ELSEVIER

0010-4485(95)00053-4

Computer-Aided Design, Vol. 28, No. 5, pp. 333-345, 1996
Elsevier Science Ltd
Printed in Great Britain

Discourse Model for collaborative design

Michael P Case and Stephen C-Y Lu*

A Discourse Model, including a structure and a process, is developed that provides software support for collaborative engineering design. The model shares characteristics of other design systems in the literature, including frames, constraints, semantic networks, and libraries of sharable design objects. It contributes a new model for conflict-aware agents, dynamic identification and dissemination of agent interest sets, a virtual workspace language, automatic detection of conflict, and a unique protocol for negotiation that ensures that interested agents have an opportunity to participate. The model is implementation independent and applicable to many research and commercial design environments currently available. An example scenario is provided in the architecture/engineering/construction domain that illustrates collaboration during the conceptual design of a fire station. Published by Elsevier Science Ltd

Keywords: agent, conflict, discourse design collaboration, concurrent engineering, blackboard architecture, KQML

INTRODUCTION

Given the number of different disciplines, trades, and roles involved in the design of a building, it is not surprising that things frequently go awry. Unexpected interdependencies are found when solutions generated by different designers are recombined. For instance, the size, composition, and placement of glazing (windows), has a significant impact on both the energy requirements for cooling and the illumination parameters of intensity and glare. Design of the lighting systems, in turn, directly influences the cooling load. Each of these design activities must be carried out by individuals who are highly trained in their particular discipline. They must work together, however, to achieve a better overall design and avoid local optimization at the expense of the whole. Construction, like many types of manufacturing, is not completely hierarchically decomposable. This means that although the design problem may be divided into smaller pieces, there are still many interdependencies among components of the subproblems.

This paper describes a model for collaborative engineering, called the Discourse Model, that treats interactions between designers as a process of discourse. When using this model, design commitments made by designers are treated as opinions, subject to review and revision by other designers. This approach requires a paradigm shift from present day automation methodologies, which treat assertions as facts and do not explicitly represent differences or agreements of opinion.

The Discourse Model is intended for use in software environments that provide automation support for collaborative engineering design. It includes both structural and process specifications. Components of the structure include specifications for a blackboard-based *workspace* that incorporates frames, constraints, semantic networks, libraries of sharable design objects, software agent modules, an electronic messaging system, and a virtual workspace language based in part on the Knowledge Query Manipulation Language (KQML)[1]. Components of the process include procedures for identifying agent interest sets, applying state transformations to the design model, switching design contexts, identifying conflicts between designers, and tracking resolved conflicts. The model is implementation independent and applicable to many research and commercial design environments currently available.

The Discourse Model has several implications for managing the complexity of large design projects that require the collaborative efforts of more than a few individuals. First, it fills a research gap between client/server based systems that use a single shared database and peer-to-peer systems that incur high data translation and semantic losses. Second, it yields a new capability for automatic identification and dissemination of information about the types of design objects that designers want information about, leading to detection of unsuspected conflict areas between designers. Finally, the conflict detection, rationalization, and resolution protocol ensures that all interested designers have an opportunity to participate in the resolution of conflicts.

US Army Construction Engineering Research Laboratories, PO Box 9005, Champaign, IL 61821, USA
* Knowledge-Based Engineering Systems Research Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
Paper received: 20 June 1994

As space limitations limit the detail that can be presented herein, additional information is available from Reference 2.

BACKGROUND

Collaboration involves communication and negotiation between engineering teams at distributed sites as subtasks are performed at each site. Collaboration technology supports the needed communication. However, different couplings between subtasks place different demands on the level of communication which must be provided by the collaboration technology of a concurrent engineering (CE) environment. In the following, we classify three different forms of collaboration corresponding to different couplings among subtasks. In each case, we describe the appropriate collaboration technology as well. We define 'communication impedance' to refer to the difficulty in communicating when teams are not collocated. This includes network delay, protocol differences between environments, and formation translations which must occur. A collaboration technology overcomes the communication impedance inherent in a distributed work environment to a greater or lesser degree, depending on what is demanded by the coupling among subtasks. Since the coupling among subtasks varies during the different phases and processes of concurrent engineering, a full spectrum of these three approaches must be available to support collaboration among engineering teams.

Loosely-coupled collaboration

In this approach, subtasks are very loosely coupled; a collaboration technology with a high communication impedance is acceptable. Communication usually consists of the transfer of input and output data streams at the beginning and end of a problem solving session. One iteration of the communication cycle may be fairly long in duration, and the communication bandwidth (i.e. the types of information communicated) is low.

In its most primitive form, loosely-coupled collaboration requires that human designers extract information from the output of one software program and transcribe it into the input of another program. At more advanced levels, formal and *de facto* standards such as IGES, DXF, and STEP permit the output of software programs to be interpreted by other software automatically. This approach suffers from the potential drawback that there may be an impedance mismatch between the internal representation of a particular software implementation and the standard used for communication. Additional communciation impedance is imposed by a need for the humans using the software to produce and exchange translated data files. Issues of conflict management are typically not addressed with this method.

The loosely-coupled approach has historically been the initial step toward concurrent engineering. It can be attained by a technologically straight forward transformation of an existing distributed engineering environment. It is profitably utilized for neatly decomposible tasks which do not require large amounts of communciation during problem solving.

Moderately-coupled collaboration

In this approach, subtasks are moderately-coupled and communication impedance must be overcome to some degree (high impedance is unacceptable). In particular, information may be exchanged between sites during the problem solving process, instead of only at the beginning or end. The iteration loop of communication is therefore tighter.

Computer tools are used to automatically manage the translation and transmission of information between sites. Interaction is mostly hidden from the user within an interface program (or 'wrapper'), which automates the required data format translations. Network location is therefore less visible to users, though interface tools must be aware of it. Translation is hidden from users in the computer tools, however many different translations may be necessary to establish communication between all sites. Redundancy and data consistency problems exist as well, and users (and programs) are still to some degree aware of the network delay separating them.

Moderately-coupled systems may or may not use a shared database as a communications medium. Hardwick *et al.* used the ROSE object-oriented database system for a 'change management' approach to conflict identification[3]. This approach managed change using 'delta' files in a way similar to commercially available software version control systems but did not provide translation capabilities. Howard and Rehat's KADBASE used a 'neutral' representation language to link both knowledge-base and database programs[4]. The advantage of this approach was that each application program only need write a translator into KADBASE's neutral language. Conflict was not addressed. Genesereth *et al.* propose a 'Federation' architecture, in which applications communicate through a network of 'facilitators'[5]. The facilitators are accessed through a software wrapper and provide services of translation, brokering, and resource allocation, but (normally) no internal data store. Conflict is handled by allowing humans to accept or reject changes propagated from other applications. Recent work has been performed in the AEC domain using the Federation Architecture by Khedro. In this work, a structural engineer, architect, foundation designer, and cost estimator were linked through the facilitator.

The moderately-coupled approach represents the current state of the art in concurrent engineering technology. This approach is vital for integration of pre-existing engineering tools. Computer-aided engineering tools (such as solid modellers, finite element programs, CAD drafting tools, and data management tools) are usually developed independently, without being framework aware, and have large established installed bases which would make any major change to these programs difficult. This approach enables existing tools to be smoothly integrated despite network displacement and representation mismatches.

Closely-coupled collaboration

In this approach, problem solving is usually layered, subtasks are very tightly coupled to one another, and communication impedance must be almost entirely

eliminated. On one layer, programs, automated agents, and human users (collectively 'agents') cause changes as they modify the product description. At a lower layer, complex knowledge-based linkages, such as constraint networks, rules, and truth maintenance, express detailed subtask interdependence. Agent level changes propagate through the lower layer in rapid automated communication over multiple physical sites. Agents are thereby notified of remote changes which are relevant to their own subtask; responsive actions may then propagate back through the lower layer. At the lower layer, the iteration loop of communiction is very tight, since many low level iterations may need to occur to enable a single high level action. The bandwidth of communication is high, since communication occurs in many forms through different knowledge based media.

In the closely-coupled approach, communication functionality is transparently and efficiently managed below the CE environment by a distributed database or a network operating system. No support for interaction between sites is needed in the CE environment; no human or automated participant must consider its network location or the network delay between sites to do the work. All users and programs retrieve and access the same set of objects, with no communication impedance. A single shared representation of data and knowledge is used, so that no translation of different representations is required. Therefore, no duplicated data management is needed or provided in the CE environment.

This communciation functionality is vital for the support of tightly-coupled collaboration. First, the unified representation eliminates the need for format translation. Translation is a much bigger problem in the complex representations used in knowledge structures. Second, the absence of observable network delay enables real-time knowledge maintenance. As relevant events occur in the environments, such as an update which would violate a constraint, or an event which would trigger a rule, appropriate responsive actions are taken which may 'chain' throughout the environment, causing many other actions. These actions may traverse many sites, and must be executed expeditiously in order to accomplish their intended purpose. Tightly-coupled collaboration enables the rapid and transparent propagation of many actions across many sites required to maintain a rich knowledge representation. When constraints are violated, they are immediately brought to the attention of a human user, who can deal with them on an *ad hoc* basis.

Blackboard-based systems typically fall into the closely-coupled category, although many are intended for multiple software agents and a single user. Examples include Designer Software[6], IDEEA[7], ICAD[8], GALILEO, and DICE[9]. More recently, systems such as SWIFT[10] and SHARED[11] have extended the blackboard approach to multiple users and incorporated object oriented database technology. These systems usually deal with conflict using a variant of a constraint satisfaction and/or propagation system.

THE DISCOURSE MODEL

The Discourse Model treats interactions between designers as a process of discourse. Design commitments made by designers are treated as opinions, subject to review and revision by other designers. This approach requires a paradigm shift from present day automation methodologies, which treat assertions as facts and do not explicitly represent differences or agreements of opinion.

The Discourse Model provides a specification for closely-coupled interaction between humans called *users*, and software entities called *agents* in a blackboard architecture called a *workspace*. The interaction between workspaces is moderately-coupled within the bounds of a shared set of concepts specified by the Discourse Model. By agreeing to share these concepts, which we will explore later, users gain the high degree of interaction afforded by a closely-coupled approach while preserving moderately-coupled characteristics such as privacy and local flexibility in representation. Most importantly, an explicit protocol for conflict detection, rationalization, and negotiation is shared by all of the collaborators using the model.

Functional requirements for the Discourse Model are listed below, followed by a short description of each. Key words that are highlighted in **boldface** indicate individual functional requirements.

Rich Modelling Environment: The first requirement in performing collaborative design activities is that each user is able to capture concepts and ideas. A design is a representation of something that will be realized by humans or machines. A rich and flexible modelling environment is needed to allow users to delineate **requirements,** develop alternative **solutions**, and **evaluate** these solutions against each other. Users need to be able to dynamically **define** design objects, establish **relationships** between them, and place **constraints** upon them. CAD **visualization** capabilities are needed as well as **symbolic** and **quantitative** representations.

Agents: An agent is one who acts for and by authority of a human user. To collaborate, users need a way to identify who has taken action on a design model so that they may carry on discourse with that agent about those actions. Agents must have a **unique identity** that is linked to a user and capture a **procedure** or body of **knowledge**. They must also be capable of **communicating** with other agents so that discourse may take place. All actions taken in the design environment should be **traceable** back to a particular agent. Agents which embody a domain of knowledge may be proactive or reactive, but they must always work for and be **accountable** to a particular user.

Distributed and asynchronous information exchange: Many engineering and architectural design activities are undertaken by groups of people that are separated by gulfs of time and space. It is highly likely that members of a design team will be working on related tasks of a project at any particular time and it is not guaranteed that users will always have access to a central data server. Therefore, users require **connectivity** that allows them to **request** and **send selected information** about the design model to other users, be able to work in **privacy** on an computer that is **isolated** from a network (e.g. a portable computer on an airplane), and be able to **update** other users about aspects of a design model that have changed when connectivity is re-established.

Conflict management: An identifying feature of col-

laboration is that individuals represent different **viewpoints**. This means that they have different goals, priorities, and even different ways of representing information. They may not always be aware of how their subtasks interact. The Discourse Model must be able to **identify** conflicts, assist in **negotiations**, help to exchange **rationale**, and keep track of conflicts that have been **resolved**.

Structure

The principle components of the Discourse Model structure are shown in *Figure 1*. The *workspace* component satisfies functional requirements for a rich modeling environment and agents. The *virtual workspace* component meets the requirement for distributed and asynchronous communication. The *conflict management* component manages information relative to the detection, negotiation, and resolution of conflicts.

Workspaces

An implementation of the Discourse Model must provide a *workspace*, which is a type of blackboard system that gives users a private environment for design synthesis and analysis. Designs are synthesized from artifacts, which are either *frames*, *semantic links*, or *constraints*. Artifacts are created and connected by *agents*, including a *user* agent controlled directly by the human user. Depending upon the implementation, a CAD graphics interface can provide visualization capabilities and allow users to interact with a graphical representation of the artifacts. To participate in the Discourse Model, all artifacts contain data about the user and agent that created them, their time of instantiation, and their status (either :IN (valid) or :OUT (not valid)). Implementations of the Discourse Model must support the concepts of *frames*, *semantic links*, and *constraints*, described below.

A frame is a specification for a building block of a system being designed, whether physical or abstract. Examples are walls, windows, ductwork, space, and requirements. Attributes of frames, such as length, height, or colour, are called *slots*. Frames can be defined dynamically. An instantiation of a frame is called an *instance*. Instances and slots may have textual *annotations* attached to them if desired.

Semantic links are used to build relationships between instantiations of frames. Examples might include a *has-part* link between a pump and its impeller or a *supported-by* relationship between a column and a beam. New link types must be dynamically definable. An instantiation of a semantic link type is called a *link*.

Constraints build and enforce relationships, both symbolic and mathematical, between slots of frame instantiations. They are also used to detect conflicts between users. For example, if the user has an opinion

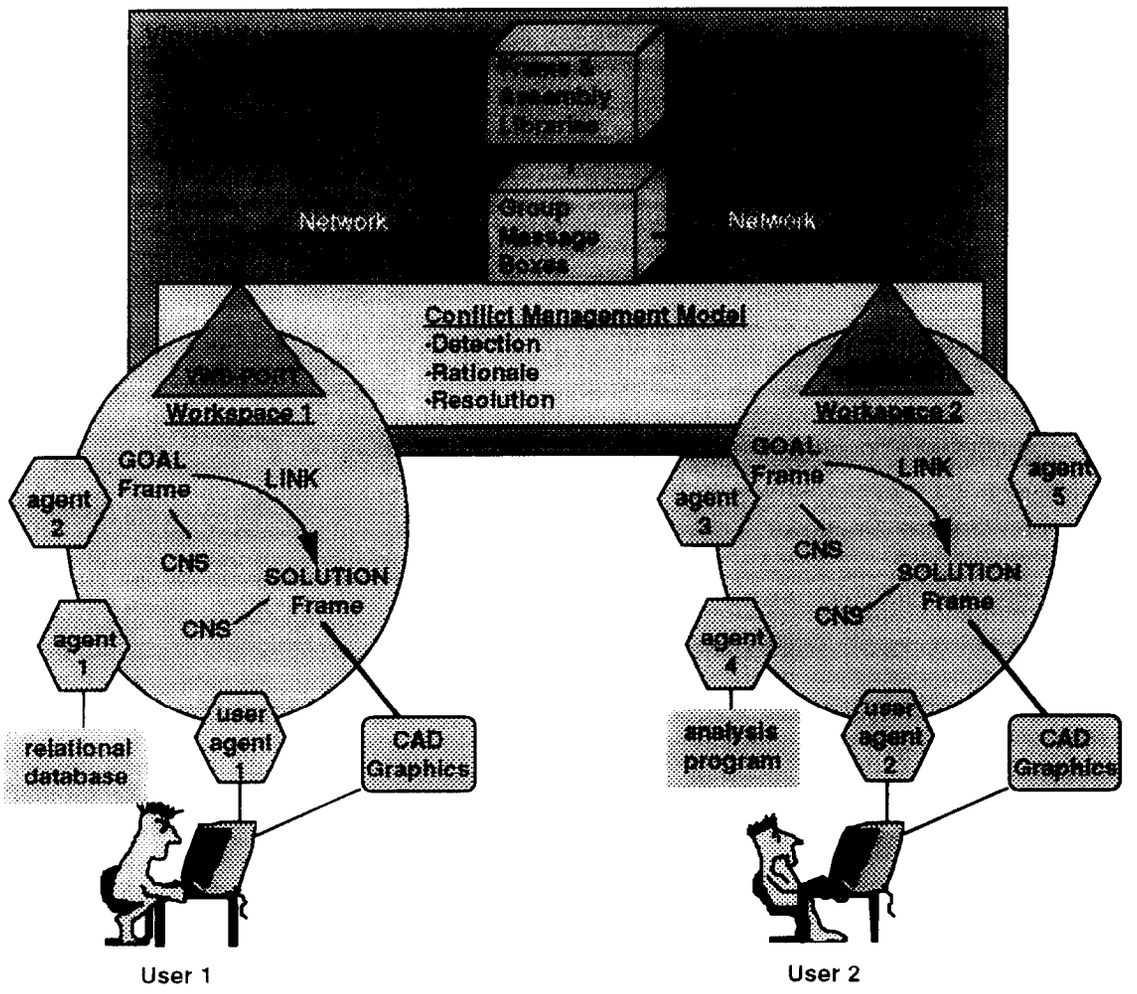


Figure 1 The structure of the Discourse Model. Principal components are workspace, virtual workspace, and conflict management

that a window should use glass with low emissivity, that opinion would be represented by attaching a constraint to the *type* slot of that window, annotated with information about the user and agent that hold that opinion. New constraint types can also be defined dynamically. An instantiation of a constraint type is called a *constraint*.

The Discourse Model uses knowledge level agents[12], which manipulate artifacts in a workspace. An agent represents the user during the design process by manipulating or reacting to artifacts present in the design context. In fact, human users are not allowed to directly manipulate the workspace. Instead, a special *user* agent must be provided by the workspace to act for the human user.

Virtual workspaces

The Discourse Model introduces a *Virtual Workspace* (VWS) that supports asynchronous and distributed collaboration. Asychronous collaboration means that users do not need to be connected to the network simultaneously or continuously in order to collaborate. The term virtual workspace signifies that to users and agents, there appears to be one large workspace rather than many small ones. A set of users that are joined together in a virtual workspace is called a *group*. Each workspace in the group is uniquely identified by a *userid* code, associated with the human user of the workspace.

Workspaces communicate using an electronic mail system called *VWS-Mail*. The rationale for using a mail system analogy is to provide *store-and-forward* capabilities. This means that workspace can send a message to another workspace with the assurance that if the other is not on-line, the message will be stored until accessed. Store-and-forward permits a group member to isolate a computer from the network for a period of time, work on the design (and perhaps travel), and then reconnect the computer to the group and get updated information.

Workspaces are connected to the VWS-Mail system by a VWS-PORT. This port acts as a representative of a user's workspace and interacts with other VWS-PORTs by formulating and sending messages in virtual workspace language (VWL), which is an inter-workspace language intended for use by programs conforming to the Discourse Model. VWL uses KQML[1] primitives to send messages. Its principal verbs are: INTEREST, SHADOW, LINK, CONSTRAIN, CONFLICT, and RATIONALE. It also receives and interprets messages from other users' VWS-PORTs. The function of the VWS-PORT is to create and maintain the virtual workspace that allows users to behave as if all of their workspaces have been joined together.

Conflict management

A principal contribution of the Discourse Model is that it provides data structures and processes for detection and negotiation of conflicts. Conflicts are kept in a *conflict event* data structure. Conflict events are divided into sets, according to their status; deferred, active, or resolved. Implementations of the Discourse Model must support *conflict events* and *conflict event sets*, described below.

When a conflict is identified, a conflict event is generated. Information stored with the conflict event includes a classification of the conflict type, a list of users that are interested in that conflict, which user originated the conflict, when the conflict was detected, which user will arbitrate the conflict, and a list of opinions of interested users. The user that arbitrates the conflict can add a deadline for resolution and the final outcome of the negotiations. A *deferred-conflict* set holds conflict events for which the conflict originator has elected to delay negotiations. An *active-conflict* set contains events that are currently being negotiated. A *resolved-conflict* set holds events that have been resolved.

Process

This section describes the Discourse Model process, which is illustrated in *Figure 2*. The phases of the process are *analysis and synthesis, information sharing*, and *conflict management*.

Analysis and synthesis

During the design and analysis phase, the designer determines functional requirements of the design, generates design alternatives, and analyses proposed solutions to determine whether they meet requirements. Implementations of the Discourse Model must support the *design transformation* and *design alternative* processes described in this section.

The Discourse Model uses a transformational model of design, wherein the data contained in artifacts such as frames, constraints, and semantic links specify the state of the design model at any particular time. To move from one state to another, an agent applies a transformation rule. After the transformation is completed, the artifacts in the workspace retain information about who made the transformation as well as information that can be used to generate rationale.

Many decisions must be made during the design process. Often, several alternatives will be generated that may satisfy any one requirement. The Discourse Model requires that frames be descendants of either a FUNCTIONAL_UNIT frame or a DESIGN_UNIT[13] frame and that instances of those frames be linked by *requires* semantic links. Marking an instance or link as :OUT will cause subsystems of that link or instance to also be considered :OUT. This provides a computationally inexpensive procedure for switching between design alternatives, based partly on an assumption-based truth maintenance system[14].

Sharing information

The virtual workspace described in a previous subsection consists of an electronic mail system, VWS-PORTs, and a virtual workspace language. To use these components, the Discourse Model process provides methodologies for the identification of agent and user *interest sets* and for *shadowing* of design artifacts (frames, constraints, and semantic links). The interest set of an agent identifies that agent's viewpoint, that is, the types of artifacts that the agent considers to be important to its task. The shadowing methodology specifies how artifacts are to be shared between users and maintained in a consistent state. Implementations of the Discourse Model must support *dissemination of interest*

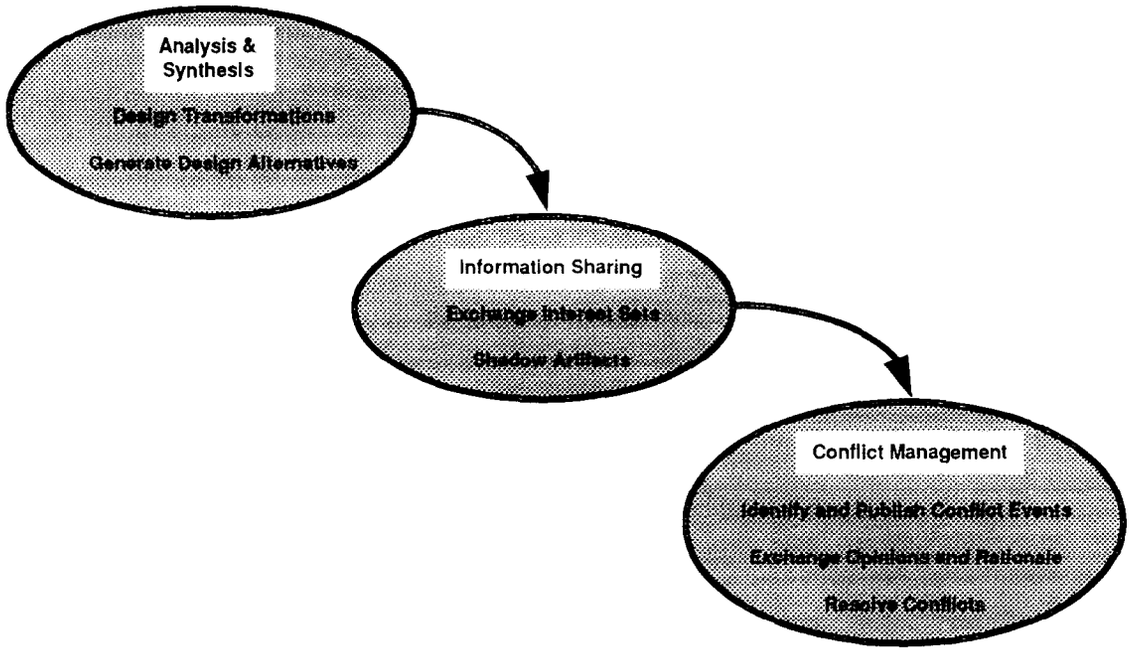


Figure 2 The Discourse Model process has three distinct phases. A designer starts by analysing requirements and synthesizing proposed design alternatives. The users in a collaborative design group then share their proposals as opinions. Finally, conflicts are identified and negotiations lead to resolution

sets and the *shadowing* process described in this section.

The concept of interest sets has become common in distributed agent and database systems[13,14]. An interest set contains a list of frames and slots that an agent considers to be important to performing some task. If a frame name is listed in the interest set without specifying any slots, then an implicit assumption is made that the agent is interested in all slots of the frame. If any slots are specified, then it is explicit that the agent is interested in only those slots. Interest sets are maintained by each agent in a workspace. Unlike other systems, the Discourse Model allows interest sets to be defined explicitly as part of the agent's definition or to be *inferred* dynamically while the agent is performing a task. The following four categories of interest inference are defined for the Discourse Model:

- Rule reference (RR): the set of frame instances and slots that are referenced in a rule owned by an agent.
- Constraints atttribute (CA): the set of frame instances and slots that are attached to a constraint owned by an agent.
- Algorithmic reference (AR): the set of frame instances and slots that are required as input or changed by output of a formal analysis algorithm (software program or function) used by an agent.
- Object instantiation (OI): whenever an agent instantiates a frame, it can reasonably be inferred that it is interested in the existence of similar frame instantiations in the environment. Therefore, if the user agent instantiates a WINDOW, it is inferred that the user agent is interested in WINDOWs.

Upon joining a collaboration group, the VWS-PORT for each user's workspace will poll all of the agents for their interest sets. Agents are required to notify the VWS-PORT if their interest sets change. A *user interest set* is constructed, based upon the union of the agent interest sets. The user interest set is broadcast to all of the other users in the group using the VWS-Mail system and VWL. If a new interest is added to any of the agent's interest sets, the VWS-PORT will be notified and the new interest will be broadcast. After broadcasting its interest set, the VWS-PORT will check its own VWS-Mail box for messages about other user's interests. The VWS-PORT is required to maintain a database of the interest sets of other users.

A complete KQML package expressing interest would appear as follows:

```
(PACKAGE :FROM 'mec01'
         :TO 'ALL'
         :ID 2944324085
         :MESSAGE
         (TELL :content    (INTEREST 'mec01'
                  WALL)
               :language   VWL
               :ontology   (arclib)))
```

This INTEREST message states that the 'mec01' user had an interest in the WALL frame from the 'arclib' library. Note that in the example, no slots are given, implying that 'mec01' is interested in all slots of the WALL frame. If slots had been specified, then only those slots would have been included in the interest set. The VWS-PORT keeps a database of the interests of other users in its external-user-interests slot. Upon receipt of an INTEREST message, a record is inserted recording the userid, the frame, and any slots specified.

A *shadow copy* is a copy of an artifact in a workspace other than the one it originated in. The VWS-PORTs

in a virtual workspace cooperate with each other to maintain copies of artifacts in all of the users' workspaces that have expressed an interest in them. When a VWS-PORT is notified about another user's interest in a frame, it queries its own workspace for instances of that frame. If any are found, a *shadowing* message is sent to the interested user describing the instance and its slot values. Messages are also sent about semantic links attached to the instance. The VWS-PORT is required to monitor the instance and report any changes to its slot values and links. As transformations are applied to the workspace, the VWS-PORT will compare new instances and links and send shadowing messages as necessary.

When a shadowing message is received from another user, the receiving VWS-PORT instantiates a shadow copy of the instance and any attached semantic links. Artifacts are annotated with the sending userid, uniquely identifying them as shadow objects. To prevent name collisions between instances, the userid of the instance owner is pre-pended onto the instance name. For each slot value received, a constraint is attached to the corresponding slot, signifying the opinion of the owning userid. A shadowed slot cannot be changed without invoking the conflict management process. To prevent circular message traffic, shadow instances are not passed on to other workspaces. For each artifact, there is a only one original plus a shadow copy for each interested workspace.

Conflict management

After a set of artifacts has passed through the analysis and synthesis and information sharing processes, all users in a group have access to either the original or shadow copies of the aritifacts. If any of the users or their agents have a conflicting opinion, the conflict management process is initiated. Implementations of the Discourse Model must support *identification of conflicts, dissemination of information about conflicts, exchange of rationale*, and *recording of conflict resolution* as described in this section.

The conflict management process is initiated by identification of a *conflict event*, described earlier. Two types of conflict events, *slot* and *link*, are recognized by the Discourse Model. A slot-type conflict event is initiated by violation of a constraint attached to a slot of a shadowed instance. A link-type conflict event is initated by an attempt to mark a semantic link as :OUT. When an agent applies a design transformation that causes either of these events to occur, the VWS-PORT is notified and the agent is queried about how it should proceed. The choices are to withdraw the design transformation, initiate negotiations immediately, or make the change but defer negotiations. If either of the later two choices are selected by the agent, the VWS-PORT will instantiate a conflict event and place it in either the active conflict set or the deferred conflict set. When a conflict event is placed in the active conflict set, either directly or from the deferred conflict set, the VWS-PORT will send a message to the owner of the artifact that is in contention. In the Discourse Model, the owner of an artifact is always the arbiter.

Upon receipt of a message about a conflict, the VWS-PORT of the arbiter's workspace assumes responsibility for management of the conflict. As a first step, a set of users that have expressed an interest in artifacts involved in the conflict event is compiled and a deadline is set for negotiations. The updated conflict event is then *published* to all of the interested users. The VWS-PORT of each user that receives the conflict message then places the conflict event into its own active conflict set. If a conflict event about the same artifact is in the deferred or resolved conflict sets, this might or might not be brought to the user's attention, depending upon the implementation.

After receiving notice of a conflict event, each user may express an opinion about the desired outcome of the conflict, which will be sent to the arbiter. The arbiter's VWS-PORT, in turn, will forward the message to the other interested users. Thus, each user is kept informed about the status of the negotiations and all opinions that have been expressed. Each user has the option of changing their opinion at any time until the deadline is reached or the conflict resolved. Although not described herein, the Discourse Model also provides a methodology for one user to request rationale from another user.

The final step in the conflict management process is the resolution of the conflict event. The Disourse Model does not presume to automatically resolve conflicts. The method of resolution, whether democratic or dictatorial, depends upon the parties involved in the collaborative effort. Rather, the Discourse Model seeks to maintain a clear record of the nature of the conflict event and its final resolution. Upon reaching the published deadline for resolution (if any), the arbiter of the conflict selects one of the opinions recorded in the conflict event and labels it as the *resolution* of the conflict event. The resolution is also annotated with a date, time, and optional textual comment. The resolved conflict event is then moved into the resolved conflict set. The arbiter's VWS-PORT then broadcasts the resolved conflict event to all of the interested users. Their VWS-PORTs receive the message and move the conflict event into their own resolved conflict sets.

IMPLEMENTATION EXAMPLE

This section describes a design scenario extracted from a more detailed concurrent engineering test bed project developed by the US Army Construction Engineering Research Laboratories. The scenario will illustrate issues that may arise during the early conceptual design phase of the design of a standard US Army Fire Station. The scenario was implemented in a system called ACE (Agent Collaboration Environment).

Establish group

For this scenario, a work group will be created consisting of a project manager, an architect, a mechanical engineer, and an operation and maintenance representative. The project manager ('prm01') is the first to receive information about the project, which she gets though a planning document labelled 'DD-1391, Military Construction Project Data'. Her first action is to view available agents and to enable a *project manager* agent that helps to initialize projects. ACE uses checklists as plans and status indicators for agents. The

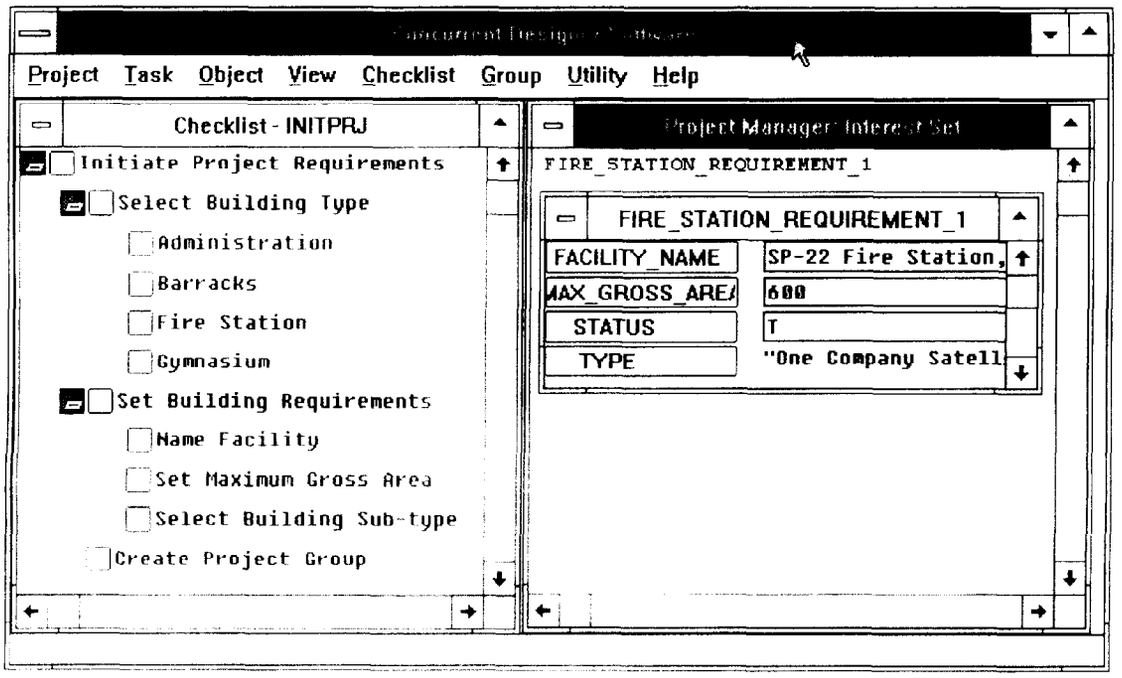


Figure 3 Checklist requirements can also be filled in by editing an instance directly from the *Project Manager* adaptive focus viewer. The checklist will dynamically keep track of the state of completion regardless of how the information is entered into the workspace

project manager agent has a checklist for initiating the project using three main steps:

(1) select a building type;
(2) set building requirements; and
(3) create a project group.

In *Figure 3*, Steps 1 and 2 are complete.

Until now, the project manager has been acting alone and the workspace has been entirely private. Once the initial functional requirements are established, the project manager is ready to assemble a project group. In ACE this is done by a software link to project management software that contains a database of available labour resources. When the group is assembled, a VWS-Mail repository is created for each member. New members may be added at any time. Once the project manager joins the newly created group, her workspace is ready to supply the initial functional requirements to any members that express an interest.

The architect ('arc01') is the next participant to play a role in the design process. After receiving notification by electronic mail that she has been assigned to a project, 'arc01' starts ACE and joins the newly formed project group. The architect has an *architectural programming agent* that establishes detailed functional requirements for buildings that have standard definitive designs (i.e. fire stations). When the agent is loaded, the VWS-PORT polls the agent for its interest set. This set, which includes the BUILDING_REQUIREMENT frame, is broadcast to all other numbers of the design group. When the workspace of the project manager learns of the architect's interest in BUILDING REQUIREMENTs, it will send a VWL SHADOW message to 'arc01' containing information about *fire station_requirement_1*. The VWS-port of 'arc01' will then instantiate the shadow copy in its workspace.

The architectural programming agent does not use a checklist. Instead, it applies a series of transformation rules that construct a set of design artifacts based upon the description given by the project manager. In this instance, the transformation rule **ONE** COMPSAT FIRE_STA is triggered by a requirement for a one company satellite fire station. It consists of the requirements and initial spaces shown in *Figure 4* in the **Architectural Programming Agent: Links: requires** window. Note that the requirements and solutions are connected by *requires* semantic links. Also shown in this figure are *has-part* and *sub-requirement* views. Although it is not readily apparent, DESIGN_UNITs are connected by *has-part* links and FUNCTIONAL_UNITs are connected by *sub-requirement* links. Once the programming agent has finished, the user is free to change requirements as necessary.

Building layout

As yet, there is little to visualize in the conventional CAD sense. Another agent, the *architectural layout agent*, is responsible for choosing the locations of the actual rooms in the fire station. This agent also uses transformation rules to instantiate walls, doors and windows. *Figure 5* shows the initial building enclosure as designed by the layout agent and rendered by a parametric link with CAD software. The walls in this building are attached by constraints so that movement of one wall will cause other walls to grow or shrink to accommodate it. Doors and windows are attached to the walls by *has-part* links and located relative to the wall that they are attached to, so that movement of a wall results in movement of its doors and windows. Once the layout agent has inserted and positioned the walls, windows, and doors, the user is free to move them about as desired. This can be done by changing parameters symbolically in ACE or by moving the graphically rendered version in the CAD environment.

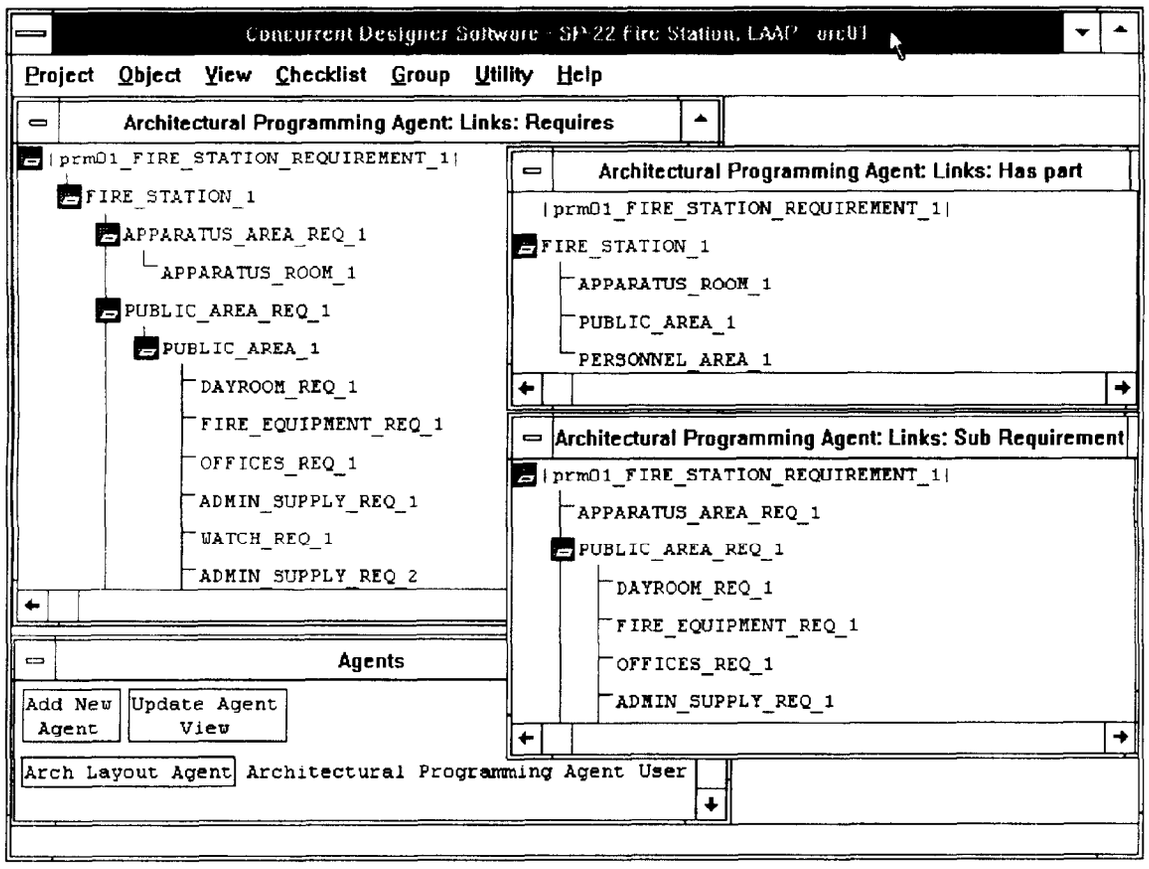


Figure 4 The Architectural Programming Agent lays out building requirements using transformation rules

A momentary digression is in order at this point to consider the implications of this particular method of communicating information about design artifacts, compared to the exchange of drawing files. In the Discourse Model, information about frame instances and their semantic relationships is exchanged symbolically. Definitions for these artifacts are contained in shared artifact libraries, which may also contain meth-

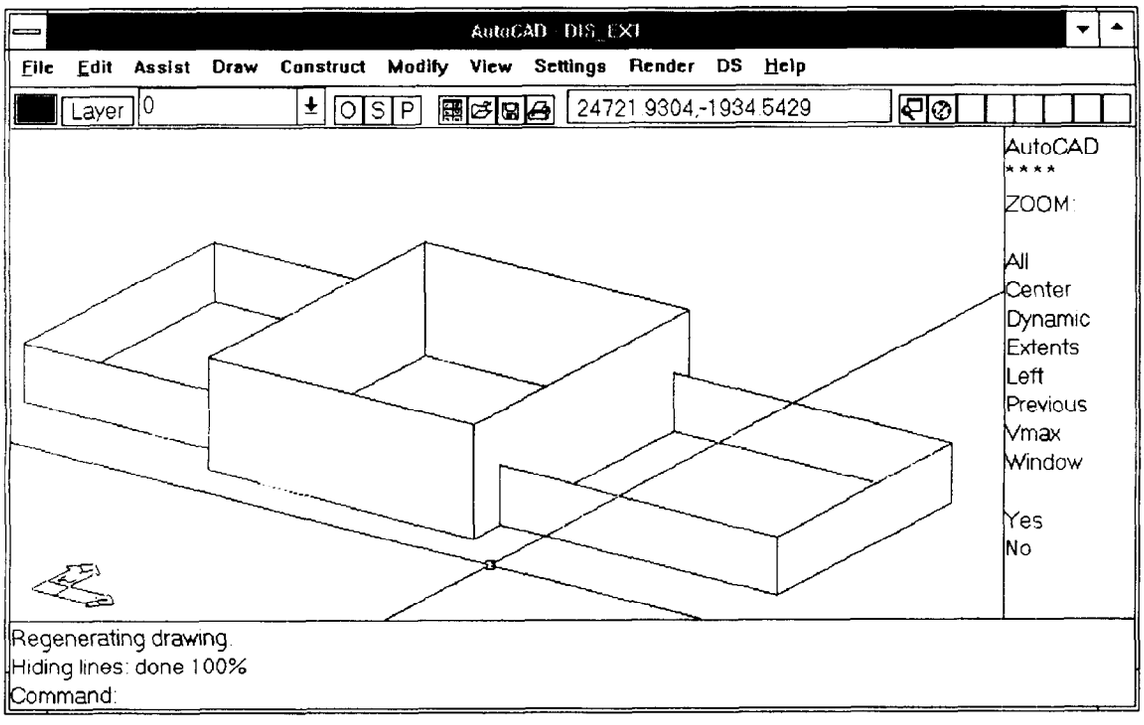


Figure 5 The initial building enclosure as designed by the Architectural Layout Agent

ods for rendering them in different CAD software implementations and from different perspectives (e.g. plan, isometric etc.). Thus, it is trivial for one group member, using CAD software package A to exchange information with another member using CAD software package B. There is a cost, of course, as the artifact libraries and drawing methods must be defined for the CAD software packages used by the members of the group.

Conceptual stage energy analysis

After the initial floor plan is laid out by the architect, the mechanical engineer can perform a preliminary thermal analysis on the design. Key decisions about a building with respect to thermal performance are often made long before the mechanical engineer is consulted. In this example, the mechanical engineer will analyse the building from a thermal point of view at the conceptual design stage and make suggestions to improve its thermal performance

One obvious but often ignored design consideration is the amount of glazing in a building. Too much glass on the south side of a building can result in excessive cooling energy loads in the summer, while too much northern glass results in excessive heating bills in the winter. There are too many variables, however, for oversimplified rules of thumb. New window systems that use double and triple panes and low-emissivity coating can result in north facing windows that admit more heat than they lose. In many climates, the only way to accurately choose a glazing strategy it to conduct an hourly heat balance simulation for the building on an annual basis.

Mechanical engineer energy interests

The mechanical engineer is interested primarily in the geometry of the building as laid out by the architect. His *thermal analysis agent's* interest causes relevant details of the building's geometry to be shadowed in the mechanical engineer's workspace. By filling in such details as thermal zones and building component thermal properties, the thermal analysis agent creates a simplified thermal model of the building. Using the BLAST[15] thermal analysis program, an initial estimate of annual energy consumption can be calculated for this building. Although not highly accurate, this initial assessment serves as a point of departure for exploring design alternatives.

Conduct parametric studies

A useful activity at this point of the design process is for the mechanical engineer to conduct parametric studies. Parameters that could include the size and type of glazing, insulation of the walls, reflectivity of the roofing system, or orientation of the building. For this example, the energy analysis agent will try three case studies, including the original, a configuration with

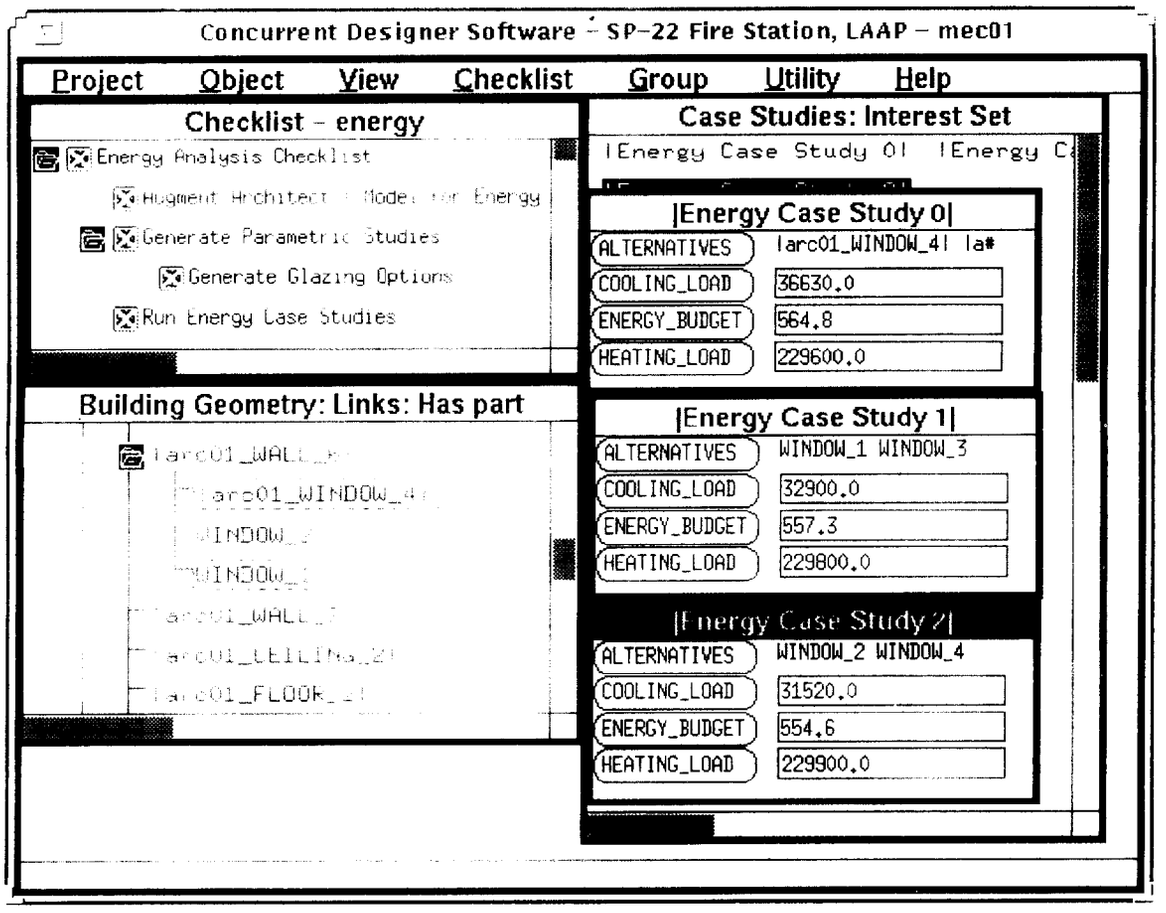


Figure 6 The Energy Analysis Agent generates glazing case studies for the mechanical engineer. Each case study contains a list of alternatives that are included. To run a case study, the agent changes the context so that the alternatives for the indicated study are valid, extracts information for the external thermal analysis program, runs the program, and brings energy metrics back into the workspace

minimal glazing, and one with about 30% of the south wall glazed.

Figure 6 summarizes the activities of 'mec01'. As shown by the checklist in the upper-left corner of the screen, the first action of the *energy analysis agent* is to augment the architect's building model. Next, the agent conducts parametric studies by finding the south walls of the building and adding two glazing alternatives to the architect's initial proposal. This is done by adding FUNCTIONAL_UNIT instances to the walls and linking the window instances through *requires* links. As only one solution can be valid at a time, two of the window instances are surrounded by red (dashed) lines and the currently valid solution is marked by a green (solid) line. The ENERGY_CASE_STUDY instances are shown on the right side of the figure. Each study instance has an alternatives slot that contains a list of the instances that must be valid for the case to apply. To run an analysis, the agent makes the members of the list valid and then calls the thermal analysis program. This approach, which uses the ability of the Discourse Model to inexpensively switch contexts, allows the agent to examine alternative solutions.

Conflict identification

For each study, a cooling load (kWh), a heating load (kWh), and total energy consumption (kWh/m^2) is calculated on an annual basis. Based on decreased cooling load, 'mec01' prefers the option of 30% south glazing over 'arc01's' initial choice. Generating the different alternatives created conflict events between 'arc01' and 'mec01'. The energy agent deferred the negotiations so that calculations could be completed to build up supporting evidence for choice.

Before beginning negotiations between the architect and the mechanical engineer, let us also include the input of the operations and maintenance representative ('oam01'). The reason for this is to illustrate that decisions often involve more than two people, so a conflict management strategy must take multiparty interest into account. In this example, the dispute is about configuration. That is, which window system should be selected? For this example, the 'oam01' user has an agent called *building maintainability agent*, that has the duty of checking designs for unreasonable maintenance requirements. One of the maintainability criteria that it looks for (whether justified or not) is to minimize glass that needs to be cleaned. It does not need access to as much of the building geometry as 'mec01', just enough to ascertain glazing levels. These information requirements are entered in the agent's interest set. The effect of this entry is that the 'oam01' agent will be a party to any conflicts that arise.

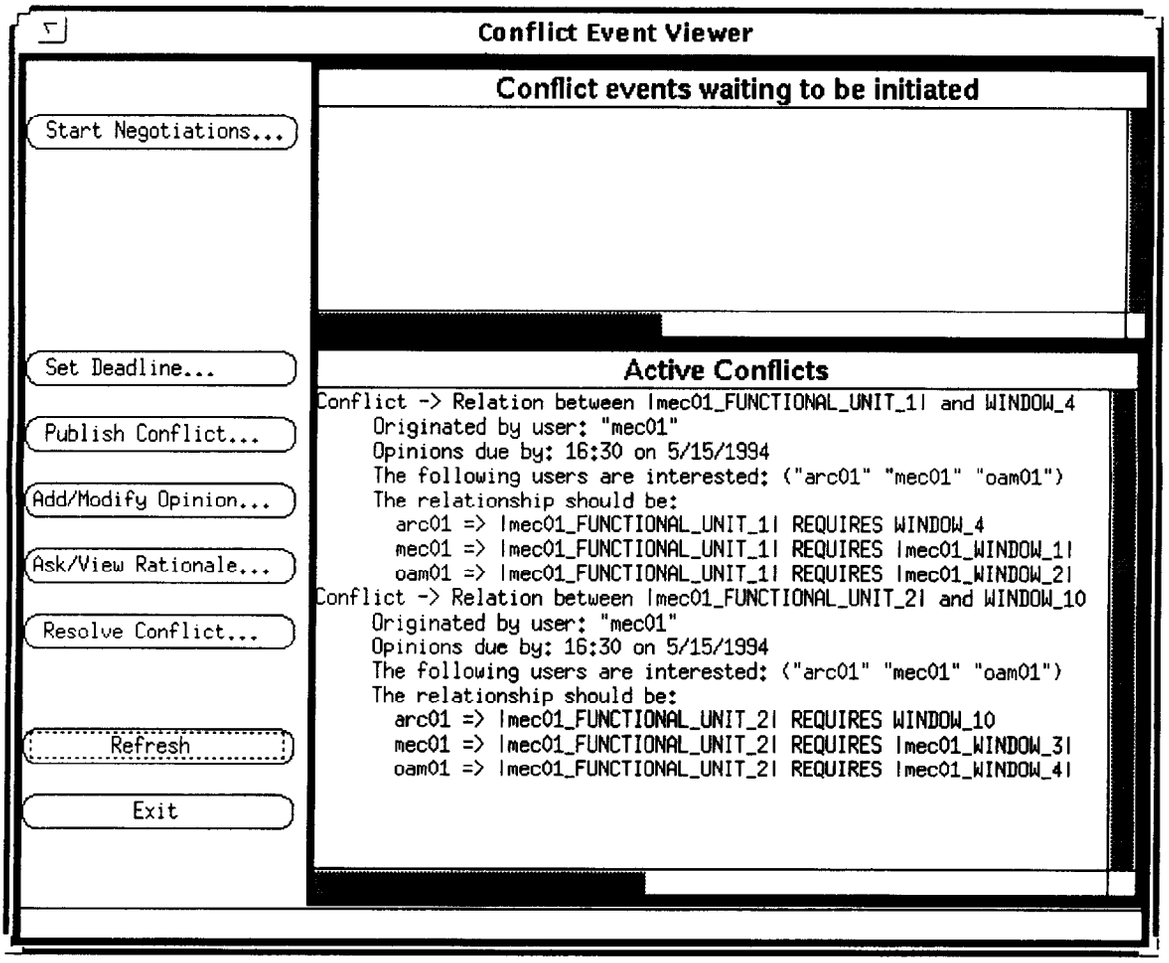


Figure 7 The architect receives a conflict message initiated by the mechanical engineer. Since the link in conflict is owned by the architect, she becomes the arbiter. The screen also shows the opinions of all of the interested users

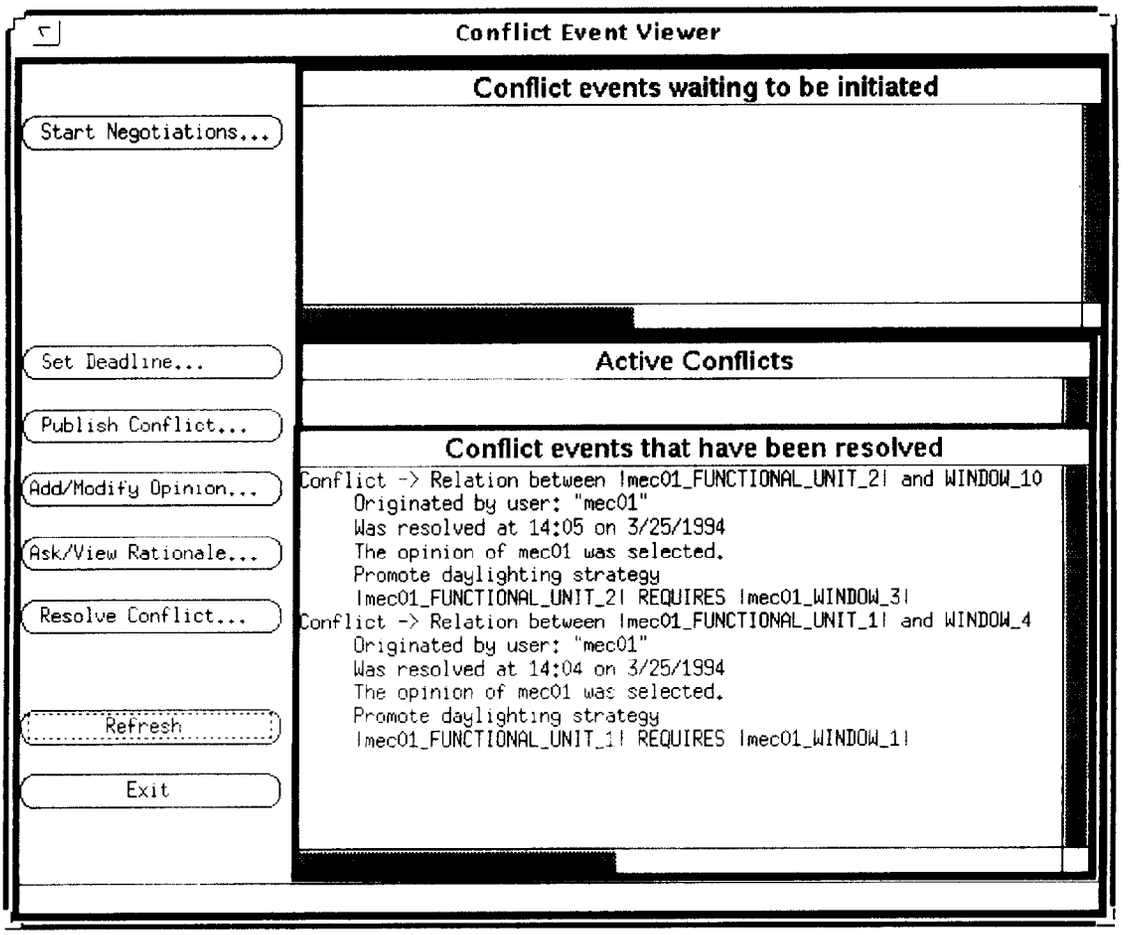


Figure 8 Once the conflict is resolved, the conflicts are moved to the resolved conflict set of the VWS-port. The Conflict Event Viewer shows the status of resolved conflicts, including when they were resolved and by whom. Information about the other options that were considered is still stored in the conflict event, but not shown

Negotiation

Returning to 'mec01', it is time to initiate negotiations. This will be done by the user, although it could as easily be initiated by the thermal analysis agent directly. The 'mec01' user selects the conflicts about glazing from the deferred conflict set (not shown) and sends them to the owner of the initially proposed configuration, 'arc01' using a CONFLICT message.

When user 'arc01' receives the conflict message, her workspace analyses the content, recognizes that 'arc01' is the arbiter of the conflict, and instantiates a conflict event in its active conflict set. It also checks its database of user interest sets to determine which other users might be interested in the frames for which instances are linked. Updated conflict messages are sent to all of the interested users, including the user that initiated the conflict. In addition to information about the nature of the conflict, the arbiter adds a deadline by which interested users must submit opinions.

Both 'mec01' and 'oam01' receive the updated conflict message and may venture their own opinions, if any. As opinions are received by 'arc01', they are combined with existing opinions and rebroadcast to interested users. The arbiter ensures that every user has a complete set of all of the opinions submitted by every user. *Figure 7* shows the conflict event, with a complete set of opinions. Note that 'oam01' selected a different option than had been included in the original conflict event. In this case, 'oam01' user prefers the minimal glazed option, 'mec01' prefers the 30% solution, and the architect wants to keep maximal glazing. At this point, the users could exchange requests for rationale, which is sent using the VWL RATIONALE message (not shown).

Resolution

In the end, the final decision about which link to choose is made by the architect, acting as arbiter of the conflict event. To resolve the conflict, she selects the 'Resolve Conflict...' button on the conflict management screen. After selecting the 30% solution, she makes an annotation describing her reasoning.

Once a choice is made, the workspace sends notification of the conflict resolution status to the interested users in the group. Upon receipt by each user, the local workspace moves the conflict event into its resolved conflict set and marks the correct link to make the 30% solution valid. Changing to another link after this point will result in generation of another conflict, re-initiating the conflict management cycle. Resolved conflict events may be reviewed by opening the resolved conflict viewer (*Figure 8*). This viewer shows information about each conflict event, including its initiator, the opinions presented, and the final decision. This viewer

provides access to a record of resolved conflicts and can be used to print a resolved conflict report.

CONCLUSIONS AND FUTURE WORK

This paper has presented both a structure and process for collaborative engineering, called the Discourse Model, that allows groups to cooperate in a distributed and asynchronous way. The structure specified the artifacts of design, including a workspace that incorporates frames, constraints, semantic networks, libraries of sharable design objects, agents, and a virtual workspace language based in part on KQML. The process contributes new procedures for identifying agent interest sets, applying state transformations to the design model, identifying conflicts between designers, and tracking resolved conflicts. Due to limitations of space, the model could not be presented in great detail. A more thorough treatment, especially of the virtual workspace language, is given by Case[2].

As in many research programs, questions of practicality and utility abound. To pursue these questions, a facility design testbed project is in progress. This project, from which the sample scenario was derived, involves design and construction planning for a fire station. Domains included in the research include architecture, construction planning, cost estimating, operations and maintenance, mechanical engineering, space planning, and structural engineering. Software agents are being developed to assist these domains in such tasks as designing footings and conducting energy analysis. Proposed new work will address facility management and sustainment issues that require the collaboration of people from different domains and professions.

The virtual workspace language is still in its infancy with respect to expressiveness and generality. One goal is an extension that will allow users to send agents into the workspaces of other users (with permission) so that they can act as design advisors and critics in real-time. More work needs to be done on the exchange of decision rationale, a topic not covered in this paper but supported by VWL. Additional implementations of the Discourse Model in heterogeneous knowledge-based and CAD systems are needed as well, to test the generality of the model. In addition to the implementation in ACE, a prototype implementation has been completed in Design^{++} (a commercial soft ware package) and one is planned for SWIFT.

REFERENCES

1 Finin, T, Weber, J, Genesereth, M, *et al.* 'Specification of the KQML agent-communication language' *Draft Request for Comments* University of Maryland (Feb 1994)

2 Case, MP 'The discourse model for collaborative design: A distributed and asynchronous approach' *PhD Thesis* University of Illinois at Urbana-Champaign, USA (1994)

3 Hardwick, M, Spooner, SL, Downie, B, Hvannberg, E and Hehta, A 'Change processing tools for an object-oriented data management system' *Proc. 17th NSF Design and Manufacturing Systems Grantees Conference* (1990)

4 Howard, HC and Rehak, DR 'KADBASE: Interfacing expert systems with databases' *IEEE Expert* Vol 4 No 4 (1989) pp 65–66

5 Genesereth, M, Huyn, P and Letsinger, R 'Agent-based framework for concurrent engineering softwarc' *Proc. CE&CALS '92* and Society for Computer-Aided Engineering (1992) pp 135–150

6 Case, MP, McConkey, I, McGraw, K and Lu, SC-Y 'Multiple cooperating knowledge sources for the design of building energy systems' *ASHRAE Trans.* Vol 1 No 2 (1990) pp 490–500

7 Herman, AE and Lu, SC-Y 'A new modelling environment for developing computer-based intelligent associates with an application in machining operation planning' *Trans. North American Manufacturing Research Institute of SME* (1989) pp 260–265

8 Pohl, J, LaPorta, J, Pohl, KJ and Snyder, J 'AEDOT prototype (1.1): An implementation of the ICADS model' *Design Institute Report, CADRU-07-92* California Polytechic State University, San Luis Obispo, USA (1992)

9 Sriam, D, Logcher, R, Groleau, N and Cherneff, J 'DICE: An object oriented programming environment for cooperative engineering design' *Technical Report IESL-89-03* IESL, Department of Civil Engineering, MIT, USA (1989)

10 Lu, SC-Y, Smith, K, Herman, A, Mattox, D, Silliman, M, Lucenti, M, Jacobs, J, Chazin, D and Case, M 'SWIFT: System Workbench for Integrating and Facilitating Teams' in Special Issue on Enabling Technologies for Concurrent Engineering, *Int. J. Intell. Cooperative Info. Syst.* in press (1994)

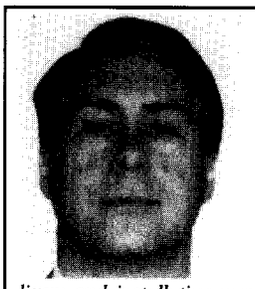
11 Wong, A and Siriram, D 'SHARED: An information model for cooperative product developement' *Res. Eng. Des.* Vol 5 No 1 (1993) pp 21–39

12 Genesereth, MR and Nilsson, NJ *Logical Foundations of AI* Morgan Kaufmann, CA (1988) p 314

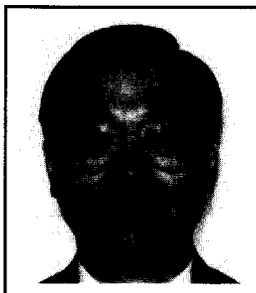
13 Woodbury, R (personal communciation) Terminology for the SEED (Software Environment for Early Design) project at Carnegie Mellon University, USA (1993)

14 deKleer, J 'An assumption-based TMS' *Artificial Intelligence* Vol 28 (1986) pp 127–162

15 Lawrie, LK 'Day-to-day use of energy analysis software' *Energy Eng.* Vol 89 No 5 (1992) pp 41–51



Michael Case received a BS in mechanical engineering from Cornell University, USA, in 1980 and a PhD in mechanical engineering from the University of Illinois, USA, in 1994. He is currently Chief of the Engineering Processes Division for the US Army Construction Engineering Research Laboratories in Champaign, USA. This division conducts research in processes and automation support for facility delivery and installation management, including CAD, cost estimating, planning, simulation, and visualization.



Stephen C-Y Lu received his BS degree from the National Taiwan University, Taiwan, and MS and PhD degrees from Carnegie-Mellon University, USA, all in mechanical engineering. He is the permanent holder of the David Packard Endowed Chair in Manufacturing Engineering in the School of Engineering at USC, USA. He is also a professor of mechanical engineering, and industrial and systems engineering. He heads the IMPACT (Improving Manufacturing Productivity with Advanced Collaboration Technology) research laboratory, and serves as the Director of USC's Center for Automation and Manufacturing Systems (CAMS). Before joining USC in January 1995, he was a full professor at the Department of Mechanical and Industrial Engineering and the Department of Computer Science at the University of Illinois at Urbana-Champaign (UIUC), USA. He was the founding Director of UIUC's Knowledge-Based Engineering Systems Research Laboratory, which pioneers the development of advanced computer technologies for automation and integration of complex engineering and manufacturing systems.