'thermal-resistance)))))))))
```

Figure 3.7 Atomic wall components for openings and segments.

openings are created (see below). This does not, however, prevent us from referencing attributes such as `area` when defining frames so long as they are defined by the time they are used.

The `opening` frame is specialized for doors and windows (Figure 3.9). The `door` exemplar specializes both `opening` (for composition and thermal analysis) and `rectangle` (for geometric characteristics). It also specializes `part`, provided by the `physical-assemblies` context, to signify that it can enter into a part/assembly relation (which is different from a region/space relation). The `rectangle` exemplar and other geometric information is defined in the `2.5d-geometry` context.

The `window` exemplar is further specialized into one- and two-paned windows. Finally, two specific kinds of windows are created: a single-paned window with a 5 millimeter pane of glass, and a double-paned window with two 5 millimeter panes separated by a 4 millimeter air gap. Note that shape has not yet been assigned to the window objects. The `cardinality` facet is used to limit the number of panes in each kind of window.

The last exemplar, for the wall itself, just gathers `segments` (see Figure 3.10), since all the important functions for thermal analysis have been defined elsewhere. The `heat-flow` slot calculates the total heat flow through a wall for a given temperature difference. The `wall` enters into both part/whole and region/space relations.

This completes the DKSL model. It is not a model of a *particular* wall, but a template from which various models can be built. The model is intended to be as general as possible, so that it may be used for other purposes that just steady-state thermal analysis. It would have been substantially simpler had we targetted it specifically and exclusively for thermal analysis, but it would also have been of very limited use.

```
(define-frame opening
  (new (wall-atom)
       (specific-heat-flow
         (derived-from (thermal-conductivity area)
           (* thermal-conductivity area)))))
(define-frame segment
  (new (wall-atom)
       (opening-area
         (derived-from ((openings (all opening)))
           (apply + (for ((o openings))
                      (o 'area)))))
       (segment-area
         (derived-from (area opening-area)
           (- area opening-area)))
       (specific-heat-flow
         (derived-from ((openings (all opening))
                        thermal-conductivity
                        segment-area)
           (+ (apply +
                (for ((o openings))
                  (o 'specific-heat-flow)))
              (* thermal-conductivity
                 segment-area))))
       (heat-flow
         (derived-from (specific-heat-flow)
           (using (delta-t))
           (* specific-heat-flow delta-t)))))
```

Figure 3.8 Exemplar for wall openings and segments.

```
(define-frame door
  (new (part opening rectangle)
       (layer (new layer))))
(define-frame pane
  (new (part layer)
       (material glass,window)))
(define-frame window
  (new (part opening)
       (pane (new pane))))
(define-frame window,1pane
  (new (window)
       (pane cardinality 1)))
(define-frame window,2pane
  (new (window)
       (pane cardinality 2)
       (gap (new (layer) (material air)))))
(define-frame window,1pane,5
  (new (window,1pane)
       (pane (thickness (centimeter 0.5)))))
(define-frame window,2pane,5-4-5
  (new (window,2pane)
       (pane (new (layer)
                  (thickness (centimeter 0.5)))
             (new (layer)
                  (thickness (centimeter 0.5))))
       (gap (thickness (centimeter 0.4)))))
```

Figure 3.9 Exemplars for doors and windows.

```
(define-frame wall
  (new (part region)
       (segment (new (segment)))
       (area
         (derived-from ((segments
                          (all parts segment)))
           (apply + (for ((s segments))
                      (s 'area)))))
       (opening-area
         (derived-from ((segments
                          (all parts segment)))
           (apply + (for ((s segments))
                      (s 'opening-area)))))
       (heat-flow
         (derived-from ((segments
                          (all parts segment)))
           (using (delta-t))
           (* (apply +
                (for ((s segments))
                  (s 'specific-heat-flow)))
              delta-t)))))
```

Figure 3.10 Exemplar for walls.

The goals for developing the model this way are: (a) for demonstration purposes, to showcase the flexibility of DKSL, and (b) to define a model that is not necessarily restricted to thermal analyses only.

5.4 USAGE OF THE WALL MODEL

This section defines a particular wall and calculates the heat flow through it. The sample wall (Figure 3.11); consists of two segments, a large rectangular segment with a door and a window, and a triangular segment with no openings. Figure 3.12 defines the necessary `segment` and `wall` frames.

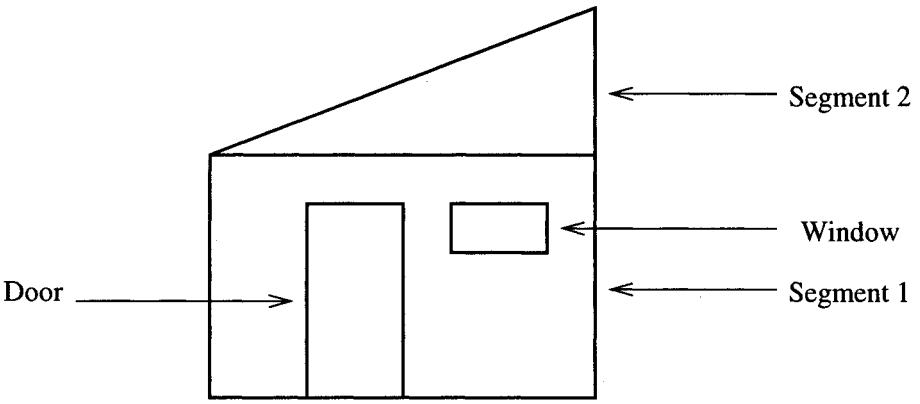

Figure 3.11 Geometry of sample wall.

Figure 3.13 shows three messages sent to the wall `w`, and the values that are returned. The third message returns the heat flow through the wall for a temperature difference of 20 degrees.

The specific wall model defined here contains all the information needed for a preliminary thermal analysis, even though the geometric relationships between the components have not yet been specified. For example, the door and window have not been positioned in `segment1`; nor have the segments been positioned with respect to each other. The operations to do this are shown in Figure 3.14. The forms in the figure take advantage of methods defined in the `2.5d-geometry` context that translate and scale objects.

Topological relations, such as the physical connection between wall segments, are not yet supported by DKSL. These kind of relations are currently a topic of study of the author.

Let us say that too much heat was being lost through the wall as defined above. One alternative is to change the single-paned window to a double-paned window. This is shown in Figure 3.15. We find that a double-paned window improves the overall thermal insulation of the wall.

```
(define-frame segment1
  (new (segment rectangle)
       (width (meter 4.0))
       (height (meter 2.5))
       (door (new (door)
                  (layer (material wood,pine,white)
                         (thickness (meter 0.06)))
                  (width (meter 1.0))
                  (height (meter 2.0))))
       (window (new (window,1pane,5 rectangle)
                    (width (meter 1.0))
                    (height (meter 0.5))))
       (outer (new (layer)
                   (material brick,face)
                   (thickness (meter 0.1))))
       (core (new (layer)
                  (material wool,rock)
                  (thickness (meter 0.1))))
       (inner (new (layer)
                   (material plaster,gypsum)
                   (thickness (meter 0.01))))))
(define-frame segment2
  (new (segment triangle)
       (width (meter 4.0))
       (height (meter 1.5))
       (outer-face (new (layer)
                        (material wood,pine,white)
                        (thickness (meter 0.005))))
       (outer (new (layer)
                   (material brick,common)
                   (thickness (meter 0.1))))
       (core (new (layer)
                  (material wool,rock)
                  (thickness (meter 0.1))))
       (inner (new (layer)
                   (material wood,pine,yellow)
                   (thickness (meter 0.005))))))
(define-frame w
  (new (wall)
       (segments segment1 segment2)))
```

Figure 3.12 Synthesis of sample wall.

```
(w 'area)              ⇒ 13
(w 'opening-area)      ⇒ 2.5
(w 'heat-flow 20)      ⇒ 1712.554
```

Figure 3.13 Queries and actions for the sample wall.

```
;;; position window and door within segment 1.
(((w 's1) 'wdw) translate 0.75 1.5 0)
(((w 's1) 'door) translate 2 0 0)
;;; position segment 2 with respect
;;; to segment 1 in the wall.
(((w 's2) 'scale 0 -1 0) 'translate 4 2.5 0)
```

Figure 3.14 Positioning objects in the sample wall.

```
((w 's1) 'wdw (combine ((w 's1) 'wdw)
                       Wdw,2Pane,5-4-5))
(w 'heat-flow 20)      ⇒ 213.065
```

Figure 3.15 Altering the window.

The `combine` form creates a new frame that is a clone of its first argument (the window in `segment1`) with any extra slots in its other arguments (`wdw, 2pane, 5-4-5` in this case). The resulting window has the same position and orientation as the original window, but will have two panes instead of one.

We close this section with two examples of *assertions* (see Figure 3.16), which can be regarded either as constraints on data, or as the statement of *facts* from which reasoning may then proceed. The first assertion may be read as "*For all segments in wall* `w`, *the heat flow through the segment for a temperature difference of 20 degrees is less than 300*". Similarly, the second example may be read as "*There exists at least one door in wall* `w`". The `forall` and `exists` constructs correspond to the two logical quantifiers in propositional logic.

5.5 EVALUATION OF DKSL

Only a preliminary evaluation of DKSL is possible at this time, since it is still under development. However, it appears that DKSL has the potential to compare favorably to other approaches.

```
(forall s (all w segment)
  (< (s 'heat-flow 20) 300))
(exists p (all w part)
  (is-a?  p door))
```

Figure 3.16 Two examples of assertions.

Table 3.3 Evaluation criteria of Eastman and Fereshetian

full abstract data types	multiple specializations
composite objects	relations within compositions
relations on object structure	relations between variables
variant relations defined operationally	variant relations
external applications integrity mgmt	management of partial integrity
schema evolution	refinement versus classes/instances

Currently, the author uses the criteria established by Eastman and Fereshetian (Eastman and Fereshetian, 1994) to evaluate DKSL; they are summarized in Table 3.3. These criteria were established to compare data models rather than KR schemes, so some mismatches are expected. In particular, the *external applications integrity management* criterion is inappropriate for DKSL. Also, there are other efforts to establish criteria (such as Ward's criteria of precision, density, and naturalness (Ward, 1992)) that are worth pursuing. The author is currently working towards a set of criteria for KR systems that incorporate these efforts and others.

Abstract data types are supported in that (a) frame interfaces are implementation independent, (b) the type comparison predicates operate uniformly on all frames, and (c) frame behavior is specified via methods. *Multiple specializations* (without name-clash resolution) and *composite frames* are obviously supported. Since compositions are inherent to DKSL, relations both *within compositions* and *on object structure* are supported. *Relations between variables* can also be represented, but due to the strictly representational nature of DKSL, the dynamics of manipulating those relations are not treated. Nonetheless, a user is free to develop Scheme procedures that use the relational information for any purpose.

DKSL is deficient with respect to the last three criteria: *variant relations*, *schema evolution*, and *management of partial integrity*. Since DKSL is in-

tended to be representational only, it does not directly support the *active*, dynamic management of the information it represents. In real design situations, the management of change, of which these three criteria are all examples, is essential. But before these issues can be addressed, the author must generate suitable theory to supplement the current formal underpinnings of DKSL; this is a point of on-going research.

6. FUTURE WORK

There are many aspects of DKSL and its underlying ontological commitments that remain to be explored. This section mentions a few of the more important ones.

Only the ontological commitments made by AIM-D are currently supported by a formalization. Other commitments mentioned in this paper are currently the subject of formalization efforts by the author. It is unclear that DKSL will not require alteration as those formalizations are achieved.

DKSL currently only advises the user about violated constraints that do not jeopardize the integrity of the KB. However, it is exactly the need to resolve such constraints that drives design processes. Therefore, some way of representing the status of constraints must be added to DKSL.

Function modeling has not been incorporated into DKSL yet.

Although an algorithm for automatic categorization of DKSL entities has been devised, it is of little usefulness in "real" application domains. Automatic categorization allows a user to develop DKSL models in isolation, yet allows the system to normalize those models with respect to other contexts. This removes the onus from the user to understand fully the potentially extensive libraries that may be available, yet still allows the system to take advantage of them to organize design knowledge. Automatic categorization may also be very relevant to certain reasoning systems, such as case-based reasoning, in that it facilitates comparing user-provided entities to case libraries.

The user interface to DKSL clearly needs to be made substantially more usable by people unfamiliar with Scheme or Lisp. Currently, the author is considering a combined graphical/textual interface written in Java to take advantage of web-based functionality. The use of Java is also expected to bear on the ability to distribute DKSL entities on the Internet. Comparatively little work has been done in this regard so far, but it seems clear that substantial performance and functionality improvements may be possible.

7. CONCLUSIONS

This paper has presented a progress report on the author's efforts to construct a KBS for engineered products that is rooted in logical foundations. Various

ontological commitments have been suggested. These commitments are necessary to establish a fundamental basis for the development of the system.

An overview of the Design Knowledge Specification Language (DKSL) has also been presented. Though still under development, DKSL appears to provide a sufficiently rich representational form for product knowledge can be stored so as to be useful in design, analysis, and even manufacturing areas. Significant work remains to be done to more fully demonstrate the system's adequacy, and to provide an interface that is useful to typical practicing designers. However, the author believes that, based on the contents of the paper, there is reason to be optimistic that these goals can be achieved.

Acknowledgments

The author gratefully acknowledges the National Sciences and Engineering Research Council of Canada for funding this work under grant number OGP0194236.

Notes

1. The name ZF comes from its originators, Zermelo and Fraenkel.

References

Akman, V. and Surav, M. (1996). Steps toward formalizing context. *AI Magazine*, 17(3):55–72.

Alberts, L. K. (1994). Ymir: A sharable ontology for the formal representation of engineering design knowledge. In Gero, J. S. and Tyugu, E., editors, *Formal Design Methods for CAD*, IFIP Transactions, pages 3–32, Amsterdam. North Holland.

Artale, A., Franconi, E., and Guarino, N. (1996a). Open problems with part-whole relations. In *Proceedings of 1996 International Workshop on Description Logics*, pages 70–73, Boston, MA.

Artale, A., Franconi, E., Guarino, N., and Pazzi, L. (1996b). Part-whole relations in object-centered systems: An overview. *Data and Knowledge Engineering*, 20:347–383.

Bernays, P. (1968). *Axiomatic Set Theory*. North-Holland Publishing Company, Amsterdam.

Borgo, S., Guarino, N., and Masolo, C. (1996). A pointless theory of space based on strong connection and congruence. In Aiello, L. C. and Doyle, J., editors, *Principles of Knowledge Representation and Reasoning - KR96*. Morgan Kaufmann.

Brachman, R. J., McGuinness, D. L., Patel-Schneider, P. F., and Resnick, L. (1991). *LIVING WITH CLASSIC: When and How to Use a KL-ONE-Like Language*, chapter 14, pages 401–456. Morgan Kaufmann Series in Representation and Reasoning. Morgan Kaufmann Publishers, Inc., San Mateo.

Chakrabarti, A. (1993). Towards a theory for functional reasoning in design. In Roozenburg, N. F. M., editor, *Proceedings of ICED 93, 9th International Conference on Engineering Design*, volume 1, pages 1–8, Zurich, Switzerland. Heurista.

Chittaro, L., Tasso, C., and Toppano, E. (1994). Putting functional knowledge on firmer ground. *Applied Artificial Intelligence*, 8:239–258.

Copi, I. M. (1979). *Symbolic Logic*. Macmillan.

Damasio, A. R. and Damasio, H. (1992). Brain and language. *Scientific American*, 267(3):89–95.

Eastman, C. M., Bond, A. H., and Chase, S. C. (1991a). Application and evaluation of an engineering data model. *Research in Engineering Design*, 2:185–207.

Eastman, C. M., Bond, A. H., and Chase, S. C. (1991b). A formal approach for product model information. *Research in Engineering Design*, 2:65–80.

Eastman, C. M. and Fereshetian, N. (1994). Information models for use in product design: a comparison. *Computer-Aided Design*, 26(7):551–572.

Genesereth, M. R. and Fikes, R. E. (1992). Knowledge interchange format reference manual, version 3.0. Technical Report Logic-92-1, Computer Science Department, Stanford University, Stanford, California.

Gerstl, P. and Pribbenow, S. (1996). A conceptual theory of part-whole relations and its applications. *Data and Knowledge Engineering*, 20:305–322.

Gruber, T. R. (1992). Ontolingua: A mechanism to support portable ontologies. Technical report, Knowledge Systems Laboratory, Stanford University, Stanford, CA.

Guarino, N., Carrara, M., and Giaretta, P. (1994). Formalizing ontological commitments. In *Proceedings of the 12th National Conference on Artificial Intelligence*, volume 1, pages 560–568, Seattle, WA, USA. AAAI Press.

Hakim, M. M. and James H. Garrett, J. (1993). A description logic approach for representing engineering design standards. *Engineering with Computers*, 9(2):108–124.

Hurst, T. N. (1991). Automated model generation using the kif declarative language. In Gupta, G. and Shoup, T. E., editors, *Proceedings of the 1991 ASME Computers in Engineering Conference*, pages 137–144. ASME, American Society of Mechanical Engineers.

IEEE (1991). Ieee standard for the scheme programming language. IEEE Std 1178-1990. Institute of Electrical and Electronic Engineers.

Jaynes, J. (1976). *The Origin of Consciousness in the Breakdown of the Bicameral Mind*. University of Toronto Press.

Johnson, R. E. and Zweig, J. M. (1991). Delegation in c++. *Journal of Object-Oriented Programming*, 4(7):31–34.

McCarthy, J. and Buvac, S. (1994). Formalizing context (expanded notes). Technical Note STAN-CS-TN-94-13, Computer Science Department, Stanford University, Stanford, CA.

Qian, L. and Gero, S. (1996). Function-behavior-structure paths and their role in analogy-based design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 10:289–312.

Reynolds, W. C. and Perkins, H. C. (1977). *Engineering Thermodynamics*. McGraw-Hill.

Salustri, F. A. (1996). A formal theory for knowledge-based product model representation. In *Knowledge-Intensive CAD II: proceedings of the IFIP WG 5.2 workshop*. Chapman & Hall.

Simons, P. (1987). *Parts, A Study in Ontology*. Clarendon Press, Oxford.

Sowa, J. (1992). Discussions about kif and related issues. Interlingua Mailing List, 20 July.

Srzednicki, J. T. J., Rickey, V. F., and Czelakowski, J., editors (1984). *Lesniewski's Systems: Ontology and Mereology*. Nijhoff International Philosophy Series 13. Martinus Nijhoff Publishers, The Hague.

Sturges, R. H., O'Shaughnessy, K., and Kilani, M. I. (1996). Computational model for conceptual design based on extended function logic. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 10:255–274.

Umeda, Y., Ishii, M., Yoshioka, M., Shimomura, Y., and Tomiyama, T. (1996). Supporting conceptual design based on the function-behavior-state modeler. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 10:275–288.

Ward, A. C. (1992). Some language-based approaches to concurrent engineering. *International Journal of Systems Automation: Research and Applications*, 2(4):335–351.

Winston, M. E., Chaffin, R., and Herrmann, D. (1987). A taxonomy of part-whole relations. *Cognitive Science*, 11:417–444.