1996 ASME Design for Manufacturing Conf., Irvine, CA, Aug. 18-22. Paper No. 96-DETC/DFM-1302

DESIGN FOR MANUFACTURABILITY VIA AGENT INTERACTION

H. Robert Frost and Mark R. Cutkosky
Center for Design Research
Stanford University
Stanford, California 94305-4021
frost@cdr.stanford.edu
cutkosky@cdr.stanford.edu

ABSTRACT[1]

We describe an approach for making the capability of manufacturing processes manifest to designers starting with the earliest stages of geometry specification. The approach involves a dialogue among design and manufacturing agents over the Internet. The dialogue focuses on the specification and exchange of process capability models for establishing "design rules on-demand" to ensure manufacturability. The models include both declarative knowledge and, for those aspects of the process that are difficult to represent declaratively, platform-independent procedural code which is automatically loaded onto the designers' machines. The approach is being implemented using agents, written in the Java[2] language, which exchange feature-based capability models. The approach is being tested initially on machining and shape-deposition processes.

1. INTRODUCTION

This paper is concerned with the development of a concurrent engineering environment which supports design-for-manufacturability for both standard and novel manufacturing processes. We believe that the successful realization of such an environment requires a mechanism by which designers can evaluate the manufacturability of potential designs according to design rules and evaluation procedures extracted from process capability models. We contend that these capability models need to be developed and maintained by specific manufacturing services and provided on-demand to designers in a machine-readable, platform-independent format.

In this paper, we propose an architecture which meets these needs by encapsulating designers and manufacturing services with software agents that exchange process and design information according to a formal communication interface. In the spirit of concurrent engineering, we want to present this information to the designer as early as possible in the design process, preferably while the basic geometry of the design is being generated and is easily influenced by design rules or guidelines for manufacturing processes.

A particular motivation for this work is the advent of novel prototyping processes such as layered shape deposition [Merz et al. 1994] and laser sintering [Beaman 1993]. These processes have unique capabilities (and a few constraints) which most designers are not familiar with and which are not be included in the DFM modules integrated with some CAD systems. Exploration and utilization of these processes therefore requires capability models which can remotely be loaded from the service provider and integrated into the design environment.

Currently, remote access to such services is almost entirely limited to communication via the World Wide Web. WWW technology is adequate for providing designers with basic descriptions of manufacturing services and can support transactions involving requests for quotations, purchase orders etc.

Today, a number of services exist on the Internet for PC

1. This work is supported by NSF grant MIP-9420396 and by the DARPA MADE program under N0001492J1833

2. Java is a trademark of Sun Microsystems, Inc. Documentation can be found at http://java.sun.com.

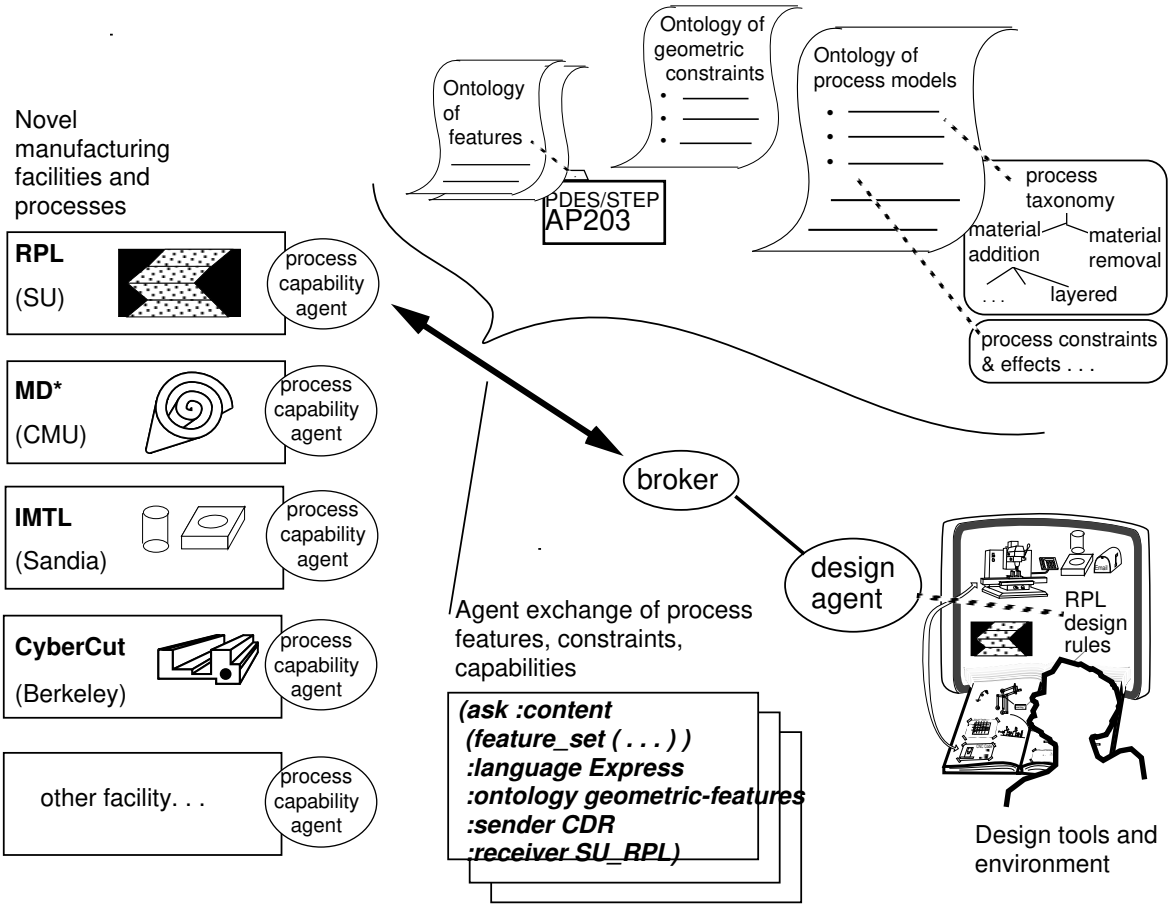

FIGURE 1. Part of an agent dialog for transmitting process capabilities into the design environment. In this case, a KQML message is sent from the design agent to obtain a list of standard features that the process can accommodate. The message content is in Express and refers to ontologies (formal agreements on terms and definitions [Gruber 1993]) of features and constraints which, in turn, refer to the PDES/STEP standard [STEP 1992; EXPRESS 1992].

board fabrication, cable harness assembly, stereolithography, etc.[3]

For the most part, these services use an HTML forms-based interface to acquire information from designers (CAD files, etc.) and customize displayed data to match designer preferences.

However, if we want to make designers confident about using novel manufacturing processes, a more intimate dialogue between design and manufacturing is required. Ideally, we would like to make all aspects of the manufacturing process accessible. For example, designers should be able to run process simulations, at either a superficial or detailed level, on partial or whole designs. They should be able to obtain manufacturability rules and guidelines that they can load directly into their native CAD environments. They should be able to load processing constraints (e.g., on materials or feature dimensions) into their CAD systems and have them checked and enforced before submitting designs to manufacturing services. In this way, designers can quickly become as familiar with manufacturing processes as they desire. In some cases, a high-level description of the process capabilities and a cost and time estimate may suffice. In other cases, for example when it is essential to minimize manufacturing costs or to meet stringent demands on tolerances or materials properties, the designer will want detailed access to process characteristics, constraints and costs – as they apply to his or her design.

3. Several examples can be found on the ARPA ACORN web page http://acorn.eit.com/acorn-services.html

The ability to submit designs with confidence that products would come back meeting specifications was a major factor in the success of the MOSIS project for VLSI design and prototyping. An important technical ingredient of MOSIS was the adoption of conservative design rules that made it possible to specify features with confidence that they would be manufacturable [Mead 1994]. The challenge in the present case is that we are dealing not with a particular semiconductor fabrication process but with a variety of mechanical processes and facilities, any one of which may be of interest. Moreover, the designers may be using a variety of CAD tools. It therefore becomes necessary to establish a standard mechanism by which designers can obtain process capability models from disparate processes on-demand, and load them into their preferred CAD environments.

Specifically, the mechanism should enable designers to locate candidate services, evaluate the processes they perform and acquire capability models which can be used to compute the manufacturability of evolving designs with an accuracy necessary to meet the design requirements. (In this context, "manufacturability" includes both the ease of fabrication and the ease of process planning.) The communication mechanism should also allow unsolicited information, such as updates on process capabilities, to be transmitted from manufacturing facilities to designers.

To enable such a dialogue between designers and manufacturing facilities we must first address the following issues:

- How is process capability represented?
- How are capability models located and acquired by the designer?
- How are capability models mapped into the design space?
- How is the information contained in these models applied during design?

These issues encompass an extensive amount of work; our particular contributions are as follows:

- an architecture for information exchange between designers and manufacturing services which includes:
- an object-oriented agent template, with specializations for design, service and service broker agents, that expedites the task of wrapping existing CAD environments and process planning software.
- a mechanism for knowledge representation which enables the exchange of declarative, procedural and human readable information.
- a negotiation protocol which outlines the sequence of communications between designers and manufacturing services.

Our approach builds upon recent developments in the application of agent-based software to engineering, including communication protocols and representation languages that allow engineering agents to exchange information via the Internet [McGuire et al. 1993; Olsen et al. 1995[4]; Pancerella et al. 1995].

The remainder of this paper will discuss our assumptions about manufacturing and design processes, our model of process capability and the agent architecture used for information exchange. We also place our approach in the context of other work on agents and technologies for sharing objects among diverse applications. This discussion will be followed by an example scenario of how this architecture can be utilized by designers to acquire and apply capability models. The discussion is focused on the processes of layered shape deposition and CNC machining services, under development at Stanford[5] and U.C. Berkeley[6], as examples of two very different manufacturing processes that can be accommodated with our approach. Appendix A provides details on our capability model representation and Appendix B discusses the agent structure and class libraries that we have developed along with pointers to downloadable code for the Java Agent Template.

2. REPRESENTING MANUFACTURING CAPABILITIES

2.1 Design and Manufacturing Features

In the following discussion, we assume that designers are doing feature-based design using three-dimensional CAD systems to represent the solid geometry of their designs. The features may include standard engineering features, such as "holes", and custom features developed by the designers for their own convenience. The difficulties of mapping from design features to manufacturing features and of extracting features from general three dimensional geometries are well known and the subject of extensive investigation elsewhere (for example, see [Gupta and Nau 1995]). The mapping efforts will presumably be facilitated by translation services, such as the on-line service provided by STEP Tools, Inc.[7].

4. A good introduction to this technology can be found in the Concur WWW pages: http://piano.stanford.edu/concur/
5. Stanford Rapid Prototyping Laboratory. See http://www-rpl.stanford.edu/
6. Berkeley CyberCut service: http://commando.berkeley.edu/CyberCut/
7. See http://www.steptools.com/

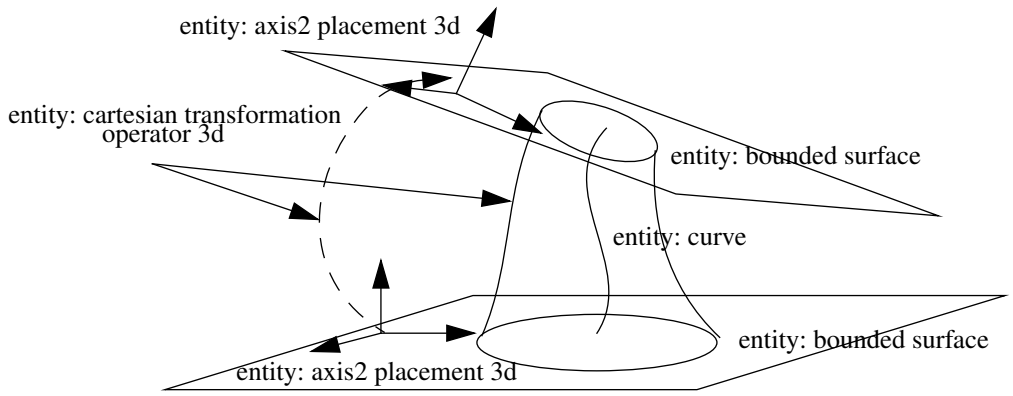

FIGURE 2. Generalized cylinders are a useful geometric representation that can be specialized to match different design and manufacturing features and facilitate the mapping between them. Corresponding STEP entities are labeled.

In the example in the following section, we will use generalized cylinders [Ponce 1990] as a geometric primitive that can be specialized to include common design and manufacturing features (see Figure 2) including those used by machining and layered shape deposition. The generalized cylinder consequently provides a useful neutral representation for mapping manufacturing features into the design space and vice versa. (However, our overall approach could easily be implemented using another geometric representation.) The parameters which define the geometry of each generalized cylinder feature can be translated into corresponding STEP, Part 42, [STEP 1991] entities represented according to AP 203 to facilitate the use of these features by multiple CAD systems (see Appendix A). We also assume that manufacturing processes can be represented in geometric terms as sequences of operations that add, modify or remove geometric features from an initial "stock" part (which may be null in the case of material deposition processes). Processes may of course have additional effects such as changing the surface finish, material hardness, stress state, etc. We observe that for machining processes such as milling or turning, the mapping between design and manufacturing features is fairly close; for example, the holes and pockets used by designers correspond to the features produced by a drill and end-mill. However, for processes such as shape-deposition, the material layers or "compacts" [Merz et al 1994] used to build 3-dimensional solids rarely have any direct relation to the design of a part.

2.2 Process Capability Model

Our architecture represents manufacturing services, and the processes they implement, using a hierarchical taxonomy of manufacturing objects, as shown in Figure 3. The basic concept is similar to the object-oriented process hierarchy originally proposed by Ray (1987). Each manufacturing object is defined in terms of attributes, and each attribute includes other manufacturing objects. At the top of this taxonomy is the manufacturing service which implements a set of manufacturing processes. Each process is defined by the materials which it can handle, and each process/material combination possesses a set of capability models. In general, each capability model includes a combination of declarative design rules and constraints, process simulation procedures and documentation. The design rules represent constraints that apply to the designs which use the materials and processes associated with the process capability model. Example rules include constraints on total part volume, geometric symmetries and expressions constraining the parameters of specific feature types. Simulation functions take as input the geometric model of a design or partial design and may return processing parameters (e.g, time, cost) or provide a graphical display of the process and its effects. In the remainder of this section we discuss design rules, constraints and simulation procedures in more detail.

Design rules and constraints

Declarative representations of process-derived geometric (and other) constraints provide a way of assuring manufacturability while maintaining a separation between the representation of the design and the description of the processes used to manufacture it. The neutral descriptions of these constraints also allow them to be used by constraint propagation systems or other reasoning systems that may be available to the designer. We note that for the purposes of ensuring design for manufacturability, it may be more useful to have something analogous to the "spare Mead-Conway rule-set" [Muhkerjee and Hilibrand 1994] originally used in MOSIS, than the sophisticated rules developed with each new VLSI generation. Indeed, it is likely that facilities will offer designers process capabilities at more than one level of sophistication and detail: a conservative set of rules that ensure easy manufacturability and increasingly sophisticated analyses and simulations that can be used for designs that cannot conform to the "easy" rules.

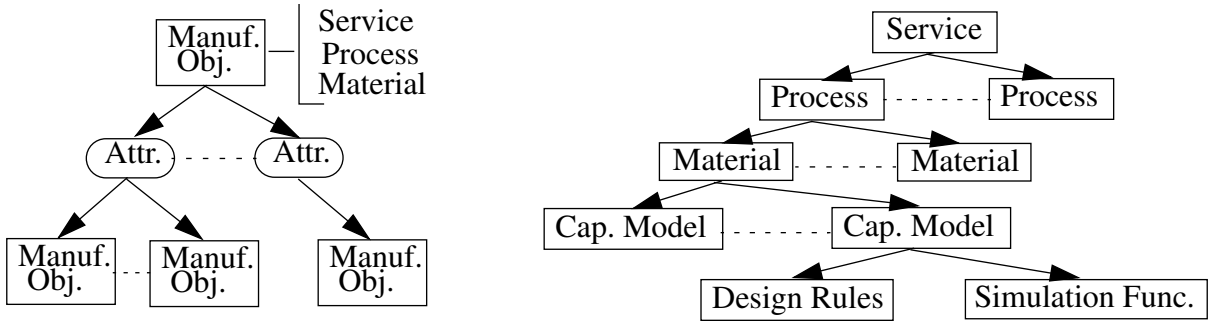

FIGURE 3. A taxonomy of manufacturing objects

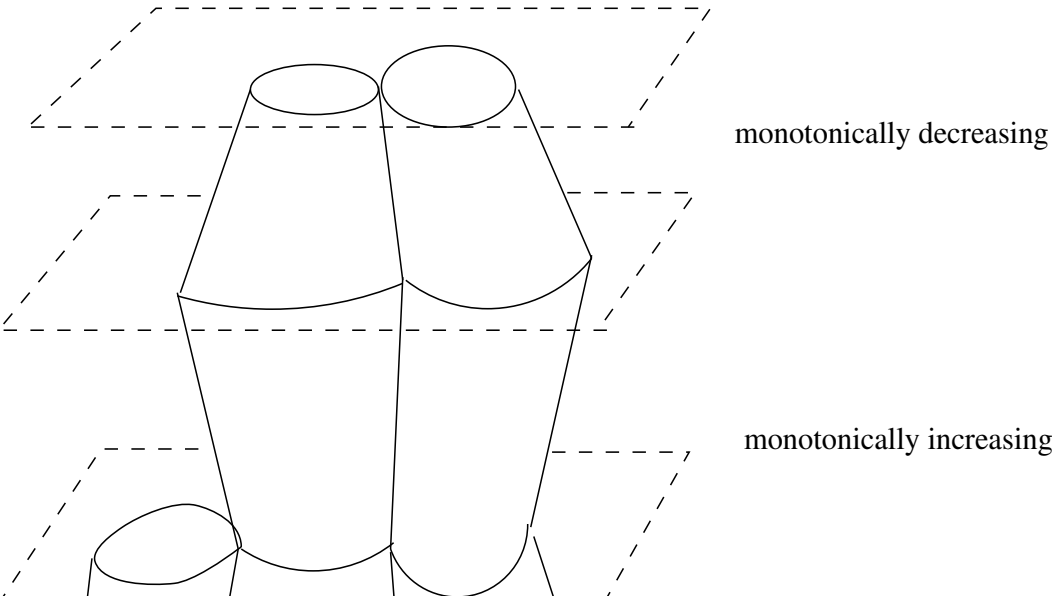

FIGURE 4. Union of multiple generalized cylinders with horizontal bands of uniform taper.

We suspect that designs will largely conform to the conservative rules with a few exceptions. Flagging these exceptions can save time when doing process planning [Merz et al. 1994]an be difficult to obtain declarative representations of manufacturing characteristics and constraints as they apply to an entire design. However, the constraints associated with manufacturing features are often easier to express. For example, design rules for machining and layered shape deposition can be expressed in terms of the specializations of generalized cylinders that apply to each process. (Appendix A outlines the geometric constraints associated with a generalized cylinder.) In the case of machining, feature constraints for "easy machining" may include maximum length/diameter ratios for holes or depth/corner-radius ratios for efficient machining with standard drills and end-mills. Other easily expressed constraints would include the maximum envelope of a part and general specifications on tolerances and surface finish, perhaps refined in terms of part size and overall geometry. The initial design constraints that we are implementing for ease of manufacturing at the U.C. Berkeley rapid prototyping facility are listed in Table 1 in Appendix A.

In the case of layered shape deposition, a primary consideration is whether the geometry of the design can efficiently be decomposed into layers. This decomposition is a function of surface profiles and desired surface finish [Merz et al 1994]. If the geometry of the part does not result in any horizontal bands or layers containing both undercuts and non-undercuts, the layering procedure is simple: layer boundaries are identified at transitions between undercut and non-undercut. However, if both undercut and non-undercut features are contained within the same horizontal band, the band must be split into a set of "compacts," significantly complicating the computational process. This complication can be avoided by building the design from primitives that taper monotonically in the direction of lay-

ering (see Figure 4) for "easy shape deposition." The feature constraints listed in Table 2 enforce just such a geometric property.

Simulation Functions

Where declarative representations of process-related constraints or characteristics are impractical, the next best solution is to provide functions that simulate the effects of processing on the design or analyze elements of the design from the standpoint of manufacturability. For example, when machining a part with thin wall sections, the most robust way to tell if chatter will be a problem is to simulate the actions of milling the part using a dynamic milling model such as the comprehensive models developed by the Machine Tool Agile Manufacturing Research Institute.[8] Making these process simulations available to the designer can help to promote an understanding of the process and encourage design for manufacturability.

A problem with procedural code is that it is typically much less portable than declarative expressions. To some extent, the portability problem can be overcome if the simulation is set up as a remote service to which designs can be submitted in a neutral format (e.g., PDES/STEP [STEP 1991]). However, this "batch" mode of interaction does not lend itself to asking lots of "what if" questions in the midst of a design session. Our experience with previous systems for concurrent design and process planning [Cutkosky and Tenenbaum 1990; 1991] suggests that a good way to encourage design for manufacturability is to make the results of fast incremental process planning and simulation available to designers whenever they ask for it. The important difference in the present case is that we have a completely distributed system with no common software platform and no prior commitment to a particular manufacturing process. To overcome this problem, we are using the Java language to provide platform-independent process simulations to the designers. The simulations will be aimed at answering "what if" questions and will emphasize fast approximate results as applied to one or a few features. Representing these procedures as a set of Java classes enables them to be dynamically loaded and run from within the designers' native CAD environment. (A portion of our agent architecture addresses the problem of dynamically loading classes into an executing runtime environment, see Figure 6.) For comprehensive simulations and analyses of a completed design, the remote service concept mentioned earlier is probably still the best option.

8. On-line dynamic milling models from MT-AMRI are provided at http://mtamri.me.uiuc.edu/mtamri.html.

3. INFORMATION ARCHITECTURE

In this section we discuss the agent architecture, templates and class libraries we have developed to exchange the design rules and procedures discussed in the last section.

We utilize the agent abstraction as a mechanism for encapsulating and exchanging distributed knowledge and functionality. Similar to objects in object-oriented programming, agents provide a mechanism for developing scalable, modular systems with well-defined interfaces. In our approach, "engineering agents" are primarily "interface" agents (we are not concerned with internal "behavior" or "beliefs") with three defining characteristics:

- They are autonomous – Each agent operates as an independent process and interacts asynchronously with other agents on a peer-to-peer level (as opposed to a client-server architecture).
- They encapsulate functionality – Agents serve as containers for human engineers, computational tools and data. The internal contents can only be accessed via a formalized interface.
- They use an agent communication language – Agents exchange information using messages with syntax and semantics defined by a standard language. The contents of these messages are machine interpretable and can include declarative or procedural knowledge.

We have implemented this agent conceptualization by developing an agent template in the Java programming language. This template provides a generic software agent, with multi-threaded execution, which communicates asynchronously with other agents distributed over the Internet using KQML [Finin et al. 1994][9] as an agent communication language. The agents utilize the platform-independent nature of Java to exchange both declarative and procedural information. The architecture of the Java Agent Template is complex and we have room here only for a general description; details can be found at http://cdr.stanford.edu/ABE/

The functionality of the agent template derives from its architecture at both a community and individual level. At the community level, we support three general classes of agents: service agents, broker agents and client agents, as shown in Figure 5. Each service agent represents a persistent engineering service on the Internet and is implemented as a stand-alone application integrated with legacy software and local databases.

9. Knowledge Query and Manipulation Language. See http://www.cs.umbc.edu/kqml/

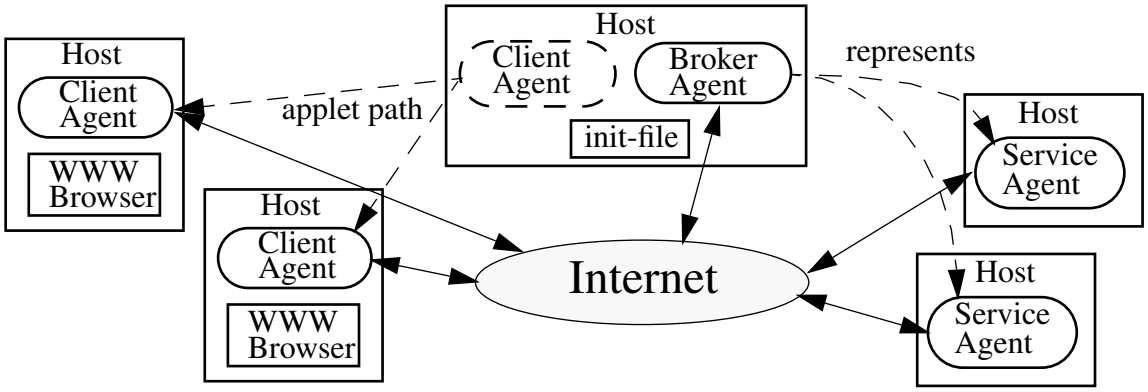

FIGURE 5. Community architecture of manufacturing services, clients and broker agents. Interaction can be either peer-peer or client-server.

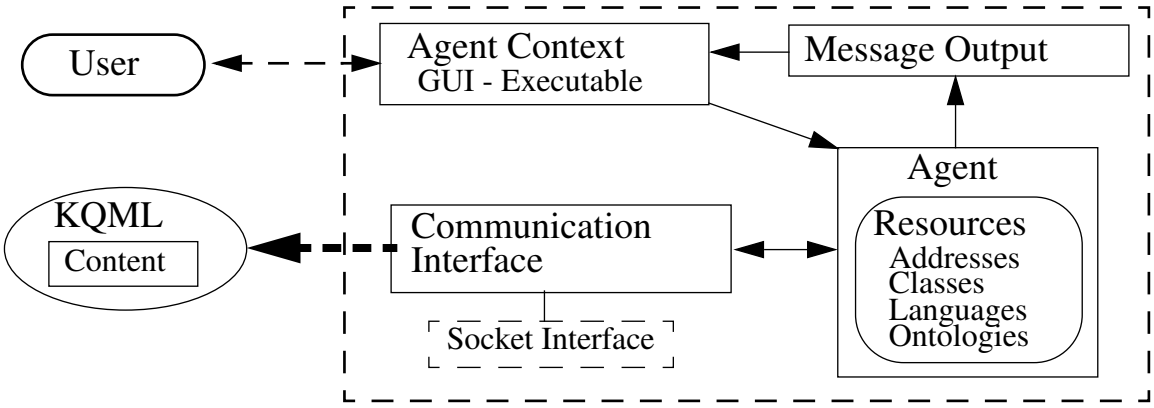

FIGURE 6. The architecture of an individual agent as implemented with the Java Agent Template.

Broker agents are also persistent stand-alone processes, which represent a consortium of service agents and function as a central directory for agent names, addresses and descriptions. The simplest type of broker is the agent name server (ANS) which provides a central directory of agent names and addresses. Client agents are potentially transient processes which can be implemented as Java Applets and dynamically loaded onto user's machine via the WWW These agents use brokers to discover specific service agents with whom they will directly communicate to access service functionality. Because of the diversity of manufacturing processes and design environments, agents must be able to execute in a range of computational environments which may possess specialized graphical interfaces and communication infrastructures. These requirements motivated us to develop an architecture which divorces an agent's contents from both its external and internal interfaces. The resulting agent template has four distinct elements: the *agent-context, communication interface, message output* and the *agent* itself, as shown in Figure 6. (See Appendix B for details on the hierarchy of Java classes used to implement this architecture.)

The *agent context* allows the both the agent and communication code to be separated from the execution environment and user interface. This structure enables the same agent to be transparently implemented as a stand-alone application, an applet or a daemon process, with functionality which does not directly depend on the presence of a graphical user interface. The interface between an agent and its context has been formalized as a *message output* object.

- The *agent* represents the primary functional element in the architecture. An agent can be modeled as a black box which asynchronously accepts and outputs messages via the *communication interface*. Each message has a top level syntax and semantics defined by KQML. This outer level wraps the inner content which has its own syntax and semantics. An agent's communication interface provides the low level mechanism for reliable transmission of messages. In the present implementation, communication is done using a socket-based interface which provides for multi-threaded message receipt and transmission.

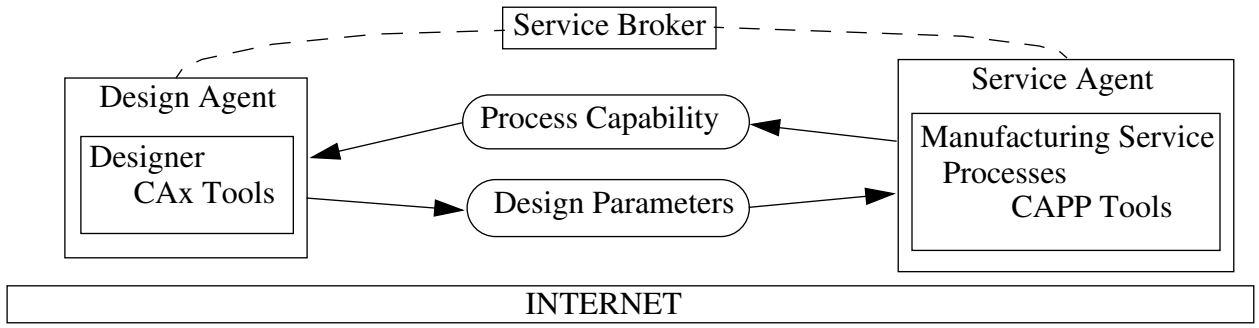

FIGURE 7. High-level view of the architecture for manufacturing agents.

- The knowledge and functionality of an agent is represented using a set of *resource* objects (e.g. *addresses*, *classes*, *languages* and *ontologies*). Each resource contains a set of objects which are instances of the same Java class (or share a common super class). The primary function of agent messages is to inquire about and exchange these resource objects. These resources can be exchanged via the declarative representation of object parameters (in the KQML message) or by remote loading of appropriate Java class files (the location of which is exchanged in KQML). Internal access to resource objects is synchronized to prevent race conditions.

3.1 Agents, distributed objects, CORBA and Java

Several people have questioned the relationship between the Java Agent Template and the work being done on CORBA (and other distributed object projects). First, one must distinguish the design paradigm (Agents and distributed objects) from the implementation (Java and CORBA) and analyze each aspect separately.

- Design Paradigm: Many people are confused by the distinction between agents and distributed objects. The first point is that agents and distributed objects are not mutually exclusive: an agent-based system can be built on a distributed object substrate. The primary difference between agents and distributed objects lies in the message structure used for communication. Whereas distributed objects communicate via method invocations agents communicate using a formal messaging protocol, such as KQML, which is divorced from the agent's implementation. Agents also generally maintain state information. With sufficient discipline, a programmer could implement formal agent messaging using method invocations but such formality is not required. Agents provide a level of abstraction above distributed objects.
- Implementation: Many people are also confused by the differences between a language such as Java and a CORBA-compliant implementation. Again, Java and CORBA are not mutually exclusive: numerous IDL compilers for Java have already been developed. Our choice of the Java language was based mainly on practically considerations. Java provides a robust mechanism, with minimum overhead, for developing an object-oriented, platform-independent program which can dynamically import new code. However, there are no particular barriers to integrating the Java Agent Template with CORBA objects.Example Scenario

This section provides an example of how the above methodology and architecture can be applied to enable a designer to locate, evaluate and apply process capability information to an actual design. An implementation of this architecture using Pro/ENGINEER for the CAD platform and shape deposition and CNC machining as manufacturing processes is under development. As the prototype evolves, details will posted at http://cdr.stanford.edu/RVPP/.

The architecture discussed in the previous section will be specialized to create three specific agents: design agents, service agents and service brokers. The human designer and his/her CAx tools are encapsulated within the design agent. A manufacturing service, the set of processes which that service implements and associated computational tools and data are represented by a service agent. Communication between design and service agents is facilitated by a service broker (see Figure 7) which represents a group of service agents and can provide high level filtering for designers.

Information concerning process capabilities and design characteristics will be exchanged between these agents as shown conceptually in Figure 8. This flow represents the following sequence of actions taken by the designer:

3.2 Locate potential service/process combinations.

The designer begins by executing a design agent on his/her local machine (perhaps via the WWW by loading a design agent as an applet from the service broker site, or by executing the agent as a stand alone process).

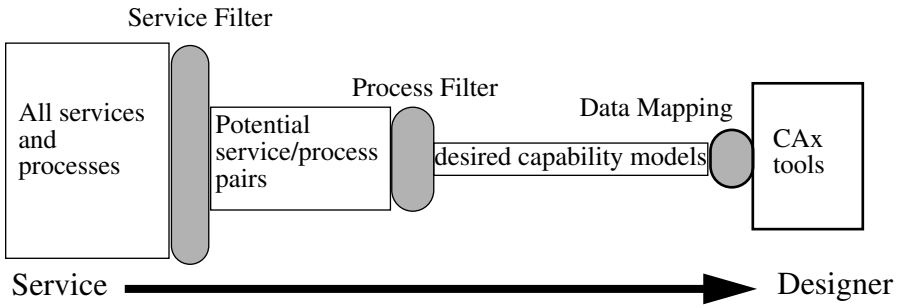

FIGURE 8. Filtering information about services and processes and mapping it to the design environment.

The agent is initialized with the location of one or more service broker agents representing groups of manufacturing services (only a single service broker is used in the current implementation). The designer will submit to the selected service broker, via the design agent, information about the design requirements which can be used for high-level filtering of potential service/process combinations. Although this filtering has not been addressed in the current implementation, we believe this step is important in the overall process and should be based on the following general categories of design constraints:

- purchasing/scheduling constraints (e.g. maximum $/hour for machining time, minimum lot size, max lead time);
- physical constraints: material properties and geometry (e.g. required stock size / general volume, symmetries, solid features, surface types, tolerances).

This filter will typically evolve over the life of the design and provides a mechanism for informing the designer about relevant new or altered manufacturing processes.

3.3 Evaluate a specific manufacturing service.

Once a list of potential manufacturing services has been obtained, the designer will undertake a detailed evaluation of the characteristics and capabilities of each. As described in Section 3, each service is modeled by a hierarchy of manufacturing objects. Communication between the design agent and service agent will enable the designer to search this hierarchy, investigating the capability models of different processes/material combinations. As the designer browses through the human-readable documentation provided by the service, the design agent builds an object model of the manufacturing service and its processes.

3.4 Evaluate a process.

The next step is to acquire process capability models. Declarative components of the capability model will be exchanged using KQML messages, procedural elements will be loaded as Java classes.

In the present implementation, the capability model provides information for evaluating the manufacturability of a design's nominal geometry. Design rules consist of constraints on the geometry of generalized cylinder features and on the CSG operations used to combine those features into more complex solids. Constraints, such as the ones summarized in Appendix A, are represented declaratively using the Express language. Each capability model will also include process simulation procedures which take designs meeting the constraints and return, if the simulation is successful, trees of processing features used to realize the design.

3.5 Use the process capability model to constrain the design process.

Acquired capability models will be mapped into a format suitable for integration with the designer's preferred CAD system (which may employ a neutral geometric representation such as STEP AP 203). Feature constraints will be applied against all feature instantiations and CSG operations attempted by the designer. Process simulations will be configured to accept input from the CAD system and return output which can be mapped into a form which the CAD system can use. These simulation procedures may be entirely represented by local Java code or can be implemented as a Java interface (locally executed) which communicates to analysis code running on a remote machine.

3.6 Submit the design for fabrication.

Completed designs will be submitted to the service agent which provided the process capability model. The manufacturing service represented by this agent can then conduct process planning and quote generation in light of the applied constraints.

4. IMPLEMENTATION STATUS AND FUTURE WORK

An approach enabling software agents to locate each other, exchange messages and automatically load resources consisting of both declarative (e.g., constraints) and procedural (e.g., process simulations) information has been developed. This approach has been implemented with the Java Agent Template, which has been tested to demonstrate agent instantiation, registration and communication of objects and messages. Several groups at Stanford, Sandia National Laboratories, and Mitre Corporation are experimenting with the Version 2.0 source code and documentation, which are available via the WWW[10].

Our next steps are to experiment with agent dialogues among design and manufacturing services to evaluate the effectiveness and limitations of this approach. An object class hierarchy for process descriptions and capabilities has been established and the generalized cylinder has been adopted as initial generic feature type as discussed in Section 2. Initial experiments will be conducted in collaboration with prototyping services at RPL,[4] U.C. Berkeley,[5] and Sandia National Laboratories.[11]

5. ACKNOWLEDGMENTS

We would like to acknowledge the input and assistance of the members of the Rapid Prototyping Laboratory at Stanford University, Carmen Pancerella of Sandia National Laboratories and Greg Olsen of Enterprise Integration Technologies. Support for this work has been provided by the National Science Foundation under grant MIP-9420396 and by the DARPA MADE program. Additional support has been provided by Sandia National Laboratories.

6. REFERENCES

Beaman, J.J., 1993, "Desktop Manufacturing", *Discovery*, University of Texas at Austin, Vol. 13, No. 1, pp. 18-22.

Cutkosky, M. R. and Tenenbaum, J. M., 1990, "A Methodology and Computational Framework for Concurrent Product and Process Design", *Mechanism and Machine Theory*, Vol. 25, No. 3, pp. 365-381.

Cutkosky, M. R. and Tenenbaum, J. M., 1991, "Providing Computational Support for Concurrent Engineering", the *International Journal of Systems Automation: Research and Applications*, Vol. 1, No. 3, pp. 239-261.

Finin, T., Weber, J., et al., 1994, "Specification of the KQML Agent-Communication Language", The DARPA Knowledge Sharing Initiative External Interfaces Working Group, Feb. 9, 1994.

Gruber, T., 1993 "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," in Formal Ontology in *Conceptual Analysis of Knowledge Representation*, Guarino and Poli, eds., Kluwer.

Gupta, S., Nau, D., 1995, "Systematic approach to analyzing the manufacturability of machined parts", *Computer-Aided Design*, Vol. 27, No. 5, pp. 323-342.

ISO 10303 Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 11: Description Methods: The EXPRESS Language Reference Manual, ISO DIS 10303-11: 1992(E).

McGuire, J.G., Kuokka, D.R., Weber, J.C., Tenenbaum, J.M., Gruber, T.R. and Olsen, G.R., 1993, "SHADE: Technology for Knowledge-Based Collaborative Engineering", *Concurrent Engineering: Research & Applications*, Volume 1, Number 3.

Mead, C., 1994, "Preface to Workshop report on New Paradigms for Manufacturing," National Science Foundation report NSF 94-123, A. Mukherjee and J. Hilibrand, eds., Arlington, VA, pp., 1-2.

Merz, R., 1994, "Shape Deposition Manufacturing", Dissertation, Department of Electrical Engineering, Technical University of Vienna.

Merz, R., Ramaswarni, K., Prinz, F.B., Terk, M., Weiss, L.E., 1994, "Shape Deposition Manufacturing", *Proc. Solid Freeform Fabrication Symposium*, University of Texas at Austin, pp 1-8.

Mukherjee, A. and Hilibrand, J., 1994,Group 1 Report, Workshop on New Paradigms for Manufacturing," National Science Foundation report NSF 94-123,, eds., Arlington, VA pp. 9-16.

G. R. Olsen, M.R. Cutkosky and J. M. Tenenbaum and T. R. Gruber, 1995, "Collaborative Engineering Based on Knowledge Sharing Agreements", *Concurrent Engineering Research and Applications*, Vol. 3, No. 2, pp. 145-159.

Pancerella, C., Hazelton, A., Frost, H.R., 1995, "An autonomous agent for on-machine acceptance of machined components", Proceedings of *Modeling, Simulation, and Control Technologies for Manufacturing*, SPIE's International Symposium on Intelligent Systems and Advanced Manufacturing.

Ponce, J., 1990, "Straight homogeneous generalized cylinders. Differential geometry and uniqueness results", International *Journal of Computer Vision*, Vol. 4, No. 1, pp 79-100.

Ray, S. R., 1987, "Process Reasoning", Computers in Industry, 9, pp. 329-335.

"STEP Part 1: Overview and Fundamental Principles", Version 9, ISO TC184/SC4/WG PMAG Document N50, December 1991.

10.http://cdr.stanford.edu/ABE/

11.http://nittany.ca.sandia.gov:8001/index.html

Appendix A: Generalized cylinders: representation and constraints

The geometry of a generalized cylinder feature is defined in terms of an axis (3-space curve parametrized by arc length), a cross-sectional surface (bounded, planar surface) and a sweeping transformation (transforms the cross-sectional surface as a function of the axis arc length, maintaining perpendicularity between the axis tangent and the surface) Figure 2 shows a generalized cylinder as defined above with STEP, Part 42 entities corresponding to each of the geometric elements.

The nominal geometry of a generalized cylinder feature will therefore be defined by the following sets of constraints (there can also be constraints involving combinations of these parameters):

- Axis Constraints: -constraints on the parametrized axis functions x(s),y(s) and z(s) and their derivatives, direct constraints on the curvature and torsion of the axis.
- Cross-Sectional Surface Constraints: -constraints on the closed curve which defines the edge of the surface
- Sweeping Transformation Constraints: -constraints on the properties of the transformation (dependency on axis arc length)

The CSG operations used to combine feature primitives into more complex solids can be defined in terms of the operator type (union or difference) and, for each operator type, the relative orientation between the local coordinate frames for each of the feature operands. This leads to the following constraint types:

- Operator Constraints: -Subset of {union, difference}.
- Operand Constraints: -Constraints on the coordinate transformation matrix

Feature Parameter	Constraint
Axis	linear (curvature = 0, torsion = 0)
Cross-Sectional Surface	concave edges must meet minimum radius of curvature
Sweeping Transformation	constant (identity matrix)
Operator	difference only
Operand	coordinate frames can only have relative rotational displacements which are integer multiples of Π/2

TABLE 1. Constraints associated with 3-axis milling

Feature Parameter	Constraint
Axis	linear (curvature = 0, torsion = 0)
Cross-Sectional Surface	if edge is concave, must meet minimum radius of curvature
Sweeping Transformation	monotonically increasing or decreasing as a function of axis arc length
Operator	union only
Operand	within one horizontal band can only combine feature whose transformations change in similar directions; rotation only about coordinate axis parallel to extrusion axis; displacement unconstrained.

TABLE 2. Constraints associated with layered shape deposition

Appendix B - details on the Java Agent Template

See http://cdr.stanford.edu/ABE for detailed information on the Java(tm) Agent Template which includes a programmers guide, class API and the distribution itself (Java source and class code), available as either a compressed tar file or a zip archive. Figure 9 shows the general package hierarchy.

Agent Life Cycle – The life of an agent constructed according to the Java Agent Template includes the following milestones:

- Execution of the AgentContext. This creates the Agent itself, associated CommunicationInterface, (SocketInterface), MessageOutput and GUI components.
- Initialization file is retrieved and processed (contains a set of KQML messages, one of which has the address of an ANS (Agent Name Server)

- Connection to the Internet and registration with the ANS
- Termination: address invalidation message is sent to the ANS

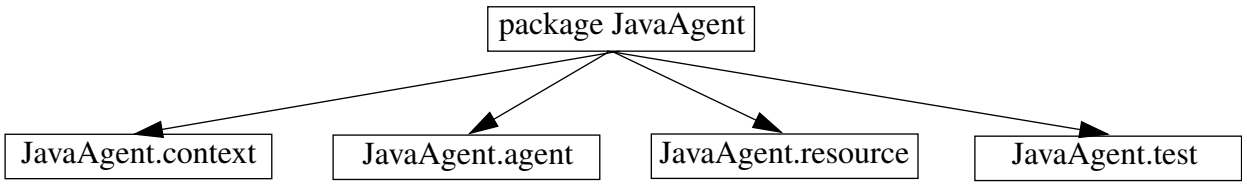

FIGURE 9. Top-level packages for the Java Agent Template.

- *Addressing* – Every agent possesses a unique name and address. Addresses consist of a host name and a port. An ANS maintains an archive of all agent names and addresses (and enforces unique naming).

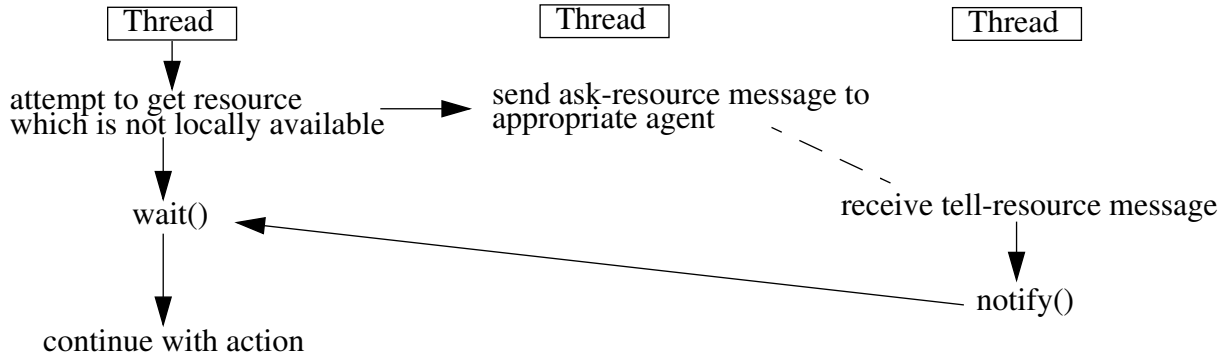

FIGURE 10. *Resource Retrieval* – The multi-threaded implementation of the Java Agent Template, combined with the need to acquire information asynchronously via inter-agent messages, necessitated the development of a specialized methodology for resource retrieval. Figure 10 illustrates this procedure. In the case of exchanged Java classes, the receipt of a tell-resource message (which provides the URL for the code) will prompt the remote loading of the class code via the NetworkClassLoader built into the agent template.

Message Interpretation – When a message is received by an agent's communication interface, the following sequence of actions take place:

- The message is parsed, at a top level, according the KQML syntax.
- The "ontology" field for the message is checked against the agent's local Ontologies resource. If the specified ontology subclass is not present, the agent "retrieves" it before continuing with the message processing.
- If the specified Ontology is successfully obtained, the interpretMessage() method of this ontology object is called to process the present message. This method first checks the "Language" field, if the language is not locally known, it is retrieved before further processing. If the language is supported and can be obtained, the content field of the message is parsed according to that language.
- The action to be taken on the message content is encoded within the ontology class.