SHADE: Technology for knowledge-based collaborative engineering

D. R. Kuokka[1], J. G. McGuire[1], R. N. Pelavin[1], J. C. Weber[2], J. M. Tenenbaum[2], T. Gruber[3] and G. Olsen[3]

[1]Lockheed Atificial Intelligence Center
3251 Hanover Street
Palo Alto CA 94304-1191 USA
[2]Enterprise Integration Technologies
459 Hamilton Ave, Suite 100
Palo Alto CA 94301 USA
[3]Knowledge Systems Laboratory
Stanford University
Stanford CA 94305 USA

Abstract

Information sharing and decision coordination are central problems for collaborative product development and enterprise-wide coordination. Designers, manufacturing engineers, and marketing engineers need to assess the impact of their decisions and notify affected parties as the product evolves. Yet, existing CAD tools tend to isolate information at tool boundaries, or make overly-strong commitments to an all-encompassing common model. Furthermore, there is often no automated support outside of the design function. The SHADE (SHAred Dependency Engineering) project is working on knowledge-based methods to improve the communication in the product development process. There are three main components of SHADE: a shared knowledge representation (language and domain-specific vocabulary); protocols for information exchange enabling change notification and subscription; and facilitation services such as content-directed routing and intelligent matching of information consumers and producers. SHADE is being applied to several real domains, including the Palo Alto Collaborative Testbed.

1 Introduction

Product development is a complex, knowledge-intensive process typically carried out by a team working from multiple perspectives. The decisions made by one team member, working on one aspect of the product, usually have significant impact on other team members. Furthermore, knowledge possessed elsewhere in the team is often needed to make more informed decision. Thus, a serial model of product development, where the emerging design passes from one engineering function to another is often inadequate. A more interactive, coordinated approach is needed, giving rise to the fields of concurrent engineering and enterprise integration.

At the heart of effective concurrent engineering is communication. 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, and getting the information to the right place is an essential prerequisite. Designers 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 other aspects of the design changes. Otherwise, unanticipated interactions may lead to expensive last-minute redesigns.

While computers are used extensively in product development, existing tools do little to facilitate information sharing and coordination. In fact, current tools often aggravate the problem by isolating information at tool boundaries, creating islands of automation. Most computer tools support specific tasks in engineering (e.g., geometric modeling, analysis), manufacturing (e.g., process planning, scheduling) or business (e.g., cash flow analysis). However, the transfer of relevant information from one tool to another is sometimes impossible. Often, the only 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. Due to this inefficiency, designers end up making 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 rarely documented effectively. For team-oriented product development to advance, the communication must be improved.

This paper gives an update on the SHADE project [Gruber et al. 1992, McGuire et al. 1992], a joint effort by the Lockheed AI Center, Stanford Knowledge Systems Lab, and Enterprise Integration Technologies. The SHADE project is primarily concerned with the information sharing aspect of the concurrent engineering problem. Our solution is to provide a medium that allows designers, through their tools, to accumulate and share engineering knowledge spanning the functionality of individual tools. This involves more than networking the tools together (to be practical, the design tools need to remain distributed due to the complexity inherent within individual design aspects). First of all, a common knowledge-level representation, that spans the intersection of all the engineering tools, is required. Second, since the exact information needs of one team member cannot be known by another, there must be a means by which information needs and capabilities can be exchanged and acted upon. Finally, since the overhead of increased communication can be large, there must be a set of services that facilitate the communication, off-loading the burden on the individual team members and tools.

Of course, a communication infrastructure is not useful without a set of tools to populate it, and a tool must assume certain responsibilities before being capable of interoperating within the SHADE environment. The tool must understand the common knowledge representation, speak the knowledge transfer protocols, and be able to make use of the facilitation services. A tool that satisfies these requirements is said to be an agent. Therefore, the general approach of SHADE is termed agent-based integration.

In fact, SHADE is one project within a larger cooperative community looking at related issues. PACT [Cutkosky et al. 1993] is a landmark demonstration of both the collaborative research effort and of agent-based technology. Work on federation architectures and

agent-based software engineering [Genesereth 1992] has served as a basis for much of the research in this area. The DARPA Knowledge Sharing Initiative [Neches et al. 1991, Patil et al. 1992] is a community-wide effort to provide an adequate representational framework for many projects. The Knowledge Centered Design project [Kuokka et al. 1993] is focusing more closely on the problem of transforming existing tools into agents that are capable of communicating via the SHADE infrastructure. Another project at Lockheed, called Cosmos [Mark et al. 1993], is focusing on providing support for negotiation and commitment reasoning within the SHADE infrastructure. Finally, the SHARE project [Toye et al. 1993] is looking at a wide range of information exchange technologies in order to help engineers and designers collaborate in mechanical domains.

2 Approach

Three basic components are embodied in the SHADE approach to agent-base integration, corresponding to the three requirements outlined above. First a common vocabulary must be defined to allow tools to express shared dependencies among themselves. Second, a set of protocols of interaction must be defined that permit advanced knowledge sharing. Finally, a set of basic facilitation services is required.

2.1 Shared Representation to Bridge Tool Perspectives

The first component of SHADE is a shared ontology: a formal specification of a shared conceptualization that provides the representational vocabulary with which agents can communicate [Gruber 1993]. The need for a shared ontology is a direct result of the multi-disciplinary nature of engineering. There are many different views of a design (function, performance, manufacturing), each with a largely different language. However, the various perspectives typically overlap, necessitating the sharing of information if design is to proceed concurrently and cooperatively. For information to be shared, there must be a commonly understood representation and vocabulary.

Whereas the language must be demonstrated as being expressive enough to bridge relationships among participating agents used in multi-disciplinary design activities, this does not imply that it must be capable of expressing the union of all distinctions made by participating agents. Many portions of a design space are of interest only to one agent, while other portions must be common to many agents. The challenge is to support different degrees of knowledge sharing, from arms-length knowledge exchanges to strong common models.

SHADE acknowledges this range of knowledge sharing by presupposing an incremental evolution of language that allows the encoding of progressively richer dependencies across tools. The language evolution would proceed from an encoding of simple dependencies among opaque elements ("object X is in some way dependent on object Y") to the gradual introduction of common models ("$Y.b = 2 * X.a + 3$") to explanations of causality ("X caused Y to fail"). This evolution would enable increasingly sophisticated types of change notification and interaction among designers. Of course, it also imposes greater demands

upon the supporting communications infrastructure.

To better support the development of shared ontologies, SHADE is working on systems and techniques for building ontologies, and applying them to construct specific vocabularies for engineering. To establish conventions, promote rigor, and facilitate enhancement and extensibility, ontologies are defined within a widely accepted, formally defined representation, and the related vocabulary is modularized into hierarchical theories. The representation, tools, techniques, and theories are discussed below.

2.2 Protocols for Coordination Among Tools

The second major component of SHADE follows from the realization that team members are inherently distributed in the real world of engineering. A set of protocols is needed to guide the knowledge transfer within the shared environment, thus enabling a diverse user community to work together effectively.

A major part of the SHADE effort is an ongoing contribution to the evolving Knowledge Query and Manipulation Language (KQML) [Finin et al. 1992] being defined by the External Interfaces working group of the DARPA Knowledge Sharing Initiative. KQML is an agent communication language whose message types have specific semantics and impose constraints on the agents uttering the messages. The protocols of interaction are *embodied within* the semantics of individual message types. That is, agents communicate effectively by abiding by the constraints imposed upon their behavior by the semantics of the messages they issue.

KQML only provides a protocol by which agents can communicate their attitudes (belief, interest, expectation). It makes no commitments to *policies* such as honesty, accuracy or completeness. That is, an agent may assert a piece of information without actually having support for the belief. However, for effective communication to occur, a collection of appropriate policies will have to be in effect. SHADE is actively investigating policies, on top of KQML, that support effective communication among agents.

By way of example, consider a design tool interacting with other tools within the SHADE infrastructure. The elements of the shared design model are actually distributed among the individual agents. Each individual agent maintains a portion of the shared set of beliefs (either embedded within tool-specific data structures or contained within the wrapper's knowledge base). To be recognized as an information service contributing part of the virtual knowledge base, an agent would use KQML to advertise its information-producing potential, or capabilities. Conversely, an agent would use KQML to declare its interests in relevant portions of the shared model, thereby enabling notification and information matchmaking. Advertisements and interests create virtual shared knowledge-based even though the actual information is distributed.

2.3 Facilitation Agents to Ease the Communication Burden

The third major component of SHADE is an (extensible) suite of core framework services to facilitate communication and coordination among agents, thereby reducing the burden

on individual agents. These services are implemented by special facilitation agents connected to the infrastructure much like application agents. Example services include 1) message routing and nameservice, 2) translating between different (locally opaque) representations, 3) subscription and notification services, 4) matching information consumers to producers based on descriptions of their information interests and information-producing capabilities (matchmaking), and 5) providing active constraint management of dependencies spanning several design-tool perspectives.

A representative facilitation service is content-based routing: the intelligent routing of information among participating tools based on previous subscriptions. This allows an agent to send a message without having to determine exactly which other agents are interested. As networks of agents get larger, this becomes a necessity as opposed to a convenience.

For content-based routing to work, however, agents must post every piece of information that they derive (that is translatable into the shared representation). This makes it feasible for a logically centralized facilitator to check all new information against interest criteria and route information appropriately. The flaw in this scheme is that an agent can produce many types of information and (assuming a pay for a service scheme) probably will not post information unless someone has explicitly requested it. What is needed is a facilitation service that locates information producers capable of fulfilling outstanding interests and manages subscriptions to these information producing services so that needed information is posted. This would ease the burden on individual agents since they could rely on the infrastructure to locate service providers. Such a facilitation agent is called a matchmaker, and is discussed in more detail below.

3 SHADE Technology Elements

The above elements of SHADE have been under active investigation. Several specific, albeit partial, solutions have been developed so far. This section outlines the major results to date.

3.1 Representation of Shared Design Knowledge

An ontology is an explicit specification of a conceptualization. A conceptualization is defined by the objects, concepts, and other entities that are presumed to exist in some area of interest and the relationships that hold among them [Genesereth and Nilsson 1987]. A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Since the set of objects and relationships in an agent are reflected in the representational vocabulary, an ontology can be given as a set of definitions for this shared vocabulary.

The common ontology, and the sentences using that ontology, must be representable in a implementation- and agent-independent format. The format used in SHADE is called KIF (Knowledge Interchange Format) [Genesereth and Fikes 1992], which is a product of the DARPA Knowledge Sharing Initiative. KIF is a machine-readable version of first order

predicate calculus, with extensions to enhance expressiveness. KIF is the content language for the SHADE framework. The KIF specification defines the syntax and semantics; the ontologies define the problem-specific vocabulary; agents exchange sentences in KIF using the shared vocabulary.

Since individual engineering tools have specialized internal formats, there is a need to maintain portable ontologies by providing translation mechanisms into and out of a shared representation. Consequently, SHADE is using the Ontolingua [Gruber 1992] system to develop and maintain its ontologies. Ontolingua provides a layer on top of KIF for writing definitions and translating them into implemented representation systems. Ontolingua can translate class, relation, function, and object definitions into several knowledge representation systems. It specializes in translating to object-centered representations, such as those used in frame systems and object oriented data description languages.

To ground our efforts in ontology construction, we have focused on the exchange of knowledge between several key perspectives in mechanical design, such as the exchange between a controls engineer and a rigid body dynamics specialist. This exchange is quite general in that it consists primarily of mathematical expressions relating commonly understood variables. Mathematical expressions are the basic language of engineering analysis. Engineers use mathematical models, such as equations, to analyze the behavior of physical systems. While standard notations exist in textbooks and other technical literature, these notations fail to capture much of the meaning intended by the modeler and leave implicit many of the details required to understand the equations. Much information is not formally stated in an expression like "$x + 10ycos(q) = T$", such as which symbols are variables or constants; whether the numbers are reals or vectors; whether the modeled quantities are static values, functions of time, or functions of time and space; and if there are units of measure assumed, what they are. Also distinctions between the symbol x (i.e. a reference to an attribute of a design feature) and the quantity it denotes (i.e. the actual value of the attribute) are almost never apparent.

An ontology written using Ontolingua is modularized into theories to promote sharing. Each theory formalizes sets of classes, functions, relations, and axioms (possibly in terms of other theories) to enrich a shared vocabulary. Each theory can serve as a "building block" by defining terms that can be assumed in derived theories. As an example of how theory modularity can be exploited, consider the theory inclusion graph of some existing Engineering Math ontologies developed under SHADE (Figure 1).

A link between theories indicates that the lower theory relies on terms defined within the upper theory. The Engineering Math ontologies are an evolving set of Ontolingua generated theories created to provide a language which can declaratively capture the semantics associated with mathematical expressions describing physical quantities. The theories focus on algebras relating engineering parameters. For our discussions, we will focus on the Physical Quantities theory. The remaining theories deal with algebraic classification of parameters and the mathematical operators (e.g. addition) associated with them.

In engineering analysis, physical quantities such as 'the speed of light' are regularly modeled by variables in equations with numbers as values. While human engineers can interpret these numbers as physical quantities by inferring dimension and units from con-

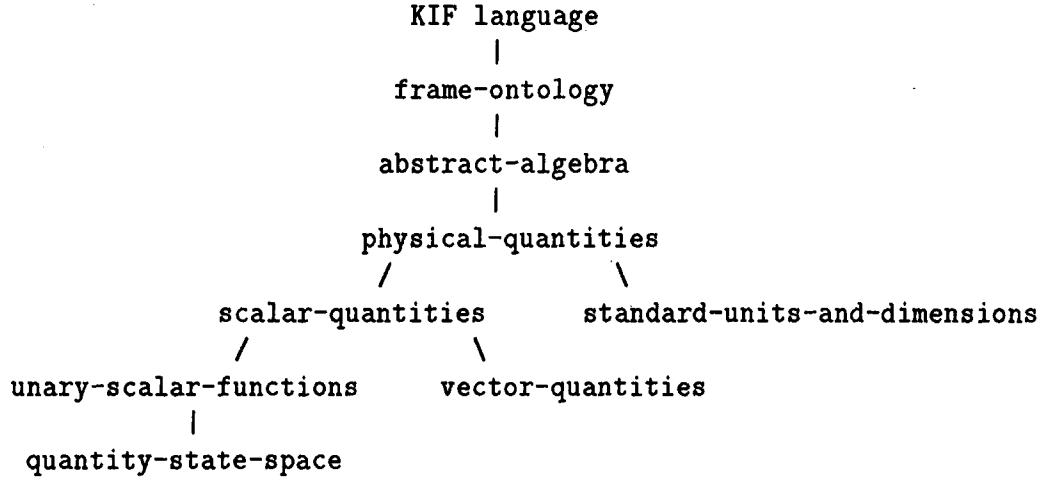

Figure 1: Inclusion graph for Engineering Math Ontologies

text, the representation of quantities as numbers leaves implicit other relevant information about physical quantities in engineering models, such as physical dimension and unit of measure. Furthermore, there are many classes of models where the magnitude of a physical quantity is not a simple real number — a vector or higher-order tensor for instance. Our goal here is to extend standard mathematics to include unit and dimension semantics.

In the Physical Quantities theory, we define the basic concepts associated with physical quantities. A quantity is a hypothetically measurable amount of something. We refer to those things whose amounts are described by physical-quantities as physical-dimensions (following the terminology used in most introductory Physics texts). Time, length, mass, and energy are examples of physical-dimensions. Comparability is inherently tied to the concept of quantities. Quantities are described in terms of reference quantities called units-of-measure. A meter is an example of an unit-of-measure for quantities of the length physical-dimension.

The physical-quantities theory defines the basic vocabulary for describing physical quantities in a general form, making explicit the relationships between magnitudes of various orders, units of measure and physical dimensions. It defines the general class physical-quantity and a set of algebraic operators that are total over all physical quantities. Specializations of the physical-quantity class and the operators are defined in other theories (which use this theory).

The theory also describes specific language for physical units such as meters, inches, and pounds, and physical dimensions such as length, time, and mass. The theory provides representational vocabulary to compose units and dimensions from basis sets and to describe the basic relationships between units and physical dimensions. This theory helps support the consistent use of units in expressions relating physical quantities, and it also supports conversion of units needed in calculations.

As examples of legal informational content, suitable for containment within a (KQML)

message (e.g. to convey expression of belief), consider:

```
(physical-dimension length)

(unit-of-measure inch)

(= (q.dimension inch) length)

(quantity (diameter shaft-a))

(= (diameter shaft-a) (* 3.6 inch))

(= (q.magnitude (diameter shaft-a) feet) 0.3)
```

The unit-of-measure "inch" and the physical-dimension "length" are introduced in the first two sentences. The dimension associated with a unit of measure is accessible via the "q.dimension" relation. Using this relation, in the third statement, the dimension of "inch" is assigned be equal to "length". In the fourth sentence, the expression denoted by "(diameter shaft-a)" is classified as a "quantity". In the fifth sentence, the value of shaft-a's diameter is equated with the quantity returned by the product $3.6 * inch$. Quantities are introduced using a specific magnitude and unit of measure, but can be described via other units of measure compatible in dimension. For example, in the sixth sentence, the "q.magnitude" relation allows the magnitude of the quantity assigned to shaft-a's diameter to be described in terms of the "feet" unit of measure - even though the quantity was defined in terms of inches.

Other theories under construction (which are derived from the Math theories) include:

- State Spaces: This theory provides a language for describing relationships among variables which are time dependent. An ordinary differential equation is a typical relation. (This is the strongest theory of commonality between the controls and dynamics specialists in the PACT scenarios described below.)
- Kinematics: This theory will provide a language for describing the relative orientation and position of reference frames and points in space.
- Rigid Body Dynamics: This theory will provide a language for describing the dynamical motion of bodies. Newton-Euler and Kane's formulations will be supported.

Another theory, non-mathematical in nature, which exists is:

- Component Hierarchies: This theory provides a language for describing hierarchically connected components, representing component connections and part/subpart relationships.

3.2 Protocols of Interaction

Sophisticated programs, especially knowledge-based agents, interact in many ways beyond the simple query-response paradigm of standards like SQL. Without an effort to understand and standardize agent communication, there would be a proliferation of incompatible, ad hoc agent communication languages. The SHADE project is actively working toward the establishment of a common agent communication language. Members of the SHADE team are key participants within the DARPA Knowledge Sharing Initiative, which is working on the definition of the Knowledge Query and Manipulation Language (KQML). SHADE is also one of the most active users of KQML.

KQML is a language for programs to use to communicate to the recipient what the sender's attitude is toward the informational content, where the content is encoded in a specified ontology. Examples of attitudes on content are:

- believing/disbelieving the information content
- indicating interest in the class of information described by the content
- advertising a capability to produce the class of information described by content
- expecting that the class of information described by the content will be promised
- fulfilling with the informational content some past commitment (i.e. a reply to a query)

KQML is also an enabler of information-flow architectures, through forwarding, broadcasting, and brokering messages. That is, an agent need not send every message directly to the final destination. Often an agent does not know exactly who should process a message, or even if an appropriate recipient exists. Forwarding and broadcasting allow the sending agent to use intermediaries to facilitate transport. Brokering allows the agent to be unconcerned with exactly who can process the message. The broker facilitation agent is responsible for finding an appropriate recipient for the request.

To enable agents to use KQML, the SHADE project has defined an initial API (application programmer interface) through which agent programs can send and receive messages. The API supports sending and receiving of KQML messages as either strings or "s-expressions". The API is currently available for C, and a LISP version is under development.

When an agent expresses an attitude in the form of a KQML message, there must be some understanding of exactly what the message means. The elaboration of the semantics of KQML attitude types [Genesereth et al. 1992] is an area of collaborative research by SHADE. To provide a semantics, an agent's internal state is abstractly modeled by its beliefs and goals. The semantic meaning of KQML message types are then modeled by the constraints they impose on the sending agent's abstract model of internal state. A recipient agent's internal state can potentially be affected by the attitudes of a sender (e.g. the recipient agent may choose to believe knowledge content because someone else believes it).

The KQML semantics elaboration effort is also characterizing how the abstract internal state of an agent constrains externally visible behavior (in the form of attitude utterances in messages) of the agent. Other examples of semantic constraints are:

- when an agent asserts a belief in some content, the internal state of the agent should reflect that belief.
- when an agent accepts the responsibility of providing a needed piece of information, the agent's internal goal state should reflect the desire to meet the responsibility, and consequently the agent's future (externally visible) behavior should live up to the promise.
- when an agent expresses the attitude of fulfilling the informational request of another agent, it should be the case that the agent believes that the other agent wanted someone to provide the information.

3.3 Message Transport

Message-passing is the glue that builds large software systems out of multiple smaller component systems. There are many new standards and toolkits that support the transport of messages among programs (e.g. OMG CORBA, OSF DCE, ISIS, BSD Sockets, etc). Even though the SHADE project is not focused on increasing this set, a reliable, easy-to-use transport mechanism is an important prerequisite of inter-agent communication. To satisfy our specific needs, the SHADE project has produced the EIT ServiceMail (tm) toolkit, which enables the creation of engineering services (rendering, layout, etc.) that are accessed using multimedia electronic mail.

The idea of ServiceMail is that of sending e-mail to programs in order to request some automated or semi-automated service. There have been several applications along this line accessible through electronic mail, including services for archival searches, mailing list manipulation, conference information, etc. Each of these applications use electronic mail in the following way: users send their requests in the headers and/or bodies of e-mail messages, including any relevant data files (e.g. a semiconductor layout representation). These applications have been very popular, largely due to the ease of e-mail access and simplicity of use. SHADE has experimented with several new applications of ServiceMail, including:

- a semiconductor fabrication process simulator service at Stanford
- a mechanical part checkplot/milling service at the Univ. of Utah
- a means of enabling corporate (e.g., Lockheed) participation in SHADE experiments. Many big companies disallow bidirectional TCP/IP internet access, for security reasons. E-mail is often the only bidirectional transport mechanism available.

SHADE has been fostering ServiceMail by freely distributing a toolkit that helps service providers get their services on-line. The current toolkit provides software to take

care of the overhead in establishing a service from scratch, including: connecting to the e-mail system, parsing incoming messages, multiplexing multiple services under a single mail address, and generating outgoing messages. Current work on the toolkit is addressing other issues such as: integrity checks on message contents, authentication of users, auditing of service requests and responses, directory services to locate service advertisements.

3.4 Facilitation Services

The SHADE infrastructure is designed to include facilitation agents that provide communication support to agents. One of the first necessary services that has been identified is content-based routing, where messages sent by an agent are routed automatically based on interests asserted by other agents. To achieve this, agents must be able to express their interests in terms of a general subscription. In the simplest case, a subscription looks like a syntactic pattern that must be unified against the message. More generally, it looks like an arbitrarily complex first-order logic condition which must be evaluated for satisfiability. An agent subscribes to information by posing a persistent query within the vocabulary of the common ontology. When relevant information is published by some other agent, it is picked up by the routing agent and forwarded on to the requesting agent. In this way, the SHADE environment is able to determine the routing of messages based upon the content of the message, rather than a wired-in prearranged address.

However, as mentioned above, content-based routing assumes that agents volunteer all information, an assumption that cannot be supported as the network grows and financial issues are considered. To overcome this problem, we developed the notion of matchmaking. Instead of routing individual messages sent by other agents, a matchmaker actually matches up advertisements and interests. This allows the matchmaker to establish connections between the producer and consumer.

For example, consider the following scenario. Agent A1 has an interest in a class of information, which it wishes to have fulfilled by another agent. It posts to the shared environment a request to monitor all future changes in the specified class of information. The shared framework, itself composed of facilitation agents, records A1's interest and determines whether the message is of interest to any other agents. A matchmaker service agent has posted an interest in other agents' interests (i.e. it wants to know what other agents want to know) and consequently is notified of A1's interest. The matchmaker service attempts to match the interest with any advertised capabilities. A match is found, the facilitation service subscribes to the relevant capability to enable notification. The capability provider, agent A2, schedules a goal to provide the service. Later, when a change occurs to the interesting class of information, the interest is incrementally fulfilled and A1 is notified.

The intention is to support knowledge based relevance reasoning to infer matches between specifications of information interests and information producing capabilities. In our experiences, it is easier to specify an interest (which looks like a knowledge base query) than it is to precisely characterize an agent's capabilities (which needs to account for all possible interest requests). Even if it were possible to precisely characterize all conditions

on capabilities, prudence dictates that the specification be overly general to allow for the development of tractable algorithms that infer matches on interests.

As an example of the progression toward knowledge-based relevance determination, consider the case where several tools exist — each capable of producing simulation results for the aspect of the design modeled from their perspective. Also assume that the shared language supports a representation for time varying data in the form:

```
(= (val ?component ?port ?time) ?value)
```

Within this format, all of the simulation agents characterize their capabilities via an extremely over-generalized specifications ("I can tell you the values of all ports of all components at any time"):

```
(advertise :content '(= (val ?component ?port ?time) ?value))
```

Conversely, some information consuming agent asks to monitor all simulation results for a specific port:

```
(monitor :content '(= (val c1 p1 ?time) ?value))
```

In order to satisfy the request, the matchmaker would infer matches with all of the overly-general capability specifications and would then request subscriptions with each simulation agent in turn until one accepted the subscription. That subscription would then be forwarded to the requesting agent.

Another possibility would be for each agent to enumerate every component-port pair for which it was capable of producing simulation results. This strategy could lead to overly verbose capability specification on large designs with many components. If, however, the common vocabulary introduced a rich inheritance language, agents could announce capability patterns annotated with type restrictions on the individual arguments. This would allow simulation agents to partially describe their capabilities without referring to specific instance identifiers (object references) contained within a design. For example, a simulation package with access to a rich model library would describe the types of components it could simulate (not the component instances themselves). Thus, there would be a natural separation between design-specific instance identifiers and persistent types that are instantiated across many designs. This would allow for more precision in capability specifications and better matchmaking, while remaining in the realm of tractable algorithms (i.e. classification using taxonomic reasoning [MacGregor 1991]).

The matchmaking service also allows for more efficient communication by alleviating message traffic bottlenecks through a centralized facilitator. Since information sources and sinks find each other during a preprocessing phase, they can communicate directly with each other thereafter. The alternative is to continually infer relevance based on subscriptions during actual information exchanges. A precompilation of message traffic dataflow is extremely important in some engineering design scenarios. Experiments were conducted based on distributed simulations over the Internet bridging several tool perspectives. Performance was considerably slowed in those cases where relevance determination was performed on each message (via a unification-oriented subscription service) and routed based on content by a centralized facilitation mechanism.

4 Applications

To drive our research, SHADE is focusing on specific design domains and scenarios. This section describes these.

4.1 PACT

SHADE's initial application focus was on PACT (Palo Alto Collaborative Testbed), a set of experiments in exercising knowledge sharing techniques among four existing concurrent engineering tools [Cutkosky et al. 1993]. The experiments involved four geographically and organizationally distributed engineering teams (Lockheed, Stanford, Enterprise Integration Technologies, and Hewlett-Packard) collaborating on scenarios of design, fabrication, and redesign of a planar robotic manipulator. Each team modeled a different aspect of the manipulator from a different engineering discipline: controller software (NVisage [Weber et al. 1992]), rigid body dynamics (NextCut [Cutkosky and Tenenbaum 1992]), circuitry (DesignWorld [Genesereth 1991]), and sensors and power system modeling (DME [Iwasaki and Low 1993]). Several collaborative design tasks were performed including dynamics model exchange between the controls agent and dynamics agent, fine-grained cooperative distributed simulation exercising each aspect supported by the four tools, and finally design modifications suggested by the simulation.

The challenge in PACT was to take four existing systems, each already a specialized framework, and to integrate them via a flexible, higher-level framework. To ground the experiments, design scenarios that would be thwarted by tool isolation were proposed. The developers of the various tools identified the information exchange necessary to enable the design scenarios. As a result of these interactions, an implicit ontology was created reflecting offline agreements. Next, each tool was wrapped up as an information agent available as a service to other agents. Over the course of the PACT experiments, the ontologies were explicitly encoded in KIF. After the form and semantics of the knowledge content had been agreed upon, KQML was used to allow expressions of attitude toward knowledge content such as belief, disbelief, and interest. Each team was supported by its own computational environment linked via the PACT framework over the Internet.

The PACT experiments demonstrated a wide variety of advanced variety of inter-agent communication. The PACT framework utilized an infrastructure postal service to allow agents to delegate all message delivery responsibilities [Genesereth 1992, Singh and Genesereth 1992]. The postal service is capable of handling "forward" messages addressed to any registered agent, and agents can query the postal agent to determine all other online registered agents. Other message traffic was permitted in the form of a point-to-point package between agents to reduce the overhead of a centralized bottleneck (e.g. during a distributed simulation).

Since the PACT experiments were meant to represent a concurrent engineering approach, it was critical that all affected parties of design changes be notified so they may assess the impact. Consequently, heavy use was made of the "subscribe" message type to convey the conditions triggering a notification. As a simple example of the utility of "subscribe" within PACT, one agent (NextCut) posts a persistent interest in the type

of motor applying torque to the manipulator arms. This way if the motor changes, the consequences of the change in the motor's features can be evaluated.

The flow of information was also subject to control in the experiment. Some tools in the PACT experiment were able to handle asynchronously transmitted partial information (one answer at a time) by while other tools required entire sets of answers bundled together. To provide flexibility in the packaging of transmitted knowledge, message types were devised to convey differing requirements on the form of the answers to requests.

The PACT experiments show how heterogeneous systems can exchange information and services as if through a shared knowledge base, even though most of the knowledge resides in the internal representations of individual tools. However, the experiment was not a complete demonstration of SHADE. The PACT architecture did not provide advanced facilitation services, such as matchmaking or sophisticated routing of information based on content. Instead all such facilitation sophistication resided directly within each individual design agent's wrapper, making wrapper development onerous. Future PACT experiments will shift functionality into the infrastructure to provide mechanisms for locating registered agents with capabilities suited to fulfilling specific information interests.

4.2 MACE

As a more complete demonstration of SHADE, and to flesh out issues not covered in PACT, the SHADE project has been working on a more in-depth incarnation of the PACT scenario. The scenario is based on the Mid-Deck Active Controls Experiment (MACE), a research prototype intended to be flown inside of the Space Shuttle Mid-Deck to study the use of active body members on vibration control in satellites. The scenario is built around the interaction of three real enginering tools: SDRC's I-DEAS (structural dynamics analysis), ISI's Matrix-X/SystemBuild (controls), and a rigid body dynamics analysis tool based on Mathematica. By focusing on real tools as applied to a real problem, SHADE will be able to validate current concepts and will be driven to enhance concepts were most needed.

4.3 VT

The idea of using formal ontologies as specifications of common conceptualizations is also being applied in an experiment by the knowledge acquisition community called Sisyphus/VT [Linster 1992]. Sisyphus is the name of of an ongoing experiment in which participants from several research groups are building knowledge system architectures and knowledge acquisition tools for specific problems in order to compare their methods, architectures, and results. The previous year's problem was a simple office-assignment task. This year's problem is elevator design, a configuration problem with many parameters and constraints. This domain is well understood, and has been the subject of research such as VT [Marcus et al. 1998], an expert system for elevator design, and the SALT system for knowledge acquisition [Marcus and McDermott 1989]. The task and domain have also been analyzed and a thorough English-language description has been published [Yost 1992]. For the purposes of the Sisyphus/VT experiment, the domain is

complex and large enough to preclude simplistic approaches (brute-force search or simple optimization techniques).

The SHADE team, in collaboration with colleagues from the knowledge acquisition community, is developing a set of formal ontologies that describe the configuration design problem and the elevator design domain. The ontologies will serve as a formal problem specification for the experiment (some participants will build agents that commit to the ontologies; i.e., will be able to accept inputs and provide outputs using the formal vocabulary and theory of the ontology). From early analysis it appears that the much of the theory underling the VT ontologies can be inherited as a specialization of the engineering math ontologies produced by the SHADE project for the exchange of behavior models and data. The design task adds the additional requirements of representing structure and design constraints declaratively.

5 Summary and Future Work

The SHADE project is creating technologies to promote information sharing among design tools within multi-disciplinary design environments. Our strategy is to provide a knowledge-based medium by which designers, through their tools, share engineering knowledge. There are three key ideas to central to the SHADE vision. First, individual design tools will communicate in a common ontology realized in a declarative representation. Second, agents will make use of knowledge transport protocols to coordinate their information needs and capabilities. Finally, the shared environment will be populated with facilitation services to ease the burden of inter-agent communication. Because the shared knowledge is expressed in a formal, declarative language, the shared environment can use deductive mechanisms to answer queries, route information, perform translations, and otherwise facilitate knowledge sharing and software interoperation.

SHADE continues to work on fundamental ontologies and support tools. In addition, we have started work on ontologies and facilitators aimed at design rationale and decision-maintenance [Gruber and Russell 1992, Petrie 1992]. SHADE also continues to be one of the driving forces behind the KQML effort. ServiceMail is gaining increased recognition, and additional enhancements are expected. Finally, SHADE is increasing its efforts toward implementing useful facilitation agents.

Work up to this point has been motivated by experiments in multi-disciplinary design settings such as PACT. This inter-project approach will continue. The SHADE group is now interacting with Lockheed's Space Systems Division to explore application of SHADE technology to support collaborative spacecraft design activities involving dynamics analysis and controls design. These activities are real-world examples of the design scenarios explored in the PACT experiments. SHADE is also being applied to the VT domain to broaden the test of the concepts.

Finally, the community of SHADE users is growing (e.g., KCD and COSMOS), but we are actively pursuing ties with other efforts in the research community to further validate the generality of our approach. The success of SHADE depends largely on the degree to which we can support coordination among numerous, heterogeneous agents.

6 Acknowledgements

We gratefully acknowledge the contributions of our colleagues in PACT, KCD, and the DARPA Knowledge Sharing Initiative, particularly, Bill Mark and Brian Livezey at Lockheed, and Mike Genesereth, Gio Wiederhold, Mark Cutkosky, and Richard Fikes at Stanford. We would also like to acknowledge Morton Hirschberg for valuable technical editing feedback.

7 References

Cutkosky M., and Tenenbaum, J. (1992). Toward a Framework for Concurrent Design. International Journal of Systems Automation: Research and Applications, 1(3).

Cutkosky, Engelmore, Fikes, Gruber, Genesereth, Mark, Tenenbaum, and Weber. (1993). PACT: An experiment in integrating concurrent engineering systems. In IEEE Computer, 26(1).

Finin, Fritzson, and McKay. (1992). An Overview of KQML: A Knowledge Query and Manipulation Language. Department of Computer Science, University of Maryland, Technical Report.

Finin, Weber, Wiederhold, Genesereth, Fritzson, McGuire, McKay, Shapiro, Pelavin, and Beck. (1992). Specification of the KQML Agent Communication Language. Official Document of the DARPA Knowledge Sharing Initiative's External Interfaces Working Group, Enterprise Integration Technologies, Inc. Tech Report 92-04.

Genesereth, M. and Nilsson, N. (1987). Logical Foundations of Artificial Intelligence. Morgan Kaufmann Publishers.

Genesereth, M. (1991). Designworld. Proceedings of the IEEE Coinference on Robotics and Automation.

Genesereth, M. (1992). An Agent-Based Framework for Software Interoperability. Proceedings of the DARPA Software Technology Conference, Meridian Corporation, Arlington, VA. Also Computer Science Department, Stanford University Tech Report Logic-92-2.

Genesereth, M. and Fikes, R. (1992). Knowledge Interchange Format, Version 3.0 Reference Manual. Computer Science Department, Stanford University, Tech Report Logic-92-1.

Genesereth, M. et al. (1992). Semantics of Performatives in ACL, Working Document. Stanford Logic Group Tech-Report.

Gruber, T. (1992). A Translation Approach to Portable Ontology Specifications. In R. Mizoguchi (editor), Proceedings of the Second Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop, Kobe, Japan. To appear in the Journal of Knowledge Acquisition.

Gruber, T., Tenenbaum, J., and Weber, J. (1992). Towards a knowledge medium for collaborative product development. In J.S. Gero, editor, Proceedings of the Second International Conference on Artificial Intelligence in Design, Pittsburgh, PA, pages 413-432, Kluwer Academic Publishers.

Gruber, T. and Russell, D. (1992). Generative design rationale: Beyond the record and replay paradigm. Technical Report KSL 92-59, Knowledge Systems Laboratory, Stanford University. To appear in Thomas Moran and John H. Carroll (editors), Design Rationale, Lawrence Erlbaum.

Gruber, T. (1993). Toward Principles for the Design of Ontologies Used for Knowledge Sharing. International Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation, Padova, Italy.

Iwasaki, Y. and Low, C. (1993). Model Generation and Simulation of Device Behavior with Continuous and Discrete Changes. Intelligent Systems Engineering, 1(2).

Kuokka, D., Livezey, B., Simoudis, E., Hood, J. (1993). Knowledge-Centered Design. Lockheed Artificial Intelligence Center Technical Report.

Linster, M. (1992). Sisyphus '92: Models of Problem Solving. GMD-Arbeitspapiere 630. GMD, SchloB Birlinghoven, Postfach 13 16, W-5205, Sankt Augustin 1. ISSN 0723-0508.

MacGregor, R. (1991). The Evolving Technology of Classification-Based Knowledge Representation Systems. In John Sowa (editor), Principles of Semantic Networks: Explorations in Representations of Knowledge, pages 385-400, Morgan Kaufmann.

Marcus, S., Stout, J., and McDermott, J. (1988). VT: An Expert Elevator Designer that Uses Knowledge-Based Backtracking. AI Magazine, pp. 95-111.

Marcus, S. and McDermott, J. (1989). SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems. Artificial Intelligence, 39(1):1-38.

Mark, Schlossberg, Ogata, MacGregor, Kuokka, Hyde, and Livezey. (1993). The Cosmos System for Distributed Design Negotiation Support. Lockheed Artificial Intelligence Center Technical Report.

McGuire, Pelavin, Weber, Tenenbaum, Gruber, Olsen. (1992). SHADE: A Medium for Sharing Design Knowledge Among Engineering Tools. Lockheed Artificial Intelligence Center Technical Report.

Neches, Fikes, Finin, Gruber, Patil, Senator, and Swartout. (1991). Enabling technology for knowledge sharing. AI Magazine, 12(3), 16-36.

Patil, Fikes, Patel-Schneider, McKay, Finin, Gruber, and Neches. (1992). The DARPA Knowledge Sharing Effort: Progress report. In C. Rich, B. Nebel, and W. Swartout (editors), Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference, Cambridge, MA, Morgan Kaufmann.

Petrie, C. (1992). A Minimalist Model for Coordination. Proceedings of AAAI-92 Workshop on Design Rationale. To appear in Enterprise Modeling, C. Petrie (editor), MIT Press.

Singh, N. and Genesereth, M. (1992). Implementation Details for Agent Based Software Engineering Interoperation. Internal Stanford University Logic Group Tech Report.

Toye, G., Cutkosky, M., Leifer, L., Tenenbaum, J., and Glicksman, J. (1993). SHARE: A Methodology and Environment for Collaborative Product Development. Stanford Center for Design Research Technical Report 1993-0420.

Weber, J., Livezey, B., McGuire, J., and Pelavin, R. (1992). Spreadsheet-Like Design Through Knowledge-Based Tool Integration. International Journal of Expert Systems: Research and Applications, 5(1).

Yost, G. (1992). Configuring Elevator Systems. Technical Report, Digital Equipment Corporation, 111 Locke Drive (LMO2/K11), Marlboro, MA, 02172.