(To appear in Proceedings of AAAI Spring Symposium on Ontological Engineering, March 1997)
http://www.cs.utexas.edu/users/pclark/papers

The Neutral Representation Project

Mike Barley, Peter Clark, Keith Williamson, Steve Woods
Boeing Research and Technology
P.O. Box 3707, WA 98124
{barley,clarkp,kew,woods}@redwood.rt.cs.boeing.com

Abstract

The evolving complexity of many modern artifacts, such as aircraft, has led to a serious fragmentation of knowledge among software systems required for their design and manufacture. In the case of aircraft design, views of the same generic design knowledge are redundantly encoded in multiple software systems, each system using its own idiosyncractic ontology, and each system containing that knowledge in an implicit, task- and vendor-specific form. This situation is expensive, due to high cost of developing from scratch, maintaining and keeping synchronized the many systems used in design.

Boeing's "Neutral Representation" project aims to address these concerns by prototyping languages and methods for making these underlying ontologies and knowledge explicit, and hence more sharable and maintainable. We are approaching this goal through three tasks: Building explicit, neutral, machine-sensible representations of design knowledge; structuring that knowledge into reusable components, indexed by the ontologies which they use; and linking those representations with existing design systems. In this paper we present the work done this year, and discuss issues related to ontological engineering and knowledge sharing which have arisen.

Introduction

Overview

The Boeing Company performs very large scale engineering of aerospace products. The 777 commercial aircraft, for example, is comprised of about 3 million parts, some 350,000 of which are significant designed parts. Some of these parts are designed and built in house, some are designed by Boeing and built by suppliers, some are designed and built by suppliers to Boeing specification, and some are standard parts, such as fasteners, which are ordered off the shelf. In any case, all parts must meet their specifications in order to be integrated and assembled. In a manner of speaking, this means that the design of all these component parts conform to an underlying, coherent model of the product.

Although this underlying model is articulated in Boeing Design Manuals, industry standards, software, and documented design processes, this articulation is typically in the form of natural language text or vendor-specific code, rather than in a neutral, machine-sensible form. As a result, knowledge cannot be automatically exchanged or shared between systems, resulting in higher development, validation, and maintenance costs. The inaccessibility of knowledge embedded in engineering software is of special concern. For example, it is typical for there to be separate applications creating the engineering design and tooling design for parts. The applications are often independently developed and maintained, but yet they do (and must) share common knowledge about the part, how it is made, and the assumptions underlying the design. When these assumptions change, these programs will become unsynchronized in a way that is hard to glean directly from their lines of code. The cost of possible desynchronization is that the part might not be producible on the specified kind of tooling - perhaps the bend radius of a sheet metal part is too sharp. Or perhaps the part so produced has to be rejected because there is too much springback.

This problem is growing, as Boeing is developing new design automation software at a significant rate. One class of software, called Knowledge Based Engineering (KBE), software is closely coupled to Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) systems (Proctor 1995). KBE is being used to create large quantities of product data in a repeatable and standard way. It is extremely cost-effective because it captures the mapping between a particular design operation and the sequence of steps needed to drive a CAD system's solid modeler to create the desired digital geometry. However, it also adds yet another encoding of design knowledge to be maintained, and the knowledge which it contains cannot be

easily transferred to or exchanged with other design systems.

There are several ongoing projects within Boeing to try to address these concerns with respect to *product instance data*, so that at least information about specific parts can be represented and exchanged in a neutral, machine-sensible form (for example work on STEP (Chen 1996), which aims to standardize the form in which product data is exchanged). The **Neutral Representation project** similarly is addressing these concerns but with respect to design knowledge, where information is mainly in the form of *general rules*. Our goal is to develop machine-sensible representations of design rules and assumptions, so that *design knowledge* can similarly be made explicit, stored, and exchanged. This includes knowledge such as the assumptions underlying a design, design constraints, knowledge about design and manufacturing features, design intent implicit in geometry data, manufacturing knowledge and geometric knowledge.

The degree to which we are successful will influence how our core design knowledge assets are expressed and utilized. The project should clarify and standardize design vocabularies and methods. It will mean that proven design methods can be used, without substantial reimplementation, for multiple aircraft models. Equally this core knowledge will be used consistently in multiple locations (wing, fuselage, and tail) on the same aircraft.

Representations and Ontologies

Before presenting the project in more detail, we describe our use of the word "ontology". As well as building representations of design knowledge, we wish to also characterize which objects those representations are about. We refer to that characterization as the representation's **ontology**: the ontology is a *specification of that which can be talked about*, a description of the representation's universe of discourse, analagous to a database schema. The ontology states the 'necessary' or universal facts, while theories using that ontology represent 'contingent' or transient facts. An ontology helps a person (or machine) identify how a specific representation 'carves up' the world into concepts, and hence how to interface to that representation correctly. In fact, the boundary between what should go in the ontology and what should be part of a specific theory is not always clear cut, and where the boundary falls is partly a pragmatic choice. However, one of the values of the distinction in the context of knowledge sharing, especially when the ontology has been standardized, is that the axioms that define it don't have to be included in every exchange; instead, they are part of the background context for communication.

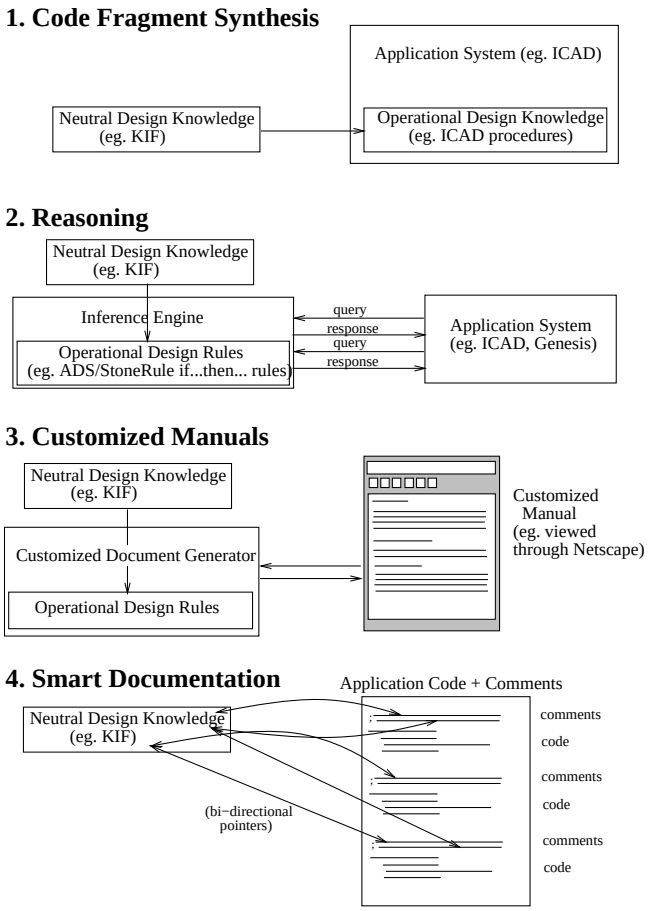

Figure 1: Scenarios of Use for A Neutral Representation.

Scenarios of Use

An explicit, machine-sensible representation of design knowledge opens the door to a wealth of (non-mutually-exclusive) possible uses. To motivate our general goal, we describe several of these:

Code Fragment Synthesis: One scenario of use is the automatic translation of design knowledge from a neutral to operational form, resulting in code fragments in some target language. Such fragments can be embedded within target application software, allowing it to use design knowledge originally encoded in the neutral repository, thus avoiding the expensive, error-prone, manual re-coding of design knowledge for each new application system developed. As well, this should improve are ability to understand precisely what is encoded in software. Does it incorporate required safety separations between electrical and fuel systems? Does it use standard fastener

types? Is it usuable for subsonic and supersonic aircraft? Even these most basic design questions can be difficult to answer if our only recourse is to lines of code.

Reasoning: Perhaps the biggest advantage of a machine-sensible encoding of knowledge is the ability to perform *inference*. Given such design rules, an inference engine can reason with and combine knowledge to answer a wide variety of questions, and tailor answers to an application system's specific context. This approach has been highly successful in many knowledge-based systems, including within Boeing (eg. ESDS (Dahl 1993), which reasons with application data to advise on tasks such as material selection, electrical engineering aspects). A neutral representation could similarly act as an active source of design information for both users and application software.

Customized Manuals: 'Information overload' is a major problem in complex domains such as aircraft design and maintenance. For example, the maintenance manuals for the 777 aircraft contain in the excess of sixty thousand pages, and even with modern search engines provide a formidable challenge for a user to locate relevant information. However, given a machine-sensible encoding of design knowledge and design rules, it becomes possible for a machine to quickly find information pertinent to a user's context and problem, and to synthesize text to describe it using standard natural language generation techniques. This approach has been prototyped in several areas, including aerospace (Marchant, Cerbah, & Mellish 1996) and distributed computing (Clark & Porter 1996).

Smart Documentation: Currently, documentation of designs and design software is typically in free text format, making the relationships and dependencies between different aspects of a design opaque. However, if documentation included pointers back to machine-sensible descriptions of the design knowledge it used, then dependencies between design aspects would become more explicit. As a result, the impact of technology or requirement changes could be more easily assessed, as the dependence of code or designs on specific rules or assumptions would have been recorded. This approach has been applied in several systems (eg. (Klein 1993; Conklin & Begeman 1987)).

Consistency Checking: Finally, given design knowledge represented only in text format, it is almost impossible to ensure that design rules are consistent. Given a machine-sensible representation, however, it becomes feasible to develop techniques to automatically check consistency. This approach has been used in many domains in expert systems research, for example in law for identifying regulatory inconsistencies.

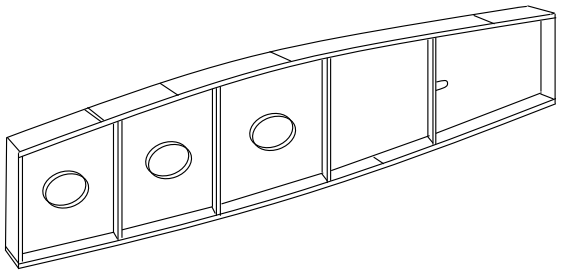

Figure 2: Drawing of a stiffened panel.

```
(defun get-w-n (local-cell-dim max-limit-width
                min-limit-width sep end-sep)
  (if (>= l-cell-dim (+ min-limit-width
                                    (* 2 end-sep)))
      (let-streams ((i (from 1))
            (width (fby (- l-cell-dim (* 2 end-sep))
                        (- (div (- (+ l-cell-dim
                                      sep)
                                   (* 2 end-sep))
                                (tail i)) sep))))
  ((return-when (<= width max-limit-width)
      (if (>= width min-limit-width)
          (list width i)
        (if (< l-cell-dim (+ width (* 2 end-sep)))
            '(0 0)
          (if (> i 2)
              (list max-limit-width (- i 1))
            (list max-limit-width 1)))))))
  '(0 0)))
```

Figure 3: Fragment of (pre-production) ICAD code for stiffened panel layout.

Although there are many possible uses for a neutral encoding of design knowledge, we have initially concentrated on the first of these (code fragment synthesis). We describe the application and approach used in more detail in the next Section.

A Prototype for Stiffened Panels

In the first phase of this project, we have focussed on a small design task, namely the layout of stiffened panels for rigid structures. Minimizing weight, while maintaining structural integrity, is an important goal in aircraft design. A simple example of this is including lightening holes (ie. holes to reduce weight) in a structure, as illustrated in Figure 2. A similar task is locating the number of stiffeners needed (and no more)

for a given panel. The computation of the size and placement of such holes, stiffeners, and other repeated elements requires various different types of knowledge, but this knowledge is typically not made explicit in the software which performs these computations. Instead, it is (at best) buried in the lines of code, or (at worst) not recorded at all. Figure 3 shows a fragment of design code for one such placement task, illustrating the inaccessibility of the design knowledge which it incorporates.

Our goals in this study were to:

1. Identify the underlying knowledge used.
2. Represent it explicitly in a 'neutral' format, and structure that representation into a set of component theories, characterized by the ontology which it uses.
3. Show how that representation can be used to synthesize operational code fragments, which could be used for this and other design tasks.

In this Section we describe the work done on these three tasks. Following this, we discuss the issues, problems and opportunities which this study presented.

Identifying Panel Design Knowledge

By manually analyzing code such as that in Figure 3, and interviewing designers, we were able to identify various fragments of the knowledge used in this task. This includes some **panel geometry knowledge**, implicit in the code, such as:

```
IF   the panel is standard shape
THEN panel length = total width of holes +
                    total width of separators +
                    total width of end-separators.

IF   the holes are of equal size
THEN total width of holes = width of hole * #holes.

IF   the holes are equally spaced
THEN total width of separators = width of separator
                               * number of separators.
IF   panel is standard shape
THEN number of separators = number of holes - 1.

IF   hole sequence is centered
THEN total width of end-separators = 2 * end-sep.
```

In addition, the code implicitly minimizes the number of elements, subject to constraints on their minimum and maximum size. The **design rationale** for this decision is not represented anywhere in the code, but can be reconstructed as one aimed to minimize cost.[1]. This rationale in turn is based on knowledge about the manufacturing process, and costs involved.

[1]This is a best-guess reconstruction for the rationale in this design: there may have been other factors also leading to this design decision which we have not recovered here.

We can additionally make this underlying design rationale explicit; not only will it provide 'traceability' for design decisions, but will allow identification of how the design should be modified if the underlying assumptions change (for example, if a new machining process is developed changing the manufacturing costs). For this purpose, a simple model of **manufactured parts** can be constructed, including rules such as:

```
IF   part is a standard panel
AND  operation is drilling lightening holes
THEN manufacturing cost = stock cost +
                ( cost-per-hole * number of holes ).

IF   part is a standard panel
THEN total cost = K1*panel-weight + K2*manuf-cost.
```

(using some appropriate values for `K1` and `K2`). Similarly, some **basic physics knowledge** and **basic geometry knowledge** is needed to compute parameters used by these rules, including:

```
IF   part is homogeneous
THEN Mass = volume * density.

Weight = Mass * G.

IF   part is a box
THEN volume = height * width * depth.

IF   part is a cylinder
THEN volume = depth * pi * r ^ 2.
```

Representation in A Machine-Sensible, Neutral Form

The above rules are an informal description of (some of) the knowledge underlying this simple design application. There are various possible languages we can use to express this knowledge more formally. We have been evaluating two, both based on first-order logic:

KIF/Ontolingua: KIF is a stable, standardized syntax for first-order logic, which aims to remove all vendor-specific notational idiosyncracies (Genesereth & Fikes 1992). In our experiments we used Ontolingua, an extended version of KIF allowing theories to be specified in a pseudo object-oriented style.

SLANG: A language which supports the construction of separate, modular theories, and their integration together (Jullig *et al.* 1995).

KIF/Ontolingua has particular advantages due to its ongoing emergence as a standard, while SLANG has particular advantages due to its capabilities for merging different theories together in sophisticated ways. To compare both of these, and examine the feasibility of using both, we have expressed the knowledge in both KIF and SLANG, and have prototyped an automatic translator for translating between the two (we comment further on this later in the Discussion Section).

Rather than formalizing the design knowledge as a single, large rulebase, we have broken it up into several component theories which interact together to produce the final design. This modularity helps improve the generality and reusability of the encoded knowledge. The theories we have identified and represented are:

- panel geometry
- manufactured parts
- basic physics
- basic geometry
- materials (not shown above)
- design goals (eg. maximize strength, minimize cost)

Each of these theories in this study is small, containing a few (1-5) definitions and axioms. In addition, each theory can be characterized by the ontology which it uses, expressed as a set of taxonomically related vocabulary terms. The ontology is important for use of the theories, allowing identification of which theories are compatible, and providing a guide to the terms which an application must 'understand' if that ontology is to be imported into it. Figures 4 and 5 illustrate the formalization of this knowledge into KIF and SLANG respectively. Each figure shows (parts of) two of these theories (called "specifications" in SLANG, and "Ontologies" in KIF/Ontolingua), one containing knowledge about manufactured panels, and one of basic physics.

Code Fragment Synthesis from the Neutral Representation

Code Generation Both the KIF and SLANG representations are reasonably language-neutral, in that their syntactic structures closely reflect their meaning (as expressed in first-order logic), and avoid implementation-specific idiosyncracies. However, to be able to exploit these representations in any practical sense, they need to be converted to an **operational form**, for example to Lisp, ADS/Stonerule rules, Prolog etc. The language-neutrality of these representations helps in this task, as the representations are not cluttered with idiosyncratic features unrelated to their meaning. This thus allows for easier, automatic translation of these structures to operational forms.

In this particular study, we have explored one particular scenario of use, namely synthesis of (ICAD-compatible) Lisp code from the original neutral representation. We are prototyping this automatic translation from the SLANG encoding. (Similar techniques could be used to convert from the KIF encoding). In this process, each axiom translates to a Lisp function. The translator, to a first approximation, applies a set of syntactic rewrite rules to convert from the neutral to

```
(Define-Ontology Panels)

(Define-Frame Panel
 :Template-Slots ((Material-Type))
 :Axioms
 ; "Manuf cost = stock cost + cost-per-hole * #holes"
  ((=> (And (Panel ?X)
            (Hole ?X ?Hole))
       (= (Manuf-Cost ?X)
          (Plus (Cost-Of-Raw-Stock ?X)
                (Times (Number-Of-Holes ?X)
                 (Cost-Of-Drilling-Hole ?X ?Hole)))))

   ; "Tot cost = K1 * panel-weight + K2 * manuf-cost"
   (=> (Panel ?X)
       (= (Cost ?X)
          (Plus (Times Purchase-Constant (Weight ?X))
            (Times Manuf-Constant (Manuf-Cost ?X)))))
  ... ))

(Define-Frame ....)
...
(Define-Ontology Basic-Physics)

(Define-Frame Mass
 :Axioms
 ; "Mass = volume * density."
  ((<=> (= (Mass ?PhysObj-0) ?Value)
        (= (Times (Volume ?PhysObj-0)
                  (Density ?PhysObj-0)) ?Value))))

(Define-Frame Aluminum-7075
 :Axioms
 ; "Parts of aluminum-7075 have density 5021.32."
  ((Forall (?Obj)
        (=> (= (Material-Type ?Obj) Aluminum-7075)
            (= (Density ?Obj) 5021.32)))))
...
```

Figure 4: Design rules, encoded in KIF/Ontolingua.

Lisp-based structure. For example, the SLANG axiom:

```
definition manuf-cost-def of manuf-cost is
axiom (equal (manuf-cost p)
         (plus (cost-of-raw-stock p)
               (times (number-of-holes p)
               (cost-of-drilling-hole p (hole p)))))
```

would translate to the Lisp function:

```
(defun manuf-cost (panel)
 (+ (cost-of-raw-stock panel)
    (* (number-of-holes panel)
      (cost-of-drilling-hole panel (hole panel)))))
```

We could apply similar techniques to generate Prolog rules, producing for example:

```
manuf_cost(Panel, Cost) :-
    isa(Panel, panel),
    cost_of_raw_stock(Panel, StockCost),
    number_of_holes(Panel, NHoles),
    hole(Panel, Hole)
    cost_of_drilling_hole(Panel, Hole, CostPerHole),
    Cost is StockCost + (NHoles * CostPerHole).
```

```
spec manufactured-panels
  % "manuf cost = stock cost + cost-per-hole * # holes."
  definition manuf-cost-def of manuf-cost is
  axiom (equal (manuf-cost p)
        (plus (cost-of-raw-stock p)
           (times (number-of-holes p)
              (cost-of-drilling-hole p (hole p)))))
  end-definition

  % "Total cost = K1 * panel-weight + K2 * manuf-cost."
  definition cost-def of cost is
  axiom  (equal (cost p)
        (plus (times purchase-constant (weight p))
           (times manuf-constant (manuf-cost p))))
  end-definition
  ...
end-spec

spec basic-physics
  % "Mass = volume * density."
  definition Mass-def of mass is
  axiom ( equal (mass x)(times (vol x)(density x)))
  end-definition

  % "Parts made of aluminum-7075 have density 5021."
  axiom ( implies (equal (material x) al-7075)
        (equal (density x) 5021) )
  ...
end-spec
```

Figure 5: Design rules, encoded in SLANG.

Ontological Commitment Under this 'code synthesis' scenario of use, vendor-specific code fragments containing design knowledge ("knowledge nuggets") can be generated for use by a designer. However, there is more involved than simply splicing code into an existing design system – for the fragments to work, the application must *conform* to the vocabulary of the imported design knowledge. Conformance[2] means that the application's objects and methods can be made to match those of the imported theory. For instance, consider the synthesized Lisp function just described for `manuf-cost()`. Importing this requires that the application can provide an object denoting a `panel` which has functions `cost-of-raw-stock()`, `number-of-holes()`, `hole()` and `cost-of-drilling-hole()` defined on it, or alternatively import additional theories which define these operations in terms of other primitives which the application can provide. Ontologies, as the objects which characterize these conceptual vocabularies, help the system designer identify what commitments the theories make, and hence what vocabulary the application must be made to conform to in order that those theories can be used. At present, our ontologies have a simple structure, consisting of a list of the objects and relations used in a theory, their taxonomic relationships, and the domain and range constraints on relations.

The scenario of use, then, is that the application engineer select one or more ontologies (corresponding to the objects manipulated in the application), and then import the theories he/she requires which are expressed in terms of those ontologies. Ideally, the application would be designed to conform to particular, established ontologies from the outset. It is important to note that applications are already written to conform, in a less rigorous way, to the vocabularies used in Boeing's design manuals: for example, if the design manual treats stiffeners (say) as having three different types, then an application will similarly treat stiffeners as having three different types in order that the design manual's rules can be expressed in the software. The issue of conformance, then, in the neutral representation project is not a new one, but an attempt to put it on a more formally defined footing.

[2]We prefer the word 'conformance' to 'commitment': An application is not constrained to be *written* using a particular vocabulary, but only to be capable of mapping ("conforming") its vocabulary and a theory's vocabulary together in order to import it.

Discussion

In the previous Section, we outlined the study we conducted in identifying, representing and using design knowledge, expressed as theories plus ontologies, for a simple design task. We now provide some discussion of the issues which arose in this study.

What Should an Ontology Contain?

We have argued for the need for a "specification of that which is talked about" in a theory – the theory's ontology – to help a person (or system) understand what a theory is about, and understand its compatibility with other theories. We have argued the ontology should be distinct from the theory itself, but what then should an ontology contain? At present, our ontologies consist of the theory's conceptual and relational vocabulary, taxonomic relationships between those concepts, and domain and range constraints on those relations. However, we could have also included other information, eg. English definitions, meronymic (part-whole) relationships, or more or even all of the axioms in the theory. It seems there is no crisp answer as to where the content boundary should lie: rather there is a trade-off between ontologies containing a small amount of information (making it simple and comprehensible) and a large amount (making the specifications more precise). The two extremes of this would thus be just a list of vocabulary terms (simplest), or the entire theory it-

self (most detailed). Advocating an ontology should only contain the 'major' or 'relatively unchanging' axioms begs the question as to which those axioms are. However, although it may be a pragmatic decision as to what an ontology should contain, we do see a clear role for *some* form of characterization of a theory, separate from the theory itself. Further work on using ontologies to achieve the specific goals for which they were built (eg. conveying a conceptual structure to people, or for automatic determination of whether theories are compatible) will help clarify what is useful and what is superfluous as ontology content. Interestingly, in the database community the distinction between the specification of the conceptual vocabulary (ie. the database schema) and content (ie. the database itself) is apparently cleaner than for knowledge-based systems.

Ontological Boundaries

In our initial study, we treated an ontology as characterizing the concepts and relations used in a theory. In its simplest formulation, each theory has its own ontology. However, in practice, we envisage an application importing rules from multiple theories (eg. where rules in one theory compute parameter values used as primitives in another), and hence the application making an "ontological commitment" to a larger ontology which includes all these component ontologies. In fact, all the theories we have constructed for this initial study could be be viewed as conforming to a single, large ontology, as our suite of theories does not contain different, conflicting theories about the same phenomenon. This raises the question of how to best characterize a theory's ontology: Should the ontology contain just the concepts explicitly used in the theory? If so, how can ontologies from different theories be combined, to characterize the overall ontology used by a collection of (merged) theories? Or should the theory be viewed as part of a larger ontology, including concepts it does not specifically reference? If so, how should the boundaries of that larger ontology be chosen?

Biased Use of 'Neutral' Languages

In an ideal scenario, theories in a 'neutral' language such as KIF will be translatable to multiple operational languages (eg. LOOM, Prolog). However, in practice only a subset of first-order logic can be easily expressed in these operational languages, and that subset will differ depending on the target language. This raises a dilemma: if KIF (say) theories are written without regard to the target language, then the theories become untranslatable and hence of no operational use; on the other hand, if KIF is written for a specific target language (ie. using only KIF constructs which are easily translatable to that target), then we lose all the benefits of neutrality and are unable to deliver the theory's contents to systems based on other target languages. Although translation to and from KIF has been demonstrated elsewhere for individual languages (eg. LOOM), there is still a need for a convincing demonstration of translating from a target language, to KIF, and then to a *different* target language. Achieving this requires both identification of language constructs which are translatable to many targets (eg. Horn clauses), and further research on translator technology. The important point here is that neutral languages are not a panacea, and careful consideration of their target uses is essential for the representations to have operational value.

The Practicality of Declarative Semantics

The two languages we have considered (KIF and SLANG) both have declarative semantics, in that it is possible to understand the meaning of individual expressions in the language in isolation, and without appeal to an interpreter for manipulating those expressions (unlike, say, a language where rule ordering affects their meaning). While many statements and rules are concisely expressible in these languages, other statements are less practical to express. For example, it is often the case that design rules have exceptions, and have exceptions to exceptions, etc. While such rules can be expressed in a declarative, first-order logic syntax ("`if A and (not B) and (not C) and (not (not D)) and ...`"), this can in practice make for large and unreadable expressions. Other notations (eg. inheritance hierarchies with exceptions) can offer a syntactically neater way of expressing these kinds of constructs. At the minimum, translation of such constructs to and from KIF is complicated in these cases.

Centralized vs. Distributed Repository

Throughout this paper, we have implicitly assumed a centralized repository containing a neutrally represented design knowledge. However, an alternative would be to have a 'virtual' repository, with different theories of design stored in the different systems which used them. Accessing knowledge would then involve knowledge exchange between a heterogeneous set of applications, rather than from a centralized resource (although to each application it would appear that a single 'virtual' repository was available to it). The distributed model requires an additional ability of applications to 'publish' the knowledge they know, in request to queries from other applications. In practice, earlier experiments in knowledge sharing (eg. PACT (Cutkosky *et al.* 1993), SHADE (McGuire *et al.* 1993)) employed a hybrid approach, where some

knowledge was exchanged between existing applications while other knowledge was explicitly formalized in a new, centralized ontology resource for those applications.

Use of Existing Knowledge Resources

There are several extensive resources already developed elsewhere, which may be of benefit to this project if we were able to import and use them. These include the Ontolingua library (containing a collection KIF-based theories) (Gruber 1993), the Cyc ontology (Cycorp, Inc. 1996), and the large amount of work done in STEP (Chen 1996; SCRA, Inc 1996). However, there are also significant barriers to being able to import and use knowledge from these resources "off the shelf". For Ontolingua, the difficulty of translating arbitrary KIF expressions to multiple operational languages remains a barrier. For STEP, there is currently no support for exchanging general design rules (as opposed to specific instance data), although Express (its conceptual schema language) does allow rules for constraint-checking to be stated. In addition, the challenge of conveying the content of these resources (without simply reciting them in their entirity) still remains a challenge needing to be addressed for their general exchange and reuse.

Summary

In modern design systems, there is a vast amount of redundant re-encoding of design knowledge, each re-encoding adding to development, validation, maintenance and synchronization costs. The root of this problem is that encoded design knowledge is inaccessible to other systems (and people), due to its being 'locked up' in task- and vendor-specific lines of code. The neutral representation project aims to reduce this problem by prototyping methods for encoding knowledge in an explicit, neutral, and machine-sensible form.

In the initial study presented here, we encoded design knowledge for stiffened panels as a set of explicit rules, represented as KIF and SLANG theories, and with associated ontologies characterizing the conceptual vocabulary of those theories. The study has illustrated the viability of both building and using such representations, and also raised several important issues about ontology content, neutral language translation, and knowledge sharing. During 1997 we are planning a larger application, which we hope will further advance our understanding of these issues.

References

Chen, S. 1996. Step: Links to projects and organizations. (http://gsun6.gintic.ntu.ac.sg:8000/~chchan/link/step.html).

Clark, P., and Porter, B. 1996. The dce help-desk project. (http://www.cs.utexas.edu/users/mfkb/dce.html).

Conklin, J., and Begeman, M. L. 1987. gIBIS: a hypertext tool for team design deliberation. In Smith, J. B., and Halasz, F., eds., *Hypertext '87 Proceedings*, 247–251. NY: Assoc. Computing Machinery.

Cutkosky, M. R.; Engelmore, R. S.; Fikes, R. E.; Genesereth, M. R.; Gruber, T. R.; Mark, W. S.; Tenenbaum, J. M.; and Weber, J. C. 1993. PACT: An experiment in integrating concurrent engineering systems. *IEEE Computer* 28–37.

Cycorp, Inc. 1996. The cyc public ontology. (http://www.cyc.com/public.html).

Dahl, M. 1993. ESDS: Materials technology knowledge bases supporting design of boeing jetliners. In *Proc 5th Innovative Applications of AI (IAAI-93)*, 26–33. CA: AAAI Press.

Genesereth, M. R., and Fikes, R. E. 1992. Knowledge interchange format: Version 3.0 reference manual. Tech Report Logic-92-1, Computer Science, Stanford Univ, CA. (http://logic.stanford.edu/kif/kif.html).

Gruber, T. R. 1993. A translation approach to portable ontology specifications. *Knowledge Acquisition* 5(2):199–220.

Jullig, R.; Srinivas, Y. V.; Blaine, L.; Gilham, L.-M.; Goldberg, A.; Green, C.; McDonald, J.; and Waldinger, R. 1995. Specware language manual. Technical report, Kestrel Institute. (http://kestrel.edu/www/specware.html).

Klein, M. 1993. Capturing design rationale in concurrent engineering teams. *IEEE Computer* 39–47.

Marchant, B. P.; Cerbah, F.; and Mellish, C. S. 1996. The GhostWriter project: a demonstration of the use of AI techniques in the production of technical publications. In *Expert Systems '96 (Proc 16th Annual Conf of the BCS Specialist Group on Espert Systems)*.

McGuire, J. G.; Kuokka, D. R.; Weber, J. C.; Tenenbaum, J. M.; Gruber, T. R.; and Olsen, G. R. 1993. SHADE: Knowledge-based technology for the re-engineering problem. *Concurrent Engineering: Applications and Research (CERA)* 1(2).

Proctor, P. 1995. Boeing adopts 'expert' design system. *Aviation Week & Space Technology* 27.

SCRA, Inc. 1996. Step/pde: Frequently asked questions. (http://www.scra.org/uspro/faq.html).