Toward a Knowledge Medium for Collaborative Product Development[*]

Thomas R. Gruber
Stanford Knowledge Systems Laboratory
701 Welch Road, Building C
Palo Alto, CA 94304
gruber@sumex-aim.stanford.edu

Jay M. Tenenbaum
EIT, Incorporated
459 Hamilton Ave, Suite #100
Palo Alto, CA 94301
jmt@eitech.com

Jay C. Weber
Lockheed AI Center
3251 Hanover St.
Palo Alto, CA 94304
jay@laic.lockheed.com

Abstract

Information sharing and decision coordination are central problems for large-scale product development. This paper proposes a framework for supporting a *knowledge medium* [32]: a computational environment in which explicitly represented knowledge serves as a communication medium among people and their programs. The framework is designed to support information sharing and coordinated communication among members of a product development organization, particularly for the tasks of design knowledge capture, dynamic notification of design changes, and active management of design dependencies. The proposed technology consists of a shared knowledge representation (language and vocabulary), protocols for foreign data encapsulation and posting to the shared environment, and mechanisms for content-directed routing of posted information to interested parties via subscription and notification services. A range of possible applications can be explored in this framework, depending on the degree of commitment to a shared representation by participating tools. A number of research issues, fundamental to building such a knowledge medium, are introduced in the paper.

[*]In *Artificial Intelligence in Design '92*, John S. Gero, editor. Proceedings of the Second International Conference on Artificial Intelligence in Design, Pittsburgh, USA, June 22-25, 1992, pages 413–432. Kluwer Academic Publishers, 1992.

1 Introduction

1.1 The Need for Knowledge Sharing and Communication Coordination in Cooperative Product Development

Product development is a knowledge- and communication-intensive process. For collaborative product development, team members need to have access to the knowledge underlying decisions by other members. They also need to assess the impact of their decisions on each other, and notify the affected parties in an appropriate way. Effective communication is especially important whenever anything affecting an existing design decision changes. In product development, something is always changing — perhaps a design requirement, an unanticipated simulation or test result, the availability of a component, or an improvement to the manufacturing process. Reacting quickly to such changes is essential for quality and productivity.

While computers are used extensively in product development, existing tools do little to facilitate information sharing and coordination. If anything, they aggravate the problem by isolating information at tool boundaries. Most computer tools support particular tasks in engineering (e.g., geometric modeling, analysis), manufacturing (e.g., process planning, scheduling) or business (e.g., cash flow analysis). Often, the principal output of one tool is a piece of paper that is mailed or faxed to team members in other departments. Those individuals must then re-enter the relevant information in the format required by their tools, resulting in team members being cut off by their tools. They make decisions on the basis of inconsistent or out of date information. Moreover, only information concerning the artifact is generally available; critical information about the decisions leading to design choices and their underlying rationale is seldom ever captured much less communicated.

1.2 The Role of a Knowledge Medium

Imagine a computational environment in which explicitly represented knowledge serves as a communication medium among people and their programs. Such an environment is called a knowledge medium [32]. Information exchange and decision making of team members is mediated by a shared knowledge base and interaction protocols. On-line tools, such as CAD/CAM, document editors, and electronic mail interfaces interact by posting information about different aspects of the design to a shared environment, in a common representation. The interactions among shared objects are represented with new knowledge structures that span the applications and engineering disciplines. Formal behavioral models are related to simulation demonstrations; design decisions to requirements; text and audio annotations to formal schematics. The relationships among shared objects range from unspecified dependency ("object X is in some way dependent on object Y") to precise analytic constraints ("$Y = 2x + 3$") and computer-generated explanations of causality ("X caused Y to fail").

When something in a product knowledge base changes, automatic analysis determines how the change relates to other product data, either by following explicit links or by inference. The change in a parameter triggers a spreadsheet-like recalculation of dependent parameter values in an equation model. The change in a different parameter invokes a rule-based constraint checker to verify that the value is within the allowable range. When dependencies are not captured precisely, the shared environment can notify all designers that the changes potentially affect and the designers make the determination of if and how to respond.

1.3 SHADE: A Representational Framework for Design Knowledge

As a start toward this vision, we are developing a representational framework called SHADE[1] to support the sharing of knowledge about designs and the coordination of information about design changes to interested parties. By *representational framework* we mean a common knowledge representation (language and vocabulary) and software interaction protocols designed to support a set of knowledge-based services. SHADE will provide the infrastructure in which the information created and manipulated by heterogeneous engineering tools can be interrelated and coordinated. Such tools include current and future word processors, CAD drawing tools, model editors, simulators, constraint checkers, and electronic mail. The target services include design knowledge capture, dynamic notification of design changes, and active management of design dependencies.

SHADE is at the core of next generation concurrent engineering systems under development at Stanford and Lockheed. The framework is expected to lead to a new generation of tools such as electronic design notebooks for recording and retrieving engineering knowledge, selective dissemination of design decisions and changes, and explanation and validation of design requirements.

The project to develop the SHADE representation and associated computational environment has only recently begun. In this paper we illustrate the intended functionality in the context of engineering redesign (Section 2), then specify the requirements (Section 3), describe the technical approach (Section 4), characterize the important research issues (Section 5), and briefly discuss the research methodology (Section 6).

2 A Scenario of Use

This section will describe a scenario in which an engineer is working in the proposed SHADE environment supporting cooperative product development. The first part of the scenario is a series of interactions between a designer and current, state of the art, engineering tools. The second part describes a hypothetical extension in which SHADE representations and services provide assistance in a collaborative setting.

2.1 Today: Eclectic Use of Isolated Design Tools

The setting is an engineer redesigning a planar manipulator, a two-fingered robot manipulator that moves a payload around in a planar workspace. The engineer is working with editors for two design descriptions: the textual requirements document and a component/connection graph of the power supplies, motors, and linkages comprising the electromechanical subsystem. The requirements document contains a section marked with a change bar. The marked

[1]for SHAred Dependency Engineering

text prescribes an increase in the maximum payload weight over the previous design. The engineer is tasked with modifying the design to accommodate the changed requirement.

The engineer starts by simulating the manipulator. She calls up a case library of simulation scripts configured for the previous design. Looking at the titles, she selects one labelled "demonstration of planar manipulator under maximum loading conditions," and runs it. The simulation system shows a graph of the position and velocity of a series of payloads under a standard test trajectory. The manipulator works fine. Then the designer adjusts the mass parameter of the payload and runs the simulation again. The simulation shows the manipulator slowing down and then failing to complete the test trajectory with the heavier payload. This is the behavior that the redesign is to fix.

To accommodate the heavier payload, she decides to replace the present motor in the planar manipulator with a bigger one. She calls up the new motor from a library, and using her component/connection editor, replaces the old one. She selects the component in her graphical interface, and then gives it an annotation: "bigger motor needed for heavier payload requirement." She then links the annotation to the previously circled requirement in the document using a mouse gesture.

She now runs the simulation using the new design, and verifies that the new motor behavior achieves the required behavior. Using a menu she saves the script and labels it with "new motor handles heavier payload." Using the mouse she graphically links this annotation to the circled requirements text and the motor icon in the component/connection editor.

This scenario is not typical of current practice, but it is completely plausible to implement with existing technology. None of the individual tools are new: the document editor, the component-centered CAD editor, the simulation system with model library. The user interface technology also exists: selecting arbitrary objects with the mouse, drawing hyperlinks between objects, attaching textual annotations to links. What is missing is the infrastructure.

2.2 Tomorrow: Loosely-Coupled Tools in a Collaborative Environment

Consider an extension to the scenario in which the designer is not working alone, but with others in an organization. When the designer selects objects in editors, runs simulation programs, and links objects and data with annotated links, the environment records these design activities in a permanent record called a design history. The system indexes the history by the selected objects and data. This history is accessible by other engineers later in the product lifecycle, and it persists after the original designers become unavailable.

One of these downstream engineers is trying to plan the manufacture of the artifact and discovers that the motor specified in the design is no longer available from the suppliers. Since she will have to substitute another motor, she needs to know why the specified motor was chosen. So she looks into the design history, keying the search on the motor part number. She is shown the information manipulated by the original designer: the changed requirement, the component/connection graph, the manufacturer's data sheet, and the related simulation results. This helps her find a compatible replacement by telling her something about the role of the previous motor in the overall design.

Consider a further extension to the scenario, in which engineers work *concurrently* and exchange data and share models as they are created and refined. Now the design knowledge environment is an active medium, *notifying* interested parties of relevant changes to the distributed design knowledge. In this scenario, when the designer replaces the motor, the change triggers a set of notifications sent to tools used by other engineers. One is for the manufacturing engineer responsible for milling the frame for the manipulator. After being notified of the motor change, she notices that the new motor has a larger shaft diameter. She acts on this information by increasing the size of the hole in the CAD specification of the mounting bracket. Then she invokes her process planning tool to verify that the specified bore hole is within manufacturability limits, and produces a new plan for machining the bracket. If the process planning tool uncovers some manufacturability problems, then appropriate designers can be notified immediately.

3 Requirements for a Representational Framework

Support for the kinds of services suggested in the scenario imposes an interesting set of requirements for a representational framework.

First, SHADE must provide a representation for shared design knowledge. The representation must be shared in the sense that participating tools can assert and exchange information within a common conceptualization. For example, a service for capturing relationships among a requirement, a motor selection, and a simulation result needs a representation for these classes of information, and a mapping to the data available from tools. The representation must be expressive enough to capture domain-specific knowledge — not just data – used by participating tools. Simulation models, manufacturability constraints, and data dependency graphs are examples of knowledge to be represented.

Although the shared representation must be able to incorporate many kinds of engineering knowledge, this doesn't imply that the representation must be capable of expressing the union of all distinctions made by the participating engineering tools. In fact, the challenge is to support degrees of shared knowledge from minimal sharing to strong common models.

The second requirement is closely related. SHADE must provide a means to represent the relationships among "foreign" data in heterogeneous formats from very different tools. In the scenario, the designer related a variety of sources of information. The design requirements were selected in text editors as strings, the model for the motor was selected from a component library and inserted as an icon into an editor, then incorporated as a behavior model in a simulation. The framework needs some way to link these data in a globally consistent way.

One option is a common data format, standardized down to the data structures. This demands a huge commitment from the developers of the participating tools, and is difficult to achieve outside of tightly integrated product lines. Instead, we reject any requirement that the fine-grained structure of all design elements be "standardized" into a common data model. All that is required is the ability to *encapsulate* design elements with a identifier and some characterization of what the element denotes. For example, a single design move might be reflected in a complex series of transactions on a CAD model, resulting in a new shape for a part. For the purpose of representing the *relationship* between the change in the geometry and other elements such as the dynamical behavior and manufacturing cost, only an identifier of the design move need be represented in the shared knowledge base. The geometric details are left to the geometry model supported by the CAD editor. The contents of the encapsulated design element is said to be *opaque* to the shared representation.

Third, as a knowledge medium for human members of product development teams, the representation must support both structured information that is interpretable by computers and unstructured information that may only be meaningful to people. Examples of the former include behavior equations and netlist descriptions of circuits. Examples of the latter include E-mail discussions, pages from a handbook containing text and diagrams, or animated displays produced by a simulation program. In the scenario, the justification of the design decision included a mix of machine-interpretable data (e.g. replayable simulations) and human-only information (e.g., annotations, informally specified requirements). A representation for such as mix of objects is called *semi-structured* [28].

Fourth, the framework must provide mechanisms for content-based information routing and automatic invocation of demons in response to changes in the data. In content-based routing, when a piece of information is made available to the shared environment, it is delivered to interested consumers on the basis of the information content. The mapping of content to addressee is based on a specification of interest called a *subscription.* This is the kind of routing provided by electronic "news clipping" services, typically on the basis of keyword indexing. The framework must provide the language for specifying interest, the mechanisms for implementing the mappings, and the protocols by which participants post information for distribution. Similarly, it must establish a protocol for associating changes in the data with the activation of "demons" (programs).

4 Elements of the SHADE Technology

SHADE is designed to achieve the aforementioned requirements and to enable the kinds of computational support illustrated in the scenario. There are three basic parts to the SHADE approach. First, design knowledge in the shared environment is specified and exchanged using a *shared knowledge representation.* This is a standard syntax and basic domain vocabulary, in a reusable form, along with a knowledge base that is accessible by all of the participating engineering tools in the framework.

Second, knowledge is exported from engineering tools into the shared environment via a protocol for encapsulation and publication. Encapsulation transforms tool-specific data into a form that is in conformance to the common knowledge representation. Publication is the process of posting new information, such as changes to a design, in way that can be monitored by demon mechanisms.

Third, the content-based routing of information among participating tools is achieved with a protocol for *subscription* (specifying interests) using the shared knowledge representation as a specification language and the *propagation* of new information to interested parties based on their subscriptions. A SHADE subscription looks like a query to a deductive database; it is an expression with some free variables to be instantiated by the desired information. The path of information flow from producer to consumer is determined according to the contents of the information packets (i.e., what the expressions mean) rather than by pre-arranged addressing.

These are the essential commitments of the approach. Let us now examine some important details.

4.1 A Shared Knowledge Representation for Specifying and Exchanging Design Knowledge

At the heart of SHADE is a representation for knowledge about a designed product. The knowledge may come from any of several sources, including human engineers and varieties of software tools. The representation formalism must be very expressive to capture the kinds of intra-tool relationships motivated by the scenario (e.g., design decisions involving requirements, structural modifications, simulation results, and textual annotations). Analyses of information demands and usage of designers in several domains support this need to represent many kinds of design knowledge from many sources [18, 21]. In addition, statements in the representation will be interpreted by humans and by tools that do not use the representation internally. For these reasons we have chosen to base the SHADE representation on a declarative language for first-order logic called KIF [11]. KIF is a proposed specification for a standard Knowledge Interchange Format, or Interlingua. It has the desired properties of interpreter-independence (it is based on well-defined semantics which do not assume any particular inference procedure) and expressiveness (it is sufficient to represent anything that can be stated in first-order logic, and also allows statements to refer to other statements as terms). KIF, however, only provides the basic syntax and formal semantics.

The bulk of the shared representation is a *common ontology* of design knowledge. In philosophy, an ontology is a systematic account of existence. In the context of AI systems, we identify the ontology with the set of formal terms with which one represents knowledge (which determine what can exist for the program). Practically, then, a common ontology is the vocabulary for representing the knowledge needed for some purposes in some domain. It is a dictionary of classes, relations, functions, and object constants, along with their definitions in human-readable text and machine-interpretable KIF sentences. In SHADE we are using a system called Ontolingua [16] for developing and maintaining the ontologies.

One can think of the design knowledge ontology as an evolving theory of distinctions that are worth representing in a *shared* design knowledge base (not an arbitrary design tool). The SHADE project is tasked with producing an ontology for design knowledge for the purposes described earlier: design knowledge capture, dynamic notification of design changes, and active management of design dependencies. Much of the intellectual foundation has been laid, and we are drawing from representation work in design automation [34], decision support [1], design rationale [23], dependency management [27, 31], applied mathematics [13, 35], behavior modelling [6, 10, 30, 36], and the representation of everyday human experience [25].

We plan to develop a basic ontological foundation and then extend it in the directions indicated by how it is used in practice by engineers. The initial SHADE ontology (based on [17] and an informal study of group ontology development) will include classes for behavioral descriptions, constraint expressions, design decisions, dependencies, design criteria, design moves, design parameters, functional descriptions, requirements, and structural descriptions. These classes reflect distinctions that have some utility for the design rationale capture and change notification tasks, and which can be naturally mapped onto existing engineering tools. With experience and the addition of participating tools, SHADE's ontology of design knowledge will be extended to integrate simulation and synthesis models, engineering data from on-line databases and libraries, and models of engineering processes throughout the product lifecycle.

The language and vocabulary are used to create a *shared knowledge base* (SKB) of information about the relationships among changing data in the participating tools. The domain of the SKB is *not* all possible information about the design (i.e., the union of all the tools' domains), but how the data produced and consumed by tools are related. The shared knowledge base serves as a common carrier for information communicated by participating tools, conceptually like a shared memory or bus. However, the SKB need not be centrally located or permanently stored; it may actually be distributed among the knowledge bases of the participating tools. This is the subject of ongoing work beyond the scope of this paper.

What is in the shared knowledge base? Consider, for example, the first scenario. When the designer graphically links the requirement text, the new motor, and the simulation, she is actually instantiating a template for recording design decision information. The relationship between these elements would be asserted into the SKB using sentences like the following. The sentences formally relate objects representing the requirement text, motor, and simulation scenario, using a design-decision object.[2]

```
(instance-of dd-1 design-decision)
(instance-of rt-1 requirement-text)
(mentioned-requirements dd-1 rt-1)
(instance-of sc-1 structural-component)
(relevant-structures dd-1 sc-1)
(instance-of ss-1 simulation-scenario)
(relevant-behaviors dd-1 ss-1)
```

That is, the design decision `dd-1` is like a "frame" that captures a systematic relationship among the requirement `rt-1`, the device component `sc-1`, and the results of a simulation, `ss-1`.

All tools that participate in the SHADE environment have access to the contents of the SKB, and can assert to it. The mapping from tool representations to the shared representation is the topic of the next subsection.

4.2 A Protocol for Encapsulation and Publication

The second component of the SHADE framework is the protocol by which loosely coupled, heterogeneous tools can export (publish) information to the SKB. The protocol is constrained by an interesting set of requirements:

- Each tool has its own internal representation and data model.
- Only some of the information processed by the tools is machine- interpretable (e.g., equations are; natural language text and bitmap images are not).

[2]This example is only meant to convey the style of representation. The actual ontology for representing design decisions is under development.

- Only some of the data can even be *represented* in the shared ontology (because the vocabulary is always incomplete).

The SHADE approach to publication is based on a few simple ideas. First, "foreign" data from the tools can be organized in terms of objects and sentences about the objects. For instance, in structure editors the objects might be components, and in a word processor, the objects might be text fragments selected by the user. Properties or attributes of objects, and their relationships to other objects, can be stated as sentences referring to the objects. Second, although there is a finite set of persistent objects in an design environment, the number of sentences one can think up about these objects is unlimited. Therefore, in the context of an arbitrary collection of independently designed tools, it is possible to share a database of objects, but not to share a complete knowledge base of sentences about them. We say that such objects are *opaque*, because the shared KB can represent the identity of the object, but not its internal details. The fourth idea unifies the problem of representing foreign data with the problem of incorporating semi-structured data. Whether the object is a formal geometric entity, fully interpreted by its originating CAD tool, or a semi-structured videotape of a simulation, interpretable only by human observers, the same opaque-object abstraction can be applied. In other words, SHADE can encapsulate an entire video as if it were a foreign object coming from a CAD tool.

The SHADE publication protocol follows naturally from this analysis. The details are under design, but the basic policy for encapsulation and publication mechanism is as follows.

- Each tool exports facts about objects it manages. The objects are encapsulated as *handles* to the shared environment (i.e., identifiers that the tool associates with persistent internal data structures). The publication protocol does not specify *which* objects and facts to export (see Section 4.3).

- Opaque objects are denoted by unique names in the shared representation. SHADE is responsible for assigning identifiers from the global namespace to object handles given by exporting tools. The unique names are formal object constants in KIF.

- The unique identifiers in the shared knowledge representation denote the objects in the world, not their binary representations in tools. The only meaning imparted to an opaque object in the shared knowledge base is its unique identity; it can be tested against other opaque objects for equality. This is sufficient to index and search for these identifiers, using simple pattern matching (no deep inference). Everything else known about an opaque object is represented with KIF sentences. The existence of an object is implied by it being asserted as an instance of a class. A permanent property of an object is asserted with a binary relation or unary function.

- SHADE applications can assert relationships among opaque objects, such as the constituents of design decisions. These relationships have a well-defined interpretation (specified in the common ontology) even if the contents of the opaque objects are only interpretable by humans or specific tools. This is guaranteed by requiring that all relation terms used in sentences be defined in the shared ontology.

- Exporting tools are responsible for the user interface that presents exported objects to humans. Some user gestures will indicate that an object is selected, and therefore party to some relationship (e.g., a design decision in the scenario). SHADE programs may call on tools to display previously exported objects. This may require the tool to reconfigure to a previous state (see Section 5.3 on issues of representing versions and time).

In the the design decision scenario of Section 2.1, when the user selects a requirement in the requirements editor, the editor associates the text string with a locally unique handle (which could be the actual bits encoding the data). It exports this handle with the assertion that the text is of type requirement-text. SHADE maps the tool-specific handle to a unique name in the SKB (`rt-1`), producing the assertion

```
(instance-of rt-1 requirement-text)
```

The same process applies to the export of the motor component from the structure editor and the simulation scenario from the simulation tool. If other members of the project team were interested in the simulation data, they could ask SHADE to request that the scenario `ss-1` be presented. The term `ss-1` denotes the scenario. On the simulation tool end the scenario is stored as a script specifying behavior models, input conditions, and other input parameters. Recreating the scenario is achieved by rerunning the simulation using the script.

4.3 Subscription and Notification

As stated earlier, a major service enabled by SHADE is content-based routing of information. As facts about opaque objects are published, the information is made available to interested parties. Conceptually, any SHADE participant can query the shared knowledge base. However, as a practical matter the virtual knowledge base may be distributed over many local knowledge bases/tools. Furthermore, it is impractical for every program to continually monitor all sources of shared information on the network. This is why it is important for the representational framework to provide the basic services for coordinating the flow of information among tools based on specifications of information needs and information generated.

The essential elements of the SHADE mechanism for content-based information routing are subscription, notification, and determining relevance. A subscription is a query using the using the shared knowledge representation. Notification is the delivery of the answer, also in the shared knowledge representation. When information is exported to the shared knowledge base, those parties who subscribed to that information are notified of the new information. Determining relevance is the task of deciding which parties would be interested in a given piece of information. There are many open research questions concerning how the messages can be routed, and how parties to the information negotiate the exchange (see [12] for details).

In the scenario of Section 2.2, the manufacturing engineer wanted to be notified when any component is added to the artifact structure model. She might specify this as

```
(interested-in agent-9
   '(instance-of ?x structural-component))
```

This says that `agent-9` should be informed when any new structural components are posted to the shared environment. Agents are programs obeying the protocol, in this case agent-9 is acting on behalf of the manufacturing engineer. When the new motor is added to the design, the agent might be informed of the fact as it was posted:

```
(instance-of sc-1 structural-component)
```

The human who owns the monitoring agent might be notified with an email message. Upon notification that *something* is happening with regard to a component, she could ask the program for details, which might turn up the link between the new motor and the design decision:

```
    (relevant-structures dd-1 sc-1)
```

Note the subscription and notification rely on the shared definition of the term `structural-component` in the shared ontology. The engineer's understanding of this term needs to be in agreement with the tool that posted the new information (practically, the agreement is with its programmer). Similarly, the design history capture tool commits to terms in the ontology for representing decisions and relating them to requirements, structures, validation test results, etc. But in this kind of subscription, the ontological commitment is very circumscribed. Relevance can be determined syntactically, with a simple pattern matching procedure. Nothing else matters about the `?x` but that it is a structural-component. The opaque object for the motor might be represented in the originating tool as a bitmap image, a geometric model, a table of manufacturer data, or a set of equations. *For the purposes of notification and decision management,* these distinctions may not matter.

Even with agreement on terms by all parties, it is difficult to anticipate the actual form in which a fact may be asserted. For the same reason, it is also difficult to formulate many queries as simple facts that will match syntactically with assertions made by information producers. The long-term power of content-based routing as an interoperability principle comes when one allows inference between subscription and notification. The knowledge used to infer the connection between new information and the relevant subscriptions is represented in *relevance theories*. Relevance theories are also represented in the shared representation.

4.4 Reasoning About Dependencies and Change with Relevance Theories

With a declarative representation one can write theories about the relevance of changes in the shared KB to specified interests. A theory might be formulated as a set of rules that describe how a change in one design element is related to a change in another. For example, one theory specifies that if a component is replaced, then the features of the replacement part might potentially change. This theory is just a compact way of stating that the weight of a replaced component may change, the cost of a replaced component may change, and so forth. Theories allow computer programs to infer relationships rather than just look them up. They let humans specify a generalization rather than enumerate a set of cases. Our manufacturing engineer can now write a subscription to be notified if anything changes in the properties of the structural components that might effect the milling process on the brackets.

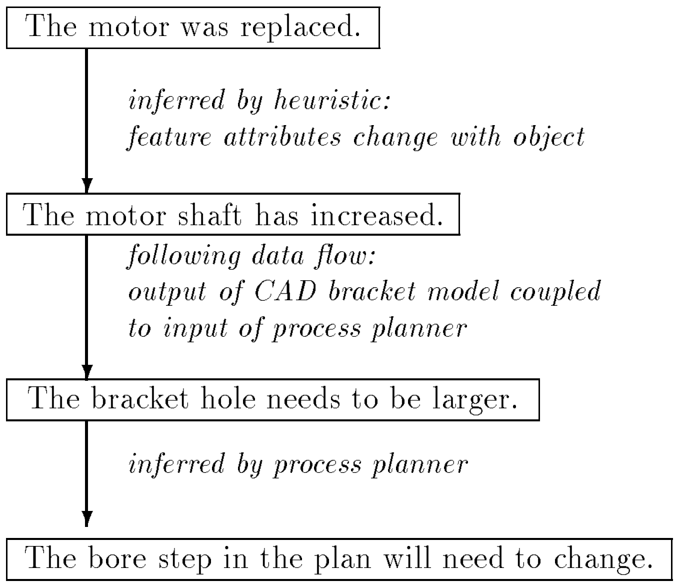
The motor was replaced.
inferred by heuristic:
feature attributes change with object
The motor shaft has increased.
following data flow:
output of CAD bracket model coupled
to input of process planner
The bracket hole needs to be larger.
inferred by process planner
The bore step in the plan will need to change.

Figure 1: Path of relevance reasoning and notification

In SHADE, relevance theories will be used to infer paths of notification in a design knowledge base. Figure 1 illustrates such a path from the planar manipulator scenario described in Section 2.2. The nodes represent changes to design elements. The arcs correspond to inferences about the relevance of changes in one element to changes in another. The first inference could be made by the simple heuristic described above: when the motor was replaced, the motor shaft diameter increased. The second inference relies on a similar heuristic, that associates paired features of mating parts; when the motor shaft diameter changes, the corresponding bracket hole must also be changed. The application of this heuristic is complicated in our scenario because the parts involved are modelled by two different CAD tools (one for simulating the motor assembly and another for creating the process plan for the mounting bracket). The relevance of shafts to holes was determined by a human engineer, and reflected in a model of the data flow between these two tools. In the third inference, a change in the input specification to the process planner leads to a change in the output plan for the milling machine. The fact that changed inputs can lead to changed output might be inferred by a general heuristic. Exactly how the change to the mounting hole will impact the process plan is inferred procedurally by the process planner itself.

5 Research Issues

5.1 The Relative Utility of Ontological Commitment

The SHADE framework has the desirable property that it can provide some level of service at almost any level of knowledge sharing. An absolutely minimal ontology would be a single class of opaque objects and a single kind of relation or link. This is equivalent to conventional hypermedia, where nodes are undifferentiated, unstructured data objects (text, sound, video bites) and the links are untyped relations. Even this minimal ontology allows the linking of design information across engineering tools. The shared knowledge base, object encapsulation, publishing, and subscription technology would all contribute to improved information access. Subscription would be primitive; one could subscribe to specific objects or everything (the latter case is equivalent to hypermedia browsing). This is the level of ontological commitment of design notebooks used to support collaboration.

With a slight enrichment in the ontology, functionality improves rapidly. A weak theory of design is sufficient to identify classes of design knowledge into which to classify new information. Requirements could be differentiated from structure descriptions and the output of simulations could be related to decisions. This kind of ontology could support a design history. A design history is a log of design moves, abstract operations on design elements such as requirements and structures. Capturing design histories in a SHADE-like tool environment could result in a record with a lot of context: the objects, the actors, the tools, and the state of the documents being operated on by the tools. These can be used to selectively retrieve design elements (e.g., "retrieve all structural changes produced by Fred last week using the Cadence schematic editor"). Moreover, if the data were classified and organized into general categories corresponding to their role in the design (e.g., a component selection identified as an "alternative" with respect to a design decision), the resulting record is much easier to index and search. Importantly, the simple ontology works when the instances are interpretable by humans, as with textual and graphical annotation. This is the level of ontological commitment of collaboration tools such as gIBIS [3, 4] and SYBIL [22, 24], which support documentation of group deliberation and decision making based on a simple ontology.

With a commitment to a rich ontology of formally represented design knowledge, the relevance of new information to expressions of interest can be inferred, rather than just matched. In addition, if the vocabulary of design knowledge permits a chain of reasoning from source to effect, then the answers – the information delivered to subscribers – can be structured as well. In particular, systems can be built to explain *how* something is relevant. For example, the torque output by a motor might be computed by an equation linking it to input current and a motor constant. In this case, the relevance of motors to torque is inferred via the equation. The resulting explanation is that the motor replacement changed the motor constant, which resulted in a new torque via the equation. This way, simulators that generate quality explanations for humans [7, 19] can be part of the knowledge medium. For example, explainable simulation can be used to capture specifications of intended function and context of use that are hard to represent formally [14, 15]. In general, if formal reasoning is used to infer relevance, then the resulting proof can serve as an explanation of how something is relevant to the object of interest. This is another way in which the knowledge medium is active; the *invocation* of simulation tools on formal data produces information which can then be published into the shared information spaces.

These potential applications suggest that increasing power results from an increasingly rich shared representation. This is a central hypothesis being explored in the SHADE project: that the utility of the services enabled is a function of the level of commitment to a shared ontology. A family of research questions emerge. What is the minimal ontology sufficient to give useful functionality? What services are enabled by what kinds of shared representation? If we view subscription/notification as information retrieval, do the hit rate and accuracy improve with increasingly rich shared ontology?

5.2 Partial Meta-models

The SHADE approach to knowledge sharing occupies an interesting niche between document and object linking protocols (OLE, Publish and Subscribe, OMG) and complete standardized formal data models used for engineering tool integration (e.g., PDES, CAD Framework Initiative). Both integration approaches play a role in supporting collaborative engineering. We have described a representational framework that achieves the capabilities of both. By encapsulating foreign data, it supports hypertext-like links and browsing among formal objects maintained by heterogeneous engineering tools as well as informal (unstructured) data contained in engineering documents. Since the representation is based on an expressively comprehensive language, SHADE also allows complete specification of shared data models. These specifications are sometime called meta-models [8, 20], since they describe the information content of base-level databases.

The SHADE framework allows us to explore the continuum of integration from content-independent linking to full meta-models. The degree of integration is determined by how much of the information manipulated by participating tools can be represented in the shared representation. We are entertaining the notion that there is a useful middle ground, a partial meta-model, in which some of the data can be represented completely, and the rest can remain tool specific or only interpretable by humans. The hypothesis is that the domain of the shared ontology should be the relationships between data exchanged by tools, and not the knowledge manipulated by the tools internally to produce their results.

5.3 Representing Change in Design Information

We have deliberately been silent on how a SHADE participating tool determines what information is worth exporting. Since the shared ontology is incapable of representing all the distinctions used internally by tools, there will always be a need to determine what is potentially relevant to the world. For the same reason, a subscription cannot tell a tool precisely what aspects of an object to be informed of, since one cannot anticipate all such properties.

For the sake of simplicity we will focus this issue on the case of design change, and assume that there is only only level of granularity of "objects" in a tool that might be mentioned in published facts. In many varieties of object-oriented representation, there is a clear enumeration of those facts about an object which are its properties. One could ask to be notified if any of those facts were stated or retracted, using free variables as needed to generalize the query. For example, to subscribe to changes in a slot of a given object, one says

```
(interested-in me
   '(particular-slot object ?any-value)).
```

To subscribe to a change in *any* slot of a given object, one might write:

```
(interested-in me
   '(?any-slot object ?any-value))
```

These are just two of many possible generalizations of change over an object. If we wanted to know about the *time-varying* change in a parameter of any model of an object, the sentence might look like this:

```
(val (temperature (has-model motor-2 ?model))
     ?time ?value)
```

The point is that what is persistent and immutable – what constitutes a design change or even an object – is tied up with how we conceptualize change and represent facts about changes. If objects are persistent, what are versions? Are facts tagged with version numbers, or associated with versions of objects? How can representations of ordinal time stamps or versions be coordinated across tools?

5.4 Inferring relevance of new information to specified interests

Inferring relevance in general is intractable [26]. However, specialized theories can implement certain kinds of relevance reasoning efficiently. In particular, when the representation is based on opaque, semi-structured objects, and the shared representation of relationships among these objects forms a semantic network in the SKB, relevance can be approximated by a spreading-activation traversal of these relations. For example, if the designer records the relations among the design elements such as requirements, component choice, and simulation results, the relevance of one design element to other shared objects can be viewed in terms of the paths between them in the network. Unfortunately, almost any object can be connected to another over a path of links. One kind of relevance theory theory would specify *which* links are plausibly relevant for a given query, constraining propagation to traverse only these links.[3]

This is only one of many heuristic approaches to determining relevance. There are also several ways one might represent the relevance theory. Framing the relevance question in the SHADE context of a potentially distributed, shared, semi-structured information space raises more interesting questions. Where is the relevance reasoning performed (i.e., at the producer, consumer, or a mediator agent)? Where is the relevance theory kept? How can end users write their own, as an extension of specifying their interests? Results could be applied to the larger contexts of information search over wide area networks. SHADE can provide an excellent test bed for experimentation in this area.

6 Discussion and Plans

We have laid the foundation for a knowledge medium that could support the information and communication intensive activities that dominate cooperative product development. These activities include engineering knowledge capture and retrieval, change notification and active management of design dependencies. The approach places all product knowledge in a coherent framework, relating, for example, requirements, artifact and process models, design decisions, test results, and engineering knowledge. Designed to integrate human organizations as well as design tools, it accommodates information in formats ranging from unstructured multimedia documents to formal engineering models.

Conventional approaches to integrating engineering knowledge rely on standardized data structures or unified meta-models, both of which require substantial commitments from tool

[3]This technique has been used successfully in information retrieval tasks [2].

designers. SHADE departs from such approaches in two fundamental ways. First, tools and data are encapsulated rather than unified. Second, tools communicate through a shared representation that is developed incrementally with use.

We are seeding the ontology with general primitives for relating the information produced by existing tools (e.g., text editors, email, on-line engineering handbooks, feature-based CAD, simulators with model libraries) for tasks such as design rationale capture and the management of design dependencies. From the start, users will be able to link objects from their tools in a hypertext style. Even this minimalist node-and-link ontology can support basic information sharing functions such as retrieval and notification. Initially, only a small amount of information used by human engineers will be machine interpretable. Similarly, only a fraction of the information embedded in existing systems will be available externally. As some of those relationships prove useful, they can be formalized and added to the shared ontology. The additional ontological commitments enhance the selectivity of retrieval and notification operations and enable the relevance of indirectly linked information to be inferred. This novel, pragmatic approach to ontology development is motivated by the heterogeneity of existing software and the adaptability of humans.

The ontology and information routing services are being developed in coordination with two other research projects concerned with knowledge sharing. DARPA's Knowledge Sharing Effort is developing mechanisms and conventions for the sharing and reuse of heterogeneous knowledge bases and knowledge-based systems [29]. This project is developing the KIF interlingua, intelligent agent communication protocols, specifications for common representation systems, and mechanisms for developing and maintaining shared ontologies. The Palo Alto Collaborative Testbed (PACT) is an integration of four experimental engineering environments from Stanford and Lockheed, covering such diverse engineering disciplines as process planning [5], software engineering [33], circuit layout [9], and device modeling [19]. PACT provides a test bed for experiments in propagating design changes across human, tool, and even tool-framework boundaries.

Acknowledgements

The local AI and Engineering research community has significantly influenced the ideas in the paper. We wish specifically to acknowledge the contributions of Mark Cutkosky, Richard Fikes, Mike Genesereth, Jay Glicksman, Bruce Hitson, Larry Leifer, Bill Mark, Jim McGuire, Rich Pelavin, and Dan Russell. Participants of the DARPA knowledge sharing effort, particularly Steve Cross, Bob Neches, Mark Stefik, and Gio Wiederhold, have also helped shape our thoughts. Finally, thanks to Morton A. Hirschberg, US Army Ballistic Research Laboratory, for last-minute technical editing.

This work was supported by DARPA prime contract DAAA15-91-C-0104 through Lockheed subcontract SQ70A3030R, monitored by the U. S. Army Ballistic Research Laboratory.

References

[1] John H. Boose, David B. Shema, and Jeffrey M. Bradshaw. Knowledge-based design rationale capture: Automating engineering trade studies. In M. Green, editor, *Knowledge*

Aided Design. Academic Press, London, 1991. In press.

[2] Paul R. Cohen and Rick Kjeldsen. Information retrieval by constrained spreading activation in semantic networks. *Information Processing and Management*, 23(4):255–268, 1987.

[3] Jeff Conklin and M. L. Begeman. gIBIS: A hypertext tool for exploratory policy discussion. In *Proceedings of the 1988 Conference on Computer Supported Cooperative Work (CSCW-88)*, pages 140–152, Portland, Oregon, 1988. ACM.

[4] Jeff Conklin and KC Burgess Yakemovic. A process-oriented paradigm for design rationale. *Human Computer Interaction*, 6(3/4): , 1991.

[5] M. R. Cutkosky and J. M. Tenenbaum. Providing computational support for concurrent engineering. *Systems Automation, Research and Applications*, 1(3): , 1991.

[6] Brian Falkenhainer and Ken Forbus. Compositional modeling: Finding the right model for the job. *Artificial Intelligence*, 51:95–143, 1991.

[7] Kenneth D. Forbus and Brian Falkenhainer. Self-explanatory simulations: An integration of qualitative and quantitative knowledge. In *AAAI-90*, pages 380–387, Boston, 1990.

[8] James A. Fulton. The semantic unification meta-model: technical approach. Standards working document ISO TC184/SC4/* WG3 N 81 (P 0), IGES/PDES Organization, Dictionary/Methodology Committee, 1991. Contact James Fulton, Boeing Computer Services, P. O. Box 24346, MS 7L-64, Seattle, WA 98124-0346.

[9] Michael R. Genesereth. Designworld. Technical Report Logic-89-3, Stanford University, Computer Science Department, 1989.

[10] Michael R. Genesereth. DSL reference manual. Technical Report Logic-90-3, Computer Science Department, Stanford University, 1990.

[11] Michael R. Genesereth and Richard E. Fikes. Knowledge interchange format, version 3.0 reference manual. Technical Report Logic-92-1, Stanford University Logic Group, 1992.

[12] Michael R. Genesereth and Jay M. Tenenbaum. An agent-based approach to software. Technical Report Logic-91-6, Stanford University Logic Group, 1991. Under revision.

[13] Robert Givan, David McAllester, and Kevin Zalondek. Ontic91: Language specification and user's manual. Technical report, MIT, 1991.

[14] Thomas R. Gruber. Model-based explanation of design rationale. Technical Report KSL 90-33, Knowledge Systems Laboratory, Stanford University, 1990. Also appears in Proceedings of the AAAI-90 Workshop on Explanation.

[15] Thomas R Gruber. Interactive acquisition of justifications: Learning "why" by being told "what". *IEEE Expert*, 6(4):65–75, 1991.

[16] Thomas R. Gruber. Ontolingua: A mechanism to support portable ontologies. Technical Report KSL 91-66, Stanford University, Knowledge Systems Laboratory, 1992.

[17] Thomas R. Gruber and Daniel M. Russell. Design knowledge and design rationale: A framework for representation, capture, and use. Technical Report KSL 90-45, Knowledge Systems Laboratory, Stanford University, 1990.

[18] Thomas R. Gruber and Daniel M. Russell. Derivation and use of design rationale information as expressed by designers. In Thomas Moran and John H. Carroll, editors, *Design Rationale*. Erlbaum, 1992. In press.

[19] Yumi Iwasaki and Chee Meng Low. Model generation and simulation of device behavior with continuous and discrete changes. Technical Report KSL 91-69, Stanford University, Knowledge Systems Laboratory, 1991.

[20] T. Kiriyama, F. Yamamoto, T. Tomiyama, and H. Yoshikawa. Metamodel: An integrated modeling framework for intelligent cad. In John S. Gero, editor, *Artificial Intelligence in Design*, pages 429–449. Computational Mechanics Publications, Southampton, UK, 1989.

[21] Tom A. Kuffner and David G. Ullman. The information requests of mechanical design engineers. *Design Studies*, 12(1):42–50, 1990.

[22] Jintae Lee. Sibyl: A qualitative decision management system. In P. Winston and S. Shellard, editors, *Artificial Intelligence at MIT: Expanding Frontiers*. MIT Press, 1990.

[23] Jintae Lee and Kum-Yew Lai. A comparative analysis of design rationale representations. Technical Report 91-121, MIT Center for Coordination Science, 1991.

[24] Jintae Lee and Kum-Yew Lai. What's in design rationale. *Human Computer Interaction*, 6(3/4): , 1991.

[25] Douglas B. Lenat, Ramanathan V. Guha, Karen Pittman, Dexter Pratt, and Mary Shepherd. Cyc: Toward programs with common sense. *Communications of the ACM*, 33(8):30–49, 1990.

[26] Alon Levy. Irrelevance in problem solving. Technical report, Stanford University, Knowledge Systems Laboratory, 1992.

[27] Mitchell D. Lubars. Representing design dependencies in an issue-based style. *IEEE Software*, pages 81–89, 1991.

[28] T. Malone, K. Yu, and J. Lee. What good are semi-structured objects? adding semiformal structure to hypertext. Sloan Working Paper Tech. Report 3064-89-MS, MIT, 1989.

[29] Robert Neches, Richard Fikes, Tim Finin, Thomas Gruber, Ramesh Patil, Ted Senator, and William R. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):16–36, 1991.

[30] Richard S. Palmer and James F. Cramer. SIMLAB: Automatically creating physical systems simulators. Technical Report TR 91-1246, Department of Computer Science, Cornell University, 1991.

[31] Charles Petrie. Planning and replanning with reason maintenance. Technical report, MCC, 1991.

[32] Mark Stefik. The next knowledge medium. *The AI Magazine*, 7(1):34–46, 1986.

[33] Jay Weber, Brian Livezey, James McGuire, and Richard Pelavin. Spreadsheet-like design through knowledge-based tool integration. *International Journal of Expert Systems: Research and Applications*, 1992. To appear.

[34] Brian C. Williams. A theory of interactions: unifying qualitative and quantitative algebraic reasoning. *Artificial Intelligence*, 51(1-3):39–94, 1991.

[35] Stephan Wolfram. *Mathematica: A System for doing Mathematics by Computer.* Addison-Wesley, 1991.

[36] B. P. Ziegler, M. Elzas, and T. Oren, editors. *Modelling and Simulation Methodology: Knowledge Systems Paradigms.* Elsevier North Holland, 1989.