OREGON STATE UNIVERSITY COMPUTER SCIENCE DEPARTMENT

A SOAR-Based Computational Model of Mechanical Design

Ulhas S. Warrier
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331

89-30-2

A SOAR-Based Computational Model of Mechanical Design

by

Ulhas S. Warrier

A research paper
submitted to
Oregon State University

in partial fulfillment of
the requirements for the degree of
Master of Science

September 1, 1988

Table of contents

1. INTRODUCTION

Considerable progress has been made in Computer-Aided Design (CAD) techniques to assist Mechanical Engineers in the detail design stages and adaptive redesign tasks. Currently, CAD tools support the designer in system layouts, sizing of components, drafting, analytical calculations, generating NC machining data and even motion or energy simulations. However, the more critical tasks of conceptual and early layout design have not been facilitated by these computer tools. The primary reason for this is the lack of understanding of the mechanical design process and what actually goes on in the designers' mind in these stages.

An empirical approach to studying the mental processes of designers, using protocol analysis, was recently undertaken by Ullman, Stauffer, and Dietterich [1-4]. The work described in this paper attempts to translate some results of that study of mechanical engineering designers to a more formal and detailed level by building a computational model of the design process. The main contribution of the present work is a control structure for the design model proposed by the protocol study.

A computational model of Design, even at an incomplete level, can take us closer to an understanding of the Design Process. For the construction of such a model, it is required to formalize the knowledge involved, the processes, and their interconnections. Like any expert system knowledge base, the pieces of domain knowledge (separate from the control knowledge) used in design have to be identified. The difficulties in building such a model can help to bring forth the deficiencies in the protocol analysis method and suggest improvements. The model can serve as a basis for testing representation techniques for the design knowledge and assessing their adequacy. Similarly, appropriate software architectures needed for incorporating the control strategy of the design process can be studied or developed. The control structure of design, used for the computer model can be refined with further empirical studies. It could be used as a basis to verify and represent future protocol analyses and other design methodology studies.

Once a competent theory of design methodology is developed, computer algorithms and other tools can be formulated to support the routine parts of conceptual design. There could be better integration of CAD systems and design knowledge. CAD systems could be made more productive by understanding the intentions of the designer, detecting errors, suggesting alternatives, and answering design queries. Progress made towards such a formal model may also have potential applications in other fields of Engineering.

Section 1.1 of this paper describes part of an earlier work by Larry Stauffer which provides data for the work in this paper. Section 1.2 presents a section of a design protocol and an analysis of it in terms of the earlier work. Section 2.1 describes SOAR, the architecture used for the computer model and an example problem is represented in SOAR in section 2.2. Sections 2.3 and 2.4 describe some special features of SOAR, and the reasons for choosing SOAR are enumerated in section 2.5.

Section 3 describes the computer model proposed by this paper, starting with a description in section 3.1 of the design phases recognized by the model. Section 3.2 explains the representation scheme used.Sections 3.3, 3.4, and 3.5 talk about the task, episode, and operator structure respectively. The section concludes with a presentation of an analysis in terms of the SOAR model. An analysis of the protocol session in section 1.2 in terms of the SOAR model is given in section 3.6. Section 4 presents a discussion starting with the requirements for a computer model in section 4.1. Section 4.2 compares the present model with the earlier one, and section 4.3 mentions some problems encountered during the implementation. Lastly, section 4.4 lists some future directions for research.

1.1 The model developed by Stauffer -

Stauffer [4] describes a protocol analysis study in which a mechanical engineer is given the task of designing an object and is required to think aloud as much as possible during the design. The verbalizations and the motions of the designer were recorded on videotape, and this was later transcribed into readable text. The protocols were then analyzed using the videotapes and text, with the goal of capturing the overall flow of the design and identifying the various processes involved. One of the two design problems that were used for protocol study, called the 'flipper-dipper', will be referred to in this paper. The 'flipper-dipper' problem statement is briefly described as follows -

Design a mechanism that will accept a 10" X 10" X .063" aluminum plate from a worker, lower one side so that it just touches the surface of a chemical bath (to receive a chemical film), lift the plate off the bath surface, flip it over, lower and coat the other side, and present it to the worker for removal. Only three of these mechanisms need to be built.

As a result of analysis of the protocols of five engineers, a problem space model of the Mechanical Design Process was developed. The rest of this section describes this model, called the Task/Episode Accumulation Model (TEA model). The model assumes a design environment (Figure 1) based on a theory of Information Processing Psychology proposed by Newell and Simon [5].

All information concerning the design session including the design states are stored in either the short term memory (**STM**), long term memory (**LTM**) or the external long term memory (**ELTM**). Information brought in to the STM from the other two locations is acted upon by a set of **operators**, in a sequence determined by a **controller**. Thus while the STM, with its limited memory, has only information required for immediate deliberation, the LTM and ELTM have a repertoire of knowledge that the designer does not need urgently but can access by some retrieval scheme if needed.

In this framework, design is viewed as a search in a problem space. A problem space can be considered as a set of given or created design **states** and the primitive **operators** applicable to these states. One problem space may differ from another by its state representation or the type of operators used. For instance, sketching could be viewed as being done in one problem space and model-building

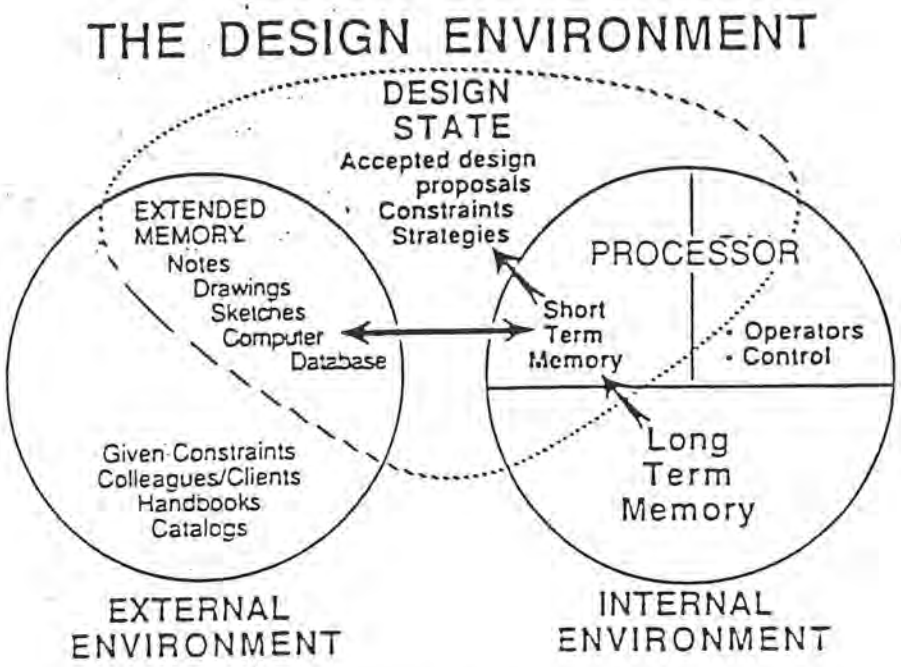


Figure 1 The Design Environment

could be in another.

Design **states** contain design elements representing information about the evolving design. These can be broadly categorized into *proposals*, *constraints* and *strategies* .

Proposals are suggestions or ideas that may or may not be accepted for achieving a goal. A proposal can be a mechanism or component being designed, or a feature of the device. For example, a proposal might be to use a cam and roller mechanism.

Constraints are design specifications that restrict the options for the remaining parts of a design. Three types of constraints may be present

- *Given* constraints are ones that the designer starts out with. These constraints can also be added later externally.
- *Introduced* constraints are posted by the designer as a result of a proposal, based on past experience.
- *Derived* constraints result from the design itself and can be relaxed or altered when they conflict with other constraints. For instance, selecting a component might introduce restrictions on geometry, configuration, or function on the rest of the design.

Strategies are meta-plans that represent sequences of actions to be taken to solve a problem. A designer might want to create a strategy or choose between strategies before proceeding with a portion of the design.

Design **operators** are primitive information processes that modify the design state by proposing new designs, evaluating proposed designs, deciding to accept or reject proposed designs. In the TEA model these operators occur at the bottom of a hierarchy of design processes, as shown in

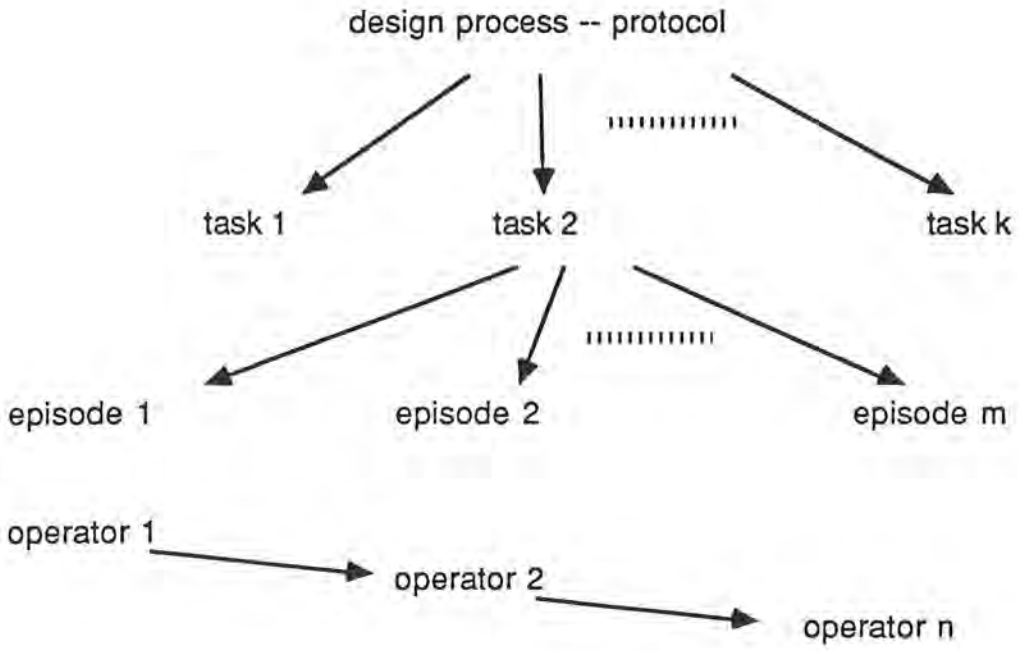


Figure 2 Processes of Design Performance

figure 2. A discussion of the processes mentioned in the figure follows, in a bottom-up order.

The TEA model contains ten operators that are classified under three types : Generation, Evaluation, and Decision.

Generation operators help to propose new design elements. There are two operators used for generation.

The *Select* operator brings information from LTM or ELTM to the STM for consideration.

The *Create* operator brings new information into STM from an obscure source, presumably as a result of steps that are not verbalized to generate information from LTM.

Evaluation operators relate or compare information (typically a proposal and a set of constraints) from a design state in order to make a decision. There are three such operators.

The *Simulate* operator reduces the arguments to be used in a comparison to the same level of abstraction and representation. This operator may involve 'hand waving', paper models, or mathematical simulations.

The *Calculate* operator is used to logically project or infer new information from the information at hand, such as adding two numbers.

The *Compare* operator compares

a) design proposal to a set of constraints to find if the proposal satisfies the constraints.

b) two constraints to check for conflict

c) two strategies to determine which one is better

Decide operators determine the fate of newly generated information after the evaluations are

done. Five operators come under this category.

The *Accept* operator includes a design proposal, constraint, or strategy under consideration as a part of the design state if the evaluation was satisfactory. It may be revoked and rejected later, however.

The *Reject* operator is invoked if the evaluation results were not satisfactory.

The *Suspend* operator is used to terminate the present problem without coming to a definite conclusion, mainly due to lack of sufficient knowledge.

Using the result of the evaluation, the *Refine* operator develops the new information to a more detailed level of abstraction.

Using the result of the evaluation, some aspects of the new information are altered, at the same level of abstraction, by the *Patch* operator.

The ten primitive operators discussed above are applied in particular sequences called **episodes**, which are focussed on primitive goals. The nature of these primitive goals changes according to the level of abstraction the designer is working in. For example, in the initial stages an episode might be 'to determine the power source for operation' and later it may be 'to fix the width of a component'.

TEA Model proposes six different kinds of episodes -

Assimilation - to gather information, usually constraints, from the ELTM or LTM into the design state.

Specify - to develop a proposal, constraint, or strategy to a less abstract level. This episode plays a major role in the protocols.

Planning - to develop strategies to solve the problems in the Design Process.

Documentation - to record information textually or graphically in the ELTM.

Repair - to alter previously specified information in the event of a conflict between constraints or between a constraint and a proposal.

Verify - to reperform an episode to confirm whether its results are still acceptable.

Within an episode, the sequence of operators to be applied is determined by a set of heuristic rules. Also, an episode may be temporarily interrupted by a sub-episode to resolve an impasse.

The episodes, focussed on primitive goals, usually appear in groups that represent **tasks**, which can generally be described as goals of larger scope. When the designer selects a new episode to work on, it is generally related to the previous one by being concerned with the same task. A task, for example, may be 'to develop the conceptual mechanism of an operation' or to 'design a gripper for the plate'.

There are four major types of tasks in the TEA model.

Conceptual Design - to assimilate the given constraints and specify forms (at least to an abstract level) that satisfy the main functional constraints that are given.

Layout Design - to specify the conceptualized forms to decreasing levels of abstraction (assemblies and

individual parts).
Detail Design - to completely refine, dimension and document the design. Generally, this type of task starts when the Designer begins making scale drawings rather than sketches.
Catalog Selection - to select some part or assembly from a catalog. The designer gathers new information and specifies or documents some components using the catalog. Thus the operators employed by the designer are grouped to form a hierarchical set of processes.

1.2 A sample protocol section and its analysis

The following (Figure 3) is an excerpt from the protocol of a flipper-dipper design at an early stage in the design.

S : I was originally thinking of may be a hand operation of some sort, er, er, like a lever operated thing where the operator provides the mechanical...thing for it and I'm not quite sure whether the... at 720 per day, that's kind of borderline, whether or not the operator can do it. If I can stay away from a mecha...mechanized thing, motor driven or something like that it will probably be, the machine will be less expensive .. er, and it will be easier to construct, er...the...the...rate at which the... One question that I asked earlier was the toughness of the film, and that has to do with the rate at which the plate can be applied to the film; if its real critical that it will need, it will need the machine to be fairly consistent in its speed, as it brings it. if it was not real critical then a person could hand operate it and even though the speed varied a little bit it wouldn't really affect the quality of the plate, so I would be...maybe experimenting with this, to try this, and see if a person had the...ability...with some training...to...er...install the film manually.

E : Right now it's how its done. Its manually installed, manually the plates are put down.

S : Uhum...

E : ...plate contact all the way...

S : Uhum...Ok. You're trying to get away from that, for, for the fatigue...

E : Right...

S : ...for the operator. Ok. So that, you, you're steering towards a...a...mechana...a machine to do it. Ok. Aaah...the idea just came across my mind to use, kind of ferris-wheel affair which would be...reasonable to fabricate...and...but ...er...my experience with this kind of thing is that it-in...installing the film means you dip one edge in first, carry it across, like you would a microtome, picking up a...a...microtome film or something like that, so...that would mean the ferris-wheel would have to articulate as it came around, to follow the surface. That could be done with a cam... and a roller, fairly simply because you could change the cam profile fairly easily

Figure 3. A section from the protocol of Subject 5 doing the flipper dipper design.

Under the TEA model this section is treated as a task with the aim of generating a concept for the flipper-dipper machine. The breakdown of the above section of protocol within this model is presented

below (Figure 4).

Specific Design Task - Conceptual design of a Ferris Wheel type mechanism
Episodes :

EPISODE 1 assimilate - methods for powering mechanism
create [proposal 1 : hand operated mechanism]
patch [proposal changed to proposal 2 : operated mechanism with a lever]
select [constraint 1 : must process 720 plates a day]
compare[proposal 2 to constraint 1]
suspend [(borderline if the operator can do it)]
create [proposal 3 : motor operated mechanism]
select [constraint 2 : minimize costs]
select [constraint 3 : keep design simple to manufacture]
compare [proposal 3 to constraint 2,3,] (inferred operator)
suspend [no decision]

EPISODE 2 assimilate - constraints pertaining to power problem
create [proposal 4 : speed of film application]
create [constraint 4 : film cannot break during application] (inferred operator)
calculate [strategy 1 : experiment if a person can apply film per constraint 4]
suspend [strategy 1]
create [constraint 5 : reduce worker fatigue]
accept [constraint 5]

EPISODE 3 specify - Ferris-wheel type mechanism
select [proposal 3]
compare [proposal 3 to constraint 5]
refine [proposal 3 => proposal 5 : Ferris-wheel type mechanism]
select [constraint 3]
create [constraint 6 : mechanism should be easy to drive]
create [constraint 7 : mechanism should be easy to speed control]
compare [proposal 5 to constraint 6,7]
accept [proposal 5]
select [constraint 8 : plate must enter with one edge leading] (inferred operator)
calculate [constraint 9 : plate must articulate on water surface]
accept [constraint 9]
create [proposal 6 : cam and roller]
accept [proposal 6]

Figure 4. Analysis of the protocol section in Figure 3 in the TEA model framework.

Nineteen sections were selected from the different stages of five design protocols, and were analyzed and represented in terms of such primitive processes. Since a computer program was not the main objective during this analysis, many implementational details were not addressed in the TEA model. For instance, it does not include a scheme for the representation of the various design elements proposed and the pertinent constraints. A mechanism for the creation of tasks and the selection of a particular task from the tasks created is not present. Similarly, no scheme exists for the creation of episodes and the selection of a particular episode and the type of episode from the episodes created. In terms of implementation, it is not clear how the different types of episodes are distinct from each other. The kind of protocol analysis given in Figure 4 indicates what actions the designers have undertaken, but it does not give an idea of why the designer might have chosen a particular process or constraint. The most important goal of the present work is to refine or patch the TEA model in order to implement it, and thus get a little more specific about the control of reasoning and the form of required domain knowledge.

2. The underlying architecture used for the computational model

To provide a basis for building our computer model, an existing software architecture called SOAR, a general purpose problem solver developed by Laird, Rosenbloom, and Newell [7] was employed.

2.1 Description of SOAR

Problem solving in SOAR occurs as a result of the action of search control functions coded in the *processing structure* on information in the *memory structure*. The memory consists of *objects* (which are simply symbols like state22 or goal6), their *augmentations*, and a *current context*.The current context has four *slots* for the different types of objects involved in a search, namely **goal, problem space, state,** and **operator**. The object in the goal slot of the current context is the current goal ; the object in the problem space slot of the current context is the current problem space; and so on. When a slot in the context is considered for instantiation with an object-symbol, the slots to its left in the list should already be instantiated (filled with an object-symbol and not 'undecided'). Thus, a problem space is set up in response to a goal, a state functions only as a part of a problem space, and an operator is to be applied at a state. When an object in a slot is replaced, all current objects to its

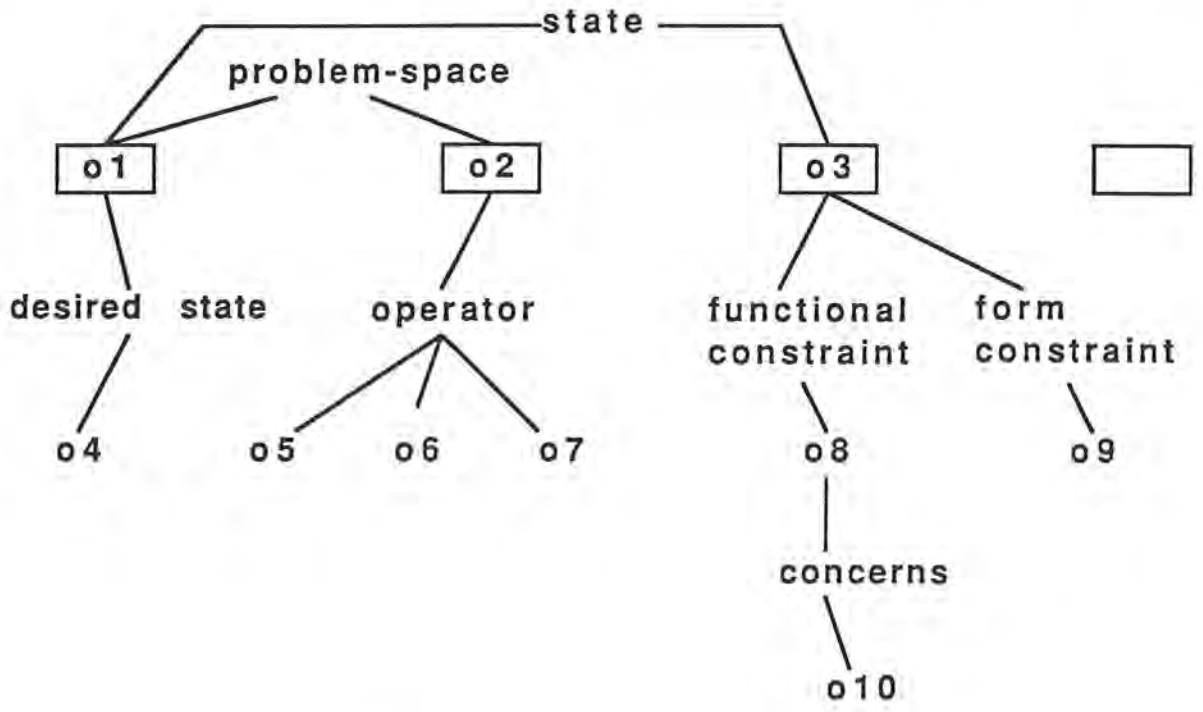


Figure 5. Memory structure of SOAR

2. The underlying architecture used for the computational model

To provide a basis for building our computer model, an existing software architecture called SOAR, a general purpose problem solver developed by Laird, Rosenbloom, and Newell [7] was employed.

2.1 Description of SOAR

Problem solving in SOAR occurs as a result of the action of search control functions coded in the *processing structure* on information in the *memory structure.* The memory consists of *objects* (which are simply symbols like state22 or goal6), their *augmentations*, and a *current context.*The current context has four *slots* for the different types of objects involved in a search, namely **goal, problem space, state,** and **operator.** The object in the goal slot of the current context is the current goal ; the object in the problem space slot of the current context is the current problem space; and so on. When a slot in the context is considered for instantiation with an object-symbol, the slots to its left in the list should already be instantiated (filled with an object-symbol and not 'undecided'). Thus, a problem space is set up in response to a goal, a state functions only as a part of a problem space, and an operator is to be applied at a state. When an object in a slot is replaced, all current objects to its

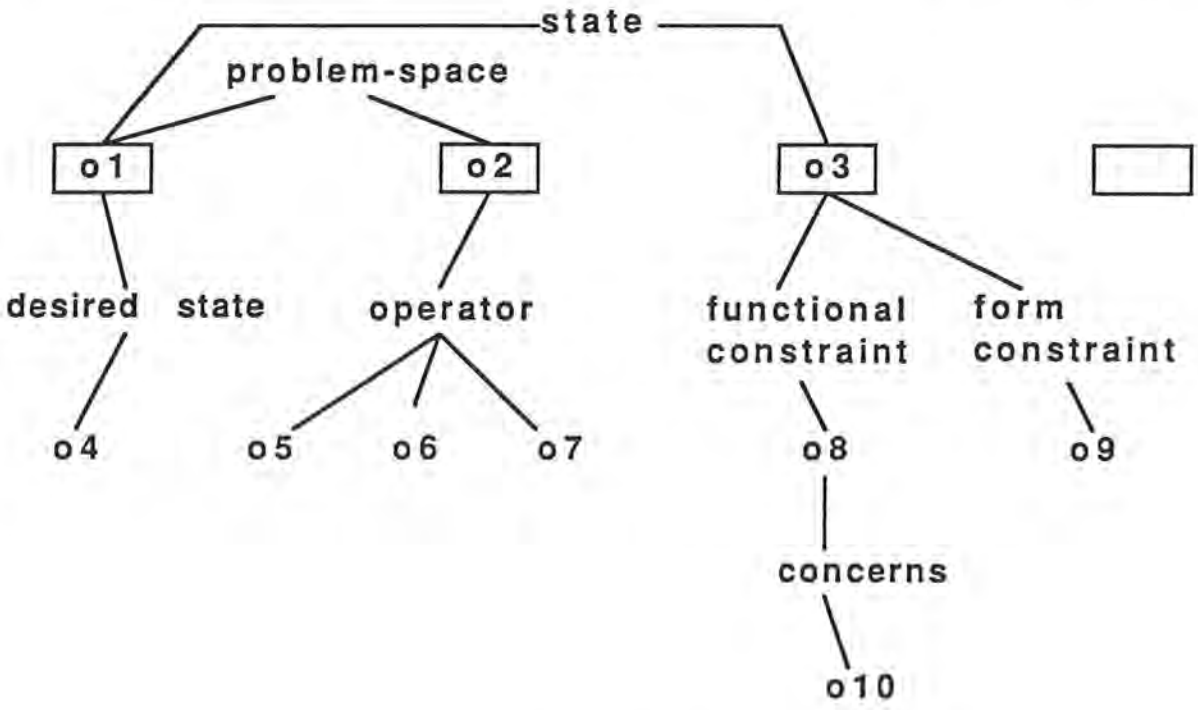


Figure 5. Memory structure of SOAR

right become undefined. Simply put, problem solving in SOAR involves instantiating these slots till an end condition (solution) is recognized.

The objects in these slots can be *augmented* with additional information. An augmentation consists of a set of objects in a labeled slot and each of the objects can be further augmented recursively. It can be thought of as a list of attribute-value pairs, where the labeled slots are the attributes and the objects are the values. The semantics of an object are derived from its set of augmentations, which may change over time, and their number is not limited by a fixed data structure.

An example of a memory structure is given in Figure 5. The symbols beginning with the letter 'o' represent objects and the symbols in the boxes represent objects of the current context; o1 is the current goal; o2 is the current problem space; o3 is the current state, and the slot for the current operator is yet undefined. In Figure 5, the goal object o1 is augmented with an attribute called 'desired state' and value o4 ; the problem space object o2 is augmented with three operators (o5, o6, and o7); state object o3 is augmented with 'functional-constraint' and 'form-constraint' attributes; the 'functional-constraint' object o8 is further augmented with an attribute named 'concerns' and the value o10.

Search-control knowledge is brought to bear in SOAR by an *Elaboration-Decision* cycle. In the elaboration phase, which always precedes the decision phase, existing objects are augmented with new objects, some of which maybe 'preferences' that contain information needed in the decision phase. Preferences are special objects with augmentations that provide information about an object that is a candidate for a slot. For example, an object will be considered for a slot only if there is a preference object with an 'acceptable' value for it and the object cannot contend for a slot if there is a preference object with a 'reject' value for it. Preferences also define a partial ordering of the objects competing for the slots in the current context. For example, there may be preferences with an 'acceptable' value for two operators to fill a slot. In this case, one operator will have an edge over the other if there is an additional preference object which indicates that that operator is 'better' than the other. In the decision phase, objects in the current context are replaced or new objects are introduced by using the knowledge accumulated during the elaboration phase. The decision procedure converts the preferences into votes and takes an action that depends on a tally of the votes for different objects.

Task-specific knowledge is brought to bear by application of operators of the current problem space when they are selected. This involves elaboration within the current goal if the operator is directly applicable. If there exists no knowledge exists of how to apply the operator, a subgoal is created to apply the operator, and the operators of the subgoal are then applied. As a result, preferences are created for a new object that represents the new state in the current goal and problem space. Other augmentations are also made that fill in the sub-structure of the new state object.

The version of SOAR used for the computer model is implemented as a forward chaining production system (similar to OPS5) and is written in Common Lisp. A production system consists of a collection of *productions* (rules) of the form -

If C_1 *and* C_2 *and ... and* C_n *then* A

where C_i are the conditions that inspect the current object context and other memory elements and A is the action to be taken if the conditions are met. The action may be either to add an augmentation, interact with external environment or to cast a vote to replace an object in the current context. All productions whose conditions are satisfied fire concurrently during the elaboration phase, and the votes of all the decision productions whose conditions are met are considered during the decision phase.

2.2 Example problem in SOAR

Suppose we have to encode the problem of sorting four numbers as a form of search in a problem space in SOAR. The problem is represented by a set of four numbered movable tiles, and one instance of it is depicted in Figure 6. There is only one operator in this set-up. This operator can transpose any two tiles that are adjacent to each other. The states of the problem space are configurations of the four numbered tiles.

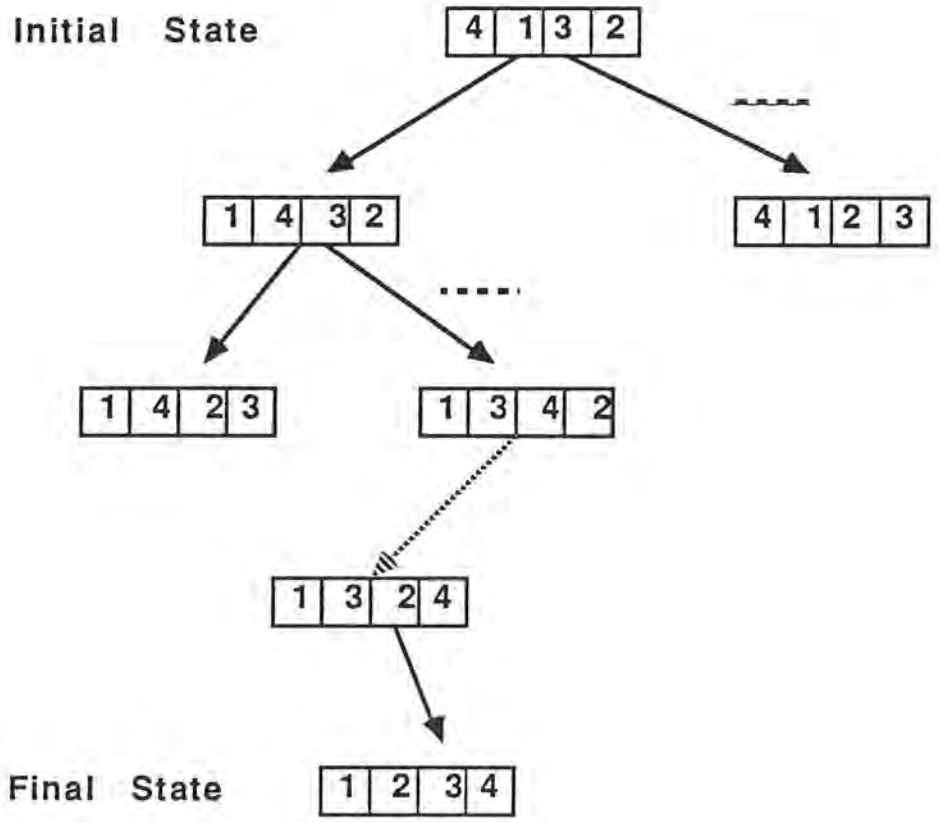


Figure 6. Sample problem in SOAR

Encoding the task in SOAR -

To select a goal and to establish the appropriate problem space and the initial state for the given task the following elaboration productions (EP) are needed.

EP1 - If the current goal is 'To-Sort'
then make an acceptable-preference for 'Sorting' as the current problem space.

EP2 - If the current goal is 'To-Sort'
then augment the goal with a desired state that contains the desired positions of tiles in cells (in sorted order).

EP3 - If the current goal is 'To-Sort' and the current problem space is 'Sorting' and there is no current state
then create an acceptable-preference for a new state, and augment it with the initial positions of tiles in cells.

EP4 - If the current problem space is 'Sorting'
then augment the current problem space with the operator(s) of 'Sorting' problem space.

EP5 - If the current problem space is 'Sorting' and there exists an operator of that problem space that can transpose two cells in the current state
then make an acceptable-preference for that operator at the operator slot.

To detect whether the goal has been achieved,

EP6 - If the current problem space is Sorting and the current state matches the desired state of the current goal in each cell
then make a reject-preference for the current goal at the goal slot.

To select and apply an operator,

EP7 - If the current problem space is Sorting and an operator is selected to act on the current state
then create an acceptable-preference for a newly created state.

EP8 - If the current problem space is Sorting and there is an acceptable-preference for a new state
then copy from the current state each cell that is unchanged by the current operator. It should be noted here that is the approach SOAR takes to solve the frame problem - the problem of describing what properties persist and what properties change as actions are performed, in a computationally rational manner.

EP9 - If the current problem space is Sorting and there is an acceptable-preference for a new state
then that state is augmented with values of the cells from the current state that are changed by the current operator (switched values of cells).

Additional elaboration productions may be needed to improve the search performance. For example, a production could prevent applying the reverse of an operator that was just applied, or evaluate new states with respect to the goal state.

These productions, along with the task-independent decision productions that translate preferences to

changes in the current context, are all that is necessary to encode this problem in SOAR.

2.3 Universal Weak Method

One underlying theme of SOAR is that the behavior of a problem-solver should vary with the amount of knowledge available for a particular task. Different weak methods should arise from the use of different task knowledge, not from the explicit selection of a method. The Universal Weak Method is an organizational framework that produces an appropriate solution method given only knowledge of the structure of the problem [6]. If there is very little or no task-specific knowledge, the Universal Weak Method suggests a default behavior which usually involves exhaustive search. When more task-specific knowledge is available to the system, the search becomes more directed and assumes the characteristics of a stronger method. This may involve moving from very weak methods such as *Generate-and-Test* to *Generate-and-Improve*, or *Means-End-Analysis*. These methods have been identified in the protocol analysis of mechanical engineer designers [7].

2.4 Difficulties and Subgoaling

During problem solving in SOAR, there may be situations where the replacement of a current object is not possible. Such situations are called *difficulties*, and there are three kinds of these :

1. No-Change : There is no change in the current context during the decision phase
2. Tie : There is a tie between competing objects for a slot.
3. All-vetoed : None of the objects have a positive vote to occupy a slot.

When a difficulty is detected in the decision phase, the current context is pushed onto a stack. A new context is created to act as the current context, with its goal slot filled with a new object representing the subgoal to resolve the difficulty. All the necessary information to diagnose and solve the subgoal in the new context is passed down from the super-context and attached to the new goal. In the present implementation of SOAR, the problem spaces that are needed by the subgoal must already exist in the system. Any production that is sensitive to the new problem space or goal contributes augmentations that, along with the universal weak method, lead to solving the problem. For example, if there is a tie between two operators in the current context (with both of them having acceptable preferences for the operator slot) a subgoal is spawned by the architecture to resolve this tie. A problem space called 'selection' is selected for the goal of this new context. For this, default productions need to be supplied to the architecture and by changing these, one can change the way SOAR resolves the tie. A new state is then selected to which the two operators are augmented. The purpose of this subgoal is to apply each of the operators to create new states and use productions that evaluate the new states in comparison to the final state. It differentiates between the operators by creating a new preference indicating one is better than the other. The subgoal is then terminated and problem solving resumes in the original context.

2.5 Reasons for choosing SOAR

The justifications for selecting SOAR as the basic architecture for the computer model are outlined below.

- SOAR views problem-solving as a heuristic search in a set of problem spaces, involving transition from one state to another by applying operators until the desired state is reached. These assumptions were also fundamental to the analysis of the protocol sessions.
- When SOAR faces a difficulty due to a 'tie' between objects to be selected or a 'no-change' situation after the selection of an object, it goes into a subgoal to resolve this impasse, resulting in nested contexts. If the problem-solving cannot progress in a subgoal it returns to the original goal where an alternate subgoal is then created. The subgoaling behavior is also observed in human designers who temporarily suspend a task, in order to solve some difficulty, and later return to it. SOAR's capability to handle a number of such subgoals also makes it appropriate for problems that have a hierarchical solution structure and fits very well into the episode scheme of the TEA model.
- A strong feature of the SOAR system is that it uses direct knowledge when available, and in the absence of such information resorts to weaker methods to solve a problem. This is essential for any model where all the knowledge for a solution may not be directly available and a default strategy should be ready at hand. Also, expert performance can be demonstrated by the model just by adding additional knowledge resulting in by-pass of several steps which are otherwise needed.
- Lastly, SOAR has a built-in learning mechanism that creates generalized "chunks" of the productions used to solve a subgoal. The performance of a novice can be modeled by using only the primitive operators and going through a fine grained series of inference steps. When such steps are done in a subgoal, the system can learn them by storing the series of inferences as a chunk of knowledge and applying the chunk later when it sees the same or generally the same situation. Thus the computer model could be very naive on its first run and with proper representations may be somewhat of an expert when it performs and learns on subsequent runs. This learning feature of SOAR has not been explored in this work, but it is an important point for future research.

3 Description of the SOAR Model

This section describes the SOAR model, developed in this research. The model is partly implemented for two sections of protocols from different problems and stages of design.

3.1 Major problem spaces in the SOAR model

A complete SOAR model would need to work in at least three major problem spaces, namely Review-specifications, Design, and Check-completeness. This can be looked upon as three different phases of design each probably requiring a different set of operators and state description to make progress in problem solving. The TEA model and the sections of protocol analyzed in figure 4 deal only with the Design phase, which is the predominant phase, but a computer model requires the other two phases for the purposes of initializing and wrapping up the design. In this paper, however, focus will be on the Design problem space and how it is implemented.

The Review-specifications phase involves gathering and organizing information about the design problem and elaborating constraints or deriving new ones, generally with the aim of 'understanding' the problem at hand. Given an explicit set of constraints (i.e., the functional specifications of the problem), the system derives new constraints or massages the given constraints using domain knowledge into more direct and applicable ones. For example, if it is given that only three of the machines are to be manufactured, then a constraint can be derived from this that no speciality manufacturing tools should be employed. During this phase all the functions that are to be done by the machine are read in and the performance pattern (the sequence of functions) of the artifact is determined. As the result of this phase, all the *given* constraints and the top level nodes of the Function tree are enumerated. This phase can be compared in some ways to the 'assimilate' episode of the TEA model, but note that the constraints gathered are for the overall problem instead of any one task in particular, and the main aim of the actions in this phase is to organize the functions to be performed. This phase is not always very explicit in the protocols, but would be needed in a computer model in order to set the stage for the Design phase to begin.

The Design phase first uses the information gathered in the previous phase along with domain knowledge to develop a concept – an abstract form that meets the functional specifications. As a result, the already existing top level nodes of the Function tree are elaborated into lower level nodes and top level nodes are contributed to the Form tree. Also during this phase, a number of design tasks are put on a task-agenda to be worked on, which in turn generates more tasks. At the completion of this phase, both of the trees will be fully expanded, all the relevant constraints will be posted, and there will be no more tasks on the task-agenda. Most of the actual design process occurs in this phase. All of the four different types of tasks proposed in the TEA model are performed, and each of these is broken down into episodes and the episodes into operators to perform the design.

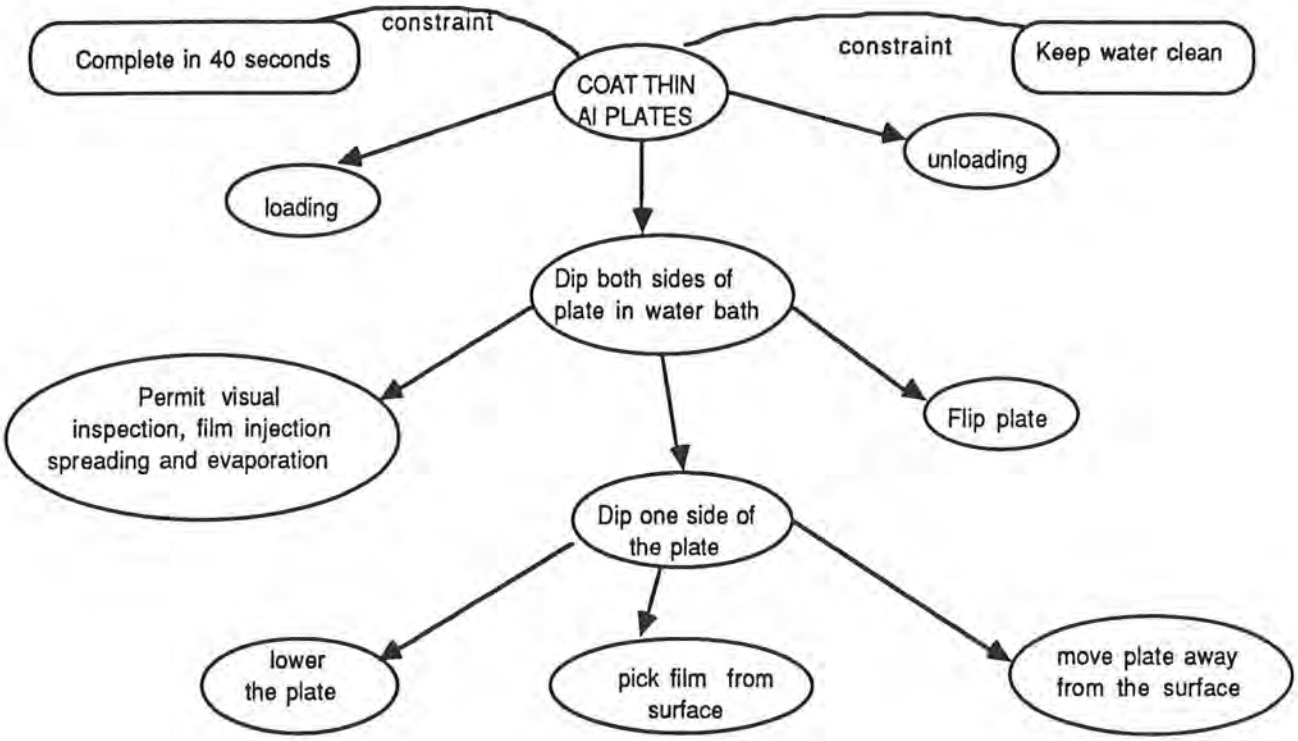
During the Check-completeness phase, the design structure (Form tree and Function tree) is examined for possible violations. If these violations cannot be patched up, more tasks are put on the task-agenda, and the design phase resumes. This phase is also not always explicit in the protocols, but there appears to be a need for such a phase to be present in a computer model. This phase can be compared to the 'verify' episode of the TEA model.

3.2 Representation scheme

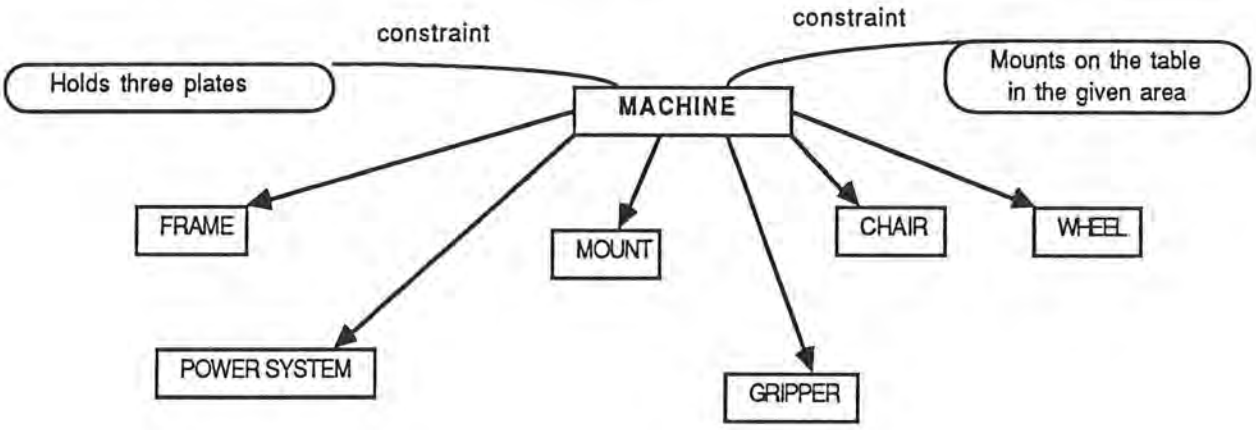
The SOAR model uses a representation scheme for a design artifact in which there are two hierarchical structures, the 'Form tree' and the 'Function tree', with the more abstract descriptions at the higher levels and the most detailed (or dimensioned) ones at the leaf nodes. Different structures were thus used for storing the information relating to form and function, mainly for efficiency. Figure 7 gives an example of this scheme used for representing the state after the conceptual design stage of the flipper-dipper problem for a designer subject. In the figure, the Function tree (with oval nodes) has the main function that is expected of the artifact as its root node. The nodes directly under this node represent a breakdown of this main function into some functions that need to be performed for achieving it. This structure could be used in suggesting new forms and to check if the designed artifact can perform all the required functions. The Form tree (with rectangular nodes) is set up in a similar manner, with the device that is being designed placed at the root node. The nodes under the root node are the components that the device is designed to be made up of, and this may only be abstract descriptions to begin with. The aim of the design process is to reduce the level of abstraction of each of these components to an extent that the artifact can be manufactured from the description. The nodes of the Form tree help in maintaining the task-agenda and in verifying that all the forms proposed are reduced to the desired level of abstraction. Nodes of the Form tree are connected to different nodes of the Function tree according to the functions that they are supposed to perform. In either tree, the nodes have links to all relevant constraints (in round-edged rectangles). At any stage of design, this structure may be only partially formed. It could be used to derive new tasks, check for consistency, completeness and correctness (constraint violations).

In the SOAR system the Formtree in figure 7, for example, would be represented as -

```
( State S4  ^Formtree FM1 )
    (Formtree FM1  ^Name Machine  ^Node  MN1  MN2  MN3  MN4  MN5  MN6  ^Constraint  Con1  Con2)
        (Node  MN1  ^Name  Frame)
        (Node  MN2  ^Name  Power-system)
        (Node  MN3  ^Name  Mount)
        (Node  MN4  ^Name  Gripper)
        (Node  MN5  ^Name  Chair)
        (Node  MN6  ^Name  Wheel)
```



Function Tree



Form Tree

Figure 7. Artifact Representation

(Constraint Con1 ^Name Hold-three-plates)

(Constraint Con2 ^Name Mount-on-given-area)

This can be depicted graphically as in figure 8 below.

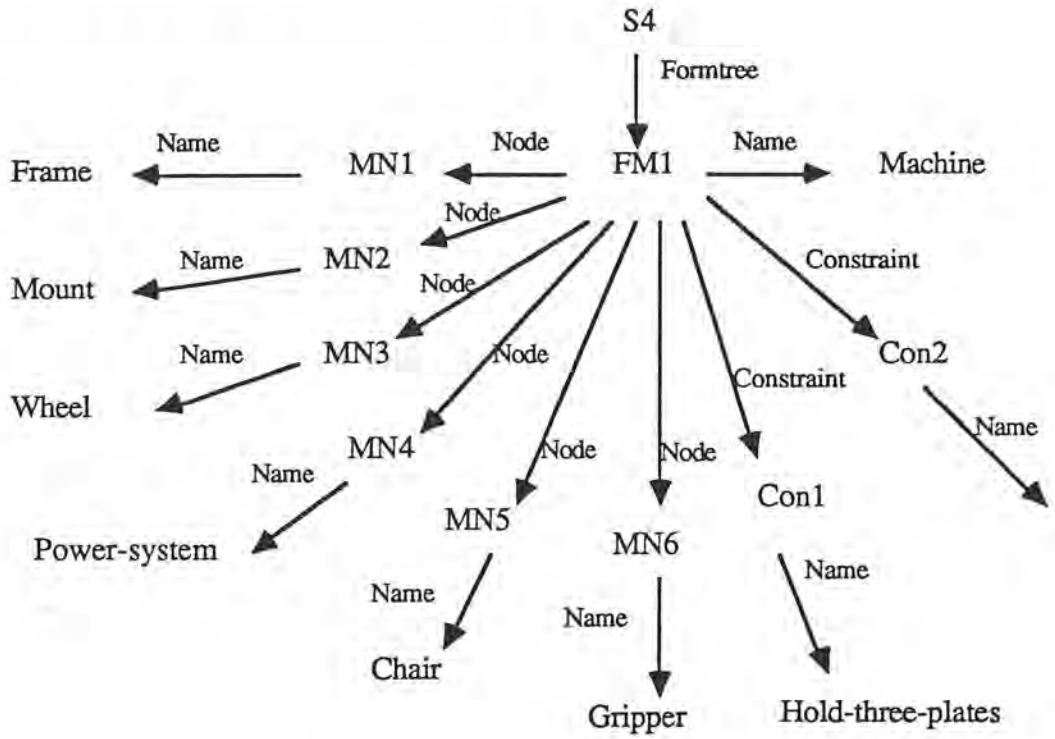


Figure 8 Graphical depiction of SOAR representation

This representation structure can also be viewed in the form of frames, as given below.

```
S4   -
        isa STATE
        node MN1
        node MN2
        node MN3
        node MN4
        node MN5
        node MN6
        constraint  Con1
        constraint  Con2

MN2 -
        isa  NODE
        name Mount
```

Thus each object can be looked upon as a frame with different attributes.

3.3 Tasks in the SOAR model

The SOAR implementation of the protocol section deals with the Design phase. It starts out with a top-level goal named 'solve-design' which has the 'design' problem space available for it. In this problem space, first the initial state is selected and given the appropriate augmentations. The initial state in the model has all the features that were present when the whole design process began, represented in the Form tree and the Function tree. This includes the specifications the designer was presented in the form of different constraints. A constraint is represented by attaching it to the particular form or function it is concerned with. For example,

State S4 ^Functiontree FN1
Functiontree FN1 ^Name Coating-film ^Constraint Con1
^Constraint Con1 ^Name Complete-in-40-secs

Note that the symbol starting with a '^' is treated as an attribute (slot). Thus, 'Name' is an attribute of the object Con1, its value (filler) is 'Complete-in-40-secs'.

At this stage, the system is at at the top level of the hierachy of processes shown earlier in figure 2. It should be noted, with respect to figure 2, that for performing processes at each lower level, the SOAR system will spawn subgoals. Thus, when it reaches the operator level, there would be a nested structure involving four goals as shown below.

```
Goal G1
Problem-space P2    (Design)
State S3
Operator O4
        Goal G5
        Problem-space P6    (Task)
        State S7
        Operator O8
                Goal G9
                Problem-space P10    (Episode)
                State S11
                Operator O12
                        Goal G13
                        Problem-space P14    (Operation)
                        State S15
                        Operator O16
```

Also note that in the present implementation all the processes (task, episode, and operator) in figure 2 are treated as SOAR operators, and if there is a dificulty in performing these, the system may create a

subgoal.

Since the subject has already made some progress in the design before reaching the start of the protocol section selected for modeling, the initial state in the model has some constraints that are introduced by the subject during the earlier design stage. Also, all the tasks that were planned before the start of the section are represented in the initial state in the task-agenda. Each item in the task-agenda has a set of attributes that help in controlling the flow of design and in maintaining consistency and completeness.

The attributes are -

- name : the name of the task (for example, detail design component A)
- form : the form(s) the task deals with
- function : the function(s) the task is concerned with
- priority : the priority class of the task

These functions and forms correspond to the functions and forms of the artifact description. New tasks are created during a proposal of a form or a function. When the form or function is proposed and accepted, it is represented in the artifact description along with its relations to other forms and functions. When the tasks are introduced, they are given a priority number from 0 to 5 according to their importance, with class 5 as the most important. This classification depends on the nature of the tasks -

* Violating - If a new proposal violates a constraint that
- was imposed by a task done before and no alternative to this new proposal is found, or
- is attached to a task yet to be done

then the task gets priority number 5.

* Inflexible - If the task has too many constraints and they cannot be relaxed, it comes under the priority class 4.

* Interacting - If a task deals with a function that interacts with the worker or a stationary object then that task is given a priority class 3.

* Costly - If the task deals with a function or component that has importance related to cost, then itemize it in class 2.

* Flexible - If the task has constraints that are relaxable or there are not too many constraints, then it comes under class 1.

* Completed- When a task is finished, it is put in class 0 (least priority).

The priority number of a task may change as the design progresses. A task at a particular level is undertaken only when all the tasks at the level above it are finished. A task may belong to more than one priority level, but the highest priority applies during the selection of tasks. It was decided to use such a priority scheme instead of using the built-in 'preference' scheme of SOAR, since the preference scheme is not suitable for a frame-work where the priority keeps changing. Since the preference

structure is not part of the state or any of the slots in a context, the old preference settings cannot be retracted while new preferences are added.

3.4 Episodes in the SOAR model

After the selection of the initial state, an operator that represents a task is selected by the 'design' problem space, according to its priority class. If the task is very simple (the system has the knowledge for its execution), then SOAR does the operation and creates a new state as a result. If the task is complex, as is often the case, SOAR generates a subgoal to deal with the task and it will return to the original goal with the changed state (if there is one) after finishing the subgoal. A problem space called 'task' is selected in the subgoal and the state from the supergoal is selected. In the episode problem space, depending on the task in hand, new episodes are suggested and are arranged in an order of importance using the preference scheme of SOAR.

3.5 Operators in the SOAR model

If the execution of the episode is straight-forward, the operator representing the episode changes the state and a different episode, the next highly preferred one, is focussed on. If no knowledge of how to directly achieve the goal of the episode exists, then SOAR creates a subgoal that deals with the execution of all the necessary steps. The problem space of this subgoal called 'episode' has a sequential set of operators that are called to perform the goal of this episode. These operators work on the superstate of the subgoal they are in.

The operators called in the 'operation' problem space are described in their order of execution as follows -

Create proposals operator -

This operator draws upon the long term memory to suggest a proposal that leads to reaching the goal of the episode. This proposal is then added as an attribute in the state and is considered for evaluations and decision in the succeeding operations before it is accepted or rejected.
Along with the posting of proposals, two other types of objects may also added to the state.
1. New constraints that are introduced as a result of the new proposal.
2. New tasks on the task-agenda.

For example, given the constraints that the value of X be between 1.5 and 2.5 and that the value of X be taken as a round number, the value of X is proposed to be 2.0.

Suppose the current goal was called 'Find-mode-of-operation', the current problem space is 'episode', the symbol for the current state is S3, and the operator is 'Create-proposals'. If there is no sufficient knowledge in the LTM to execute this operator directly, SOAR experiences a 'no-change' difficulty. Subsequently, a subgoal called 'Operator-subgoal' is created to gather the necessary

constraints and other information present in the state. A special problem space called 'operation' is selected for this subgoal and the state S3 is brought in. In this context, an operator called **'Gather-constraints'** is called upon, which creates a new state (NS3) that contains the features of the state S3 and all the constraints relevant to the current episode that are collected. The 'Create-proposals' operator is then applied to the new state (NS3) to generate a proposal. The SOAR run for this sequence would look like the following -

```
Goal  G1 : Find-mode-of-operation
Problem-space P2 : Episode
State S3
Operator O4 : Create-proposals
        ==> Goal G5  : Operator-subgoal   ( Operator no-change)
             Problem-space P6: Operation
             State S3
             Operator O7 : Gather-constraints
             State NS3
             Operator O4 : Create-proposals
State S8
```

If more than one proposal is suggested by the Create-proposals operator, the resultant states (say, S8a and S8b) will compete with each other. This represents a 'tie' difficulty in SOAR, and an appropriate subgoal (G9), called 'Operator-subgoal', is generated to resolve the situation. In this subgoal, a problem space (P10) called 'Selection' is created. A new state (S11) is then generated for this subgoal which has an evaluation slot (E12 and E13) for each of the competing states (S8a and S8b). As soon as the evaluation slots in this state is filled with some values (numeric or symbolic) that convey the superiority of one state the other, 'Operator-subgoal' is terminated and a 'preference' is made for the winning state. To help fill the evaluation slots (E12 and E13) operators named 'Evaluate-object' (O14 and O15) are invoked in the 'Selection' problem space, for evaluating each of the competing states (S8a and S8b). Let us follow the execution of an 'Evaluate-object' operator (O14). This operator usually cannot be executed directly, therefore a subgoal (G16), called 'Operator-subgoal', is generated.

In the subgoal (G16), a special problem space (P15) called 'Comparison' is selected. One of the competing states (say, S8a) is brought in, and an operator called 'Evaluate' (O18) is selected. This operator uses any knowledge available to evaluate the current state (S8a) and fills the evaluation slot (say, E12) of the state in the 'Selection' problem space with a value.

The subgoal (G16) is then terminated, 'Evaluate-object' operator (O14) is rejected, and the other 'Evaluate-object' operator (O15), for evaluating the other state (S8b) is then executed in similar manner. The SOAR trace of the execution of the above events is given below.

```
Goal  G1 : Find-mode-of-operation
Problem-space P2 : Episode
State S3
Operator O4 : Create-proposals
        ==> Goal G9 : Operator-subgoal   (State tie)
            Problem-space P10 : Selection
            State S11                              {contains E12 and E13}
            Operator O14 :  Evaluate-object (S8a)
                ==> Goal G16 : Operator-subgoal   (Operator no-change)
                    Problem-space P15 : Comparison
                    State S8a
                    Operator O18 : Evaluate
            Operator O15 : Evaluate-object (S8b)
                ==> Goal G19 : Operator-subgoal   (Operator no-change)
                    Problem-space P15 : Comparison
                    State S8b
                    Operator O20 : Evaluate
State S8a
```

This illustrates how the subgoaling scheme of SOAR is utilized for some of the problem-solving involved in the protocol section.

Evaluate Proposals operator -

In this step the proposal is evaluated using domain knowledge. If no knowledge exists to directly evaluate the proposal, and if more information needs to be collected then a subgoal is set up similar to the one for 'Create Proposals' operator, and an operator called 'Gather-factors'is executed in this subgoal. The 'Gather-factors' operator collects all the necessary aspects to be considered for the evaluation of the proposal, and puts them as a special attribute of the state. It attaches a 'weight ' to each of these factors, establishing the relative importance of each evaluation factor in the decision about the proposal. The 'Evaluate Proposals' operator is executed after all the factors are gathered.

During the execution of the 'Evaluate Proposals' operator each factor contributes a value (called 'evalue'). This is the product of the goodness-value, that is, how good the proposal is according to this factor (on a scale of -1 to 1), and the 'weight' of that factor. The proposal thus gets a set of such values attached to it at the end of this operation. For example,

1. If a machine component is involved and
   the component has been known to be made before and
   Fabrication is a factor under consideration then
   evalue = 0.8 * 'cweight'$_{Fabrication}$

2. If a drive is involved and
control is automatic and
Safety-if-interrupted is a factor under consideration then
evalue = -0.2 * 'cweight'$_{Safety\text{-}if\text{-}interrupted}$

All those factors that did not contribute an 'evalue' (due to lack of knowledge to evaluate) and the ones that contributed a negative value are stored in the state in a separate attribute called 'consider-factors'. These factors will be considered for a patch-up of the proposal.

After all the possible 'evalue's are calculated, the system generates a subgoal that selects a problem space called 'patch' and the state of its supergoal. In the 'patch' problem space, an operator is called for each of the factors in the 'consider-factors' attribute of the state. For a factor that has contributed negatively, a patch of the proposal is made if possible using the knowledge available. If a patch is performed, the 'Evaluate proposal' operator is called within this subgoal and the proposal is reevaluated with the patch. For the factors that did not result in any 'evalue', additional information is asked for from outside that will help in evaluating the proposal with respect to that factor. Also performed are refinements of the proposals, in order to raise its 'evalue'. After all the possible patches are done, the subgoal is terminated and SOAR returns to the 'episode' problem space.

Decide operator -

Based on the 'evalue's of a proposal as a result of evaluations done in the previous operation, a decision is made (based on a trivial calculation at present) to either accept or reject :
- accept the proposal and create a new state with the changes proposed along with the constraints and agenda items. If more than one satisfactory proposal exists, then the one with highest evaluation score is selected.
- reject the proposal and return to the state that existed before considering this proposal. The proposal will not exist in the system anymore.

In either case the episode subgoal is terminated and the system returns to the 'task' problem space to shift its focus of attention on other episodes. If no more episodes that belong to the present task exist, the system terminates the subgoal for episodes and returns to the 'task' problem space and continues to work on a different task.

3.6 Protocol analysis in the SOAR model

The model is implemented for two protocol sections in the Design problem space. One of them deals with the conceptual design of the 'flipper-dipper' machine. A part of this section was presented in Figure 3 in Section 1. This section presents the analysis of that part in terms of the SOAR model.

The terms used in the presentation of the analysis are explained first -
CP - Create Proposals

GPC - Gather Proposal Constraints
EP - Evaluate Proposals
GEF - Gather Evaluation Factors
DP - Decide Proposals
{ } - comments

Please note that the evaluation results such as good, fairly good, are represented in the computer program as numbers. Also, operations that were introduced for this model, but not explicitly verbalized by the subject are denoted as 'assumed' in their comments section.

The following is the analysis of the part of a protocol session that was given in Figure 3. The state at the beginning of this section is represented as

```
(StateS4  ^Formtree M5  ^Functree N6  ^Task-agenda T7)
        (Formtree M5  ^Name Machine)
        (Functree N6  ^Name Coating-film  ^Object Plate  ^Node  C11 C12 C13  ^Constraints C14 C15)
                (Constraints C11  ^Name Keep-water-clean  ^concerns D16 D17)
                (Constraints C11  ^Name Complete-in-40-secs)
        (Task-agenda T7  ^Name Conceptual-design  ^Priority 1)
        (Node  C11  ^Name Loading)
        (Node  C12  ^Name Dipping  ^Description Both-sides  ^Node  D16 D17 D18)
                (Node  D16  ^Name Dippiing  ^Description One-side  ^Node  D19 D20 D21)
                        (Node  D19  ^Name Lowering)
                         (Node  D20  ^Name Lifting  ^Object Film  ^Description From-water-surface)
                         (Node  D21  ^Name Moving  ^Description  Away-from-water-surface)
                (Node  D17  ^Name Stopping  ^Subject Worker  ^Description Let-visual-inspection
                                                Let-film-injection Let-Spreading  Let-Evaporation)
                (Node  D18  ^Name Flipping)
```

Task - Conceptual design of flipper dipper

Episode 1 - determine mode of operation

CP - Hand operated machine {proposal 1}
Power operated machine {proposal 2}

{ A tie between two states occurs. This is resolved by the evaluation of these states in the 'Compare' problem-space)

EP -

GEF - (factor 1) - Volume of 720 plates per day {since hand operation is involved}
(factor 2) - cost {since power operation is involved}
(factor 3) - ease to construct {since power operation is involved}

(factor 4) - constistency of speed {since hand operation and power operation are being compared}

EP - (factor 1) - proposal 1 not good
proposal 2 very good
(factor 2) - proposal 1 good
proposal 2 not very good
(factor 3) - proposal 1 good
proposal 2 not very good
(factor 4) - proposal 1 cannot evaluate
{subgoal to find whether constistency of speed is important to hand operation the subject gathers this from the experimenter}
proposal 1 good
proposal 2 good

DP - Accept (proposal 2) { the information about the mode of operation is added to the the top node of the Function tree }

Episode 2 - Create abstract form

CP -

GPC - loading (plate) , dipping (plate), unloading (plate) {the main functional specifications}

CP - Ferris wheel mechanism (proposal 3) {have Gripper, Chair, Frame, Power system, so on. These will be added through the 'Node ' attribute to the Form tree in the state when this proposal is accepted}

EP -

GEF - (factor 1) - ease to construct
(factor 2) - ease to drive
(factor 3) - dipping function ... {to be performed like a microtome}

EP - (factor 1) - good
(factor 2) - good
(factor 3) - not good {articulation needed}

PATCH -
use cam and roller {subgoal for patching}

The protocol section and its implementation in SOAR goes further with various evaluations of proposal 3 and ultimately accepting it into the state at the end of Episode 2. This concludes the discussion of the representation scheme, control structure, and the primitive operators used in the computational model.

4. Discussion

The scheme of processes mentioned above was implemented with the objective of meeting some basic requirements of a computer model.

4.1 Criteria for computer models

A computer model should be general enough to accommodate both the systematic and the opportunistic modes of design. SOAR accommodates exploration of all the alternatives present and can search through the space of solutions until an evaluation can be made for each alternative. An opportunistic component is attached to the present model by delaying the exploration of alternatives to a proposal (posting it as a task of low priority) until the present one is found to be totally unacceptable. The model uses a systematic layering of tasks according to their importance and uses this to decide which task to work on. However, within a layer, the task selected is related to the previous task in terms of the form, function, or constraints it deals with. At the beginning of a task, episodes are created depending on the task being worked on. While working on the episodes, new episodes are created on an ad-hoc basis depending on the importance of a proposal to the stage of design. For example, during the task of the conceptual design of the flipper-dipper mentioned above, a proposal was made to include a drive. Immediately an episode is created to elaborate the drive mechanism since it has bearing on the conceptual design of the machine.

Within an episode, the model goes through all the major primitive operators (create, evaluate, decide) systematically, but uses other operations such as refining and patching when needed.

A computer model should be able to encompass the design strategies of different designers and several design problems, so it can be used in the attempt to explain a design process and thus understand more about it. Additional features can then be added to the model to make it more robust or specialized. This would be more of a continuous effort of trial and error with empirical and analytical methods rather than building the right model at one attempt. This feature of the model is demonstrated to some extent by using the model for representing two protocol sessions taken from two different designers working on two different problems. Appendix 1 presents the second protocol session and its and analysis on terms of the SOAR model.

A computational model also should be as elementary as possible so that it can model a naive designer and also be able to take jumps of inference to emulate experts. The computer model discussed above can behave in either of these ways depending on the knowledge available.

4.2 Comparison with the TEA model

The TEA model provides a good basis for building a computational model of design. The computer model proposed in this paper, however, differs from the TEA model proposed earlier in

many subtle ways, which are discussed below.

The four different tasks proposed in the TEA model are put under one phase of design in the present model, mainly because the distinction between these phases (conceptual, layout and detail design phases) are quite abstract and ill-defined. Under the Design phase, the model deals with these phases differently by taking advantage of the representation structure used. For example, if a function exists in the structure that has no form attached to it, a task (conceptual design) would have been created to deal with this. Similarly, if a form node is not elaborated to the level of dimensions, another task (detailed design) would be present to deal with this.

The distinction between types of episode is not made in the present model as it was done in the TEA model. A description of how the different kinds of episodes used in the TEA model are depicted in terms of the computer model is given here.

Assimilation - There is no separate episode for this purpose but this is inherent in the design of the model. All episodes have primitive operators to gather the relevant constraints for a proposal, for its refinement, or for its evaluation.

Repair - Although the current implementation does not demonstrate this, when a constraint violation is noticed and a local patch-up is not possible, the mechanism of SOAR allows backing up to the point where there was no violation and proposing a new alternative. Again, there is no special episode for this purpose, and it is implicit in the architecture used.

Specification - All episodes can be considered to be basically of this type. Within this episode different problem spaces may be used for different ways of specifying information.

Documentation - When documentation is done for problem solving, it is handled by a special problem space within a normal episode.

Plan - Much of the actions in this episode are selection of strategies and can be handled by having default productions in SOAR that allow consideration of a different problem space to attack a design problem. However, planning in a truer sense should be done by having a problem space for creating these problem spaces themselves. Such a feature is not existent in SOAR and is being explored by its developers.

Verification - Local verifications (within a task) are done by the evaluate operator within an episode. Reperforming of tasks to check global consistency on the arrival of additional information is handled in the Completeness-check phase and so is treated as a task by itself. The present implementation does not however, demonstrate this action.

The operators proposed in TEA model are also used in a different form in the computer model. The ten operators described earlier in the paper and their relation to the present model are described.

Create - This operator is equivalent to the Create proposal operator.

Select - The selection of old information is done by the Gather operators.

Calculate, Simulate - There are no explicit operators to do these operations.

Compare - Most of the functions of this operator, like checking a proposal with constraints, is done by

the Evaluate operator.

Accept and Reject - No separate operators for these are present, but these occur as a result of the Evaluate operator.

Suspend - When a proposal is rejected mainly due to lack of evaluation knowledge, a task is put on the agenda to work on it later.

Patch - This is done in a special problem space within the function of the Evaluate operator

Refine - This operation is performed within the function of the Evaluate operator.

In terms of a computational model, the TEA model was incomplete in many ways -

It did not include a model of control. The SOAR model exhibits control by

1. subgoaling on impasse, which models all forms of subgoaling
2. a stereotypic operator sequence within an episode, which models the gethering of constraints and their use
3. a task priority structure, which determines which task will be worked on next
4. evaluation mechanism, which makes the decision making process precise

It did not give any specific details of the representation of the design state and design constraints. The SOAR model refines the representation of state to include functiontree, the formtree, and the various attributes of forms and functions.

It did not give any specific details of the representation of design knowledge. The SOAR model used SOAR (elaboration) productions to represent design knowledge.

4.3 Some problems encountered

The SOAR architecture provided convenient mechanisms for implementing some aspects of the TEA model including a) subgoaling via impasses and b) the use of "unimplemented" operators to cause subgoals and thus dictate the stereotypic structure of episode. SOAR productions provided an acceptable representation of design knowledge.

The SOAR architecture did not provide built-in mechanisms for handling some aspects of the model. However, these aspects, like the task priority system and the evaluation scoring system, were fairly easy to represent. The SOAR system is not very well developed for interaction with the outside world. External LISP functions can take from and return to an attribute only numerical values and not symbols. Objects once introduced in SOAR cannot be retracted by the action of a production (the garbage collector does it after a decision cycle if the object has been rejected by the decision). As a result of this at each small change of state, all the unchanged features of the state have to be copied over. Also, due to this, the hierarchical representation structures have to kept fairly flat for efficiency purposes. In the present implementation this is not much of a problem, but a better scheme is required for modeling the entire design process.

A major problem in building the computational model was the incompleteness of the protocols.

One has to make several assumptions of how the designer may have arrived at some decisions, and this may sometimes have serious implications in building a correct model for design. The designers do not verbalize the justifications for most of the 'jumps' in reasoning, many of which they probably do not know themselves. One very glaring and obvious conclusion of this work is that protocol analysis should be treated, especially for the purposes of building a computational model, merely as an exploratory tool. If one were to use protocol study for building a model, the protocols should be taken with the idea of getting the control knowledge of the designer. More emphasis should be given on why the designers performed a certain action rather than just what the designers do. This is a particularly difficult task since the designers may go into phases of introspection which may adversely affect the process of protocol data gathering. This could be done to some extent by replaying the videotape after each brief session and asking of the designers the rationalizations for their actions.

4.4 Future directions

The model proposed in this paper could be implemented completely by representing a larger portion of the protocol, one where the selection of tasks, repair, and other features that could not be demonstrated here will be encountered. The model should be verified by representing sections of protocol from various stages of design and involving different design problems.

An important feature of SOAR, the learning mechanism, could be studied and exploited in terms of the design process. Once a reasonably solid computer model is built, comparison studies of experts and novices along with the model could prove to be interesting.

A special problem space for creating or selecting problem spaces could be a way to introduce the automated planning element into the model. Work being done in this area should be studied and exploited.

If the reasons for a proposal can also be included in the nodes of the representation structure built in this model, it can be used as a record of execution and can be extended to provide an explanation or report of design.

References

1. David G. Ullman, Thomas G. Dietterich, Mechanical Design Methodology : Implications on Future Developments of Computer-Aided Design and Knowledge-Based Systems, *Engineering with Computers*, 2, 21-29 (1987).

2. David G. Ullman, Thomas G. Dietterich, Larry A. Stauffer, A Model of the Mechanical Engineering Design Proces Based on Empirical Data : A Summary, to be presented in AIENG-88 Conference, Palo Alto, CA, Aug 14, 1988.

3. David G. Ullman, Larry A. Stauffer, Thomas G. Dietterich,Toward Expert CAD, *Computers in Mechanical Engineering*, Vol. 6, No. 3, p56-70 (Nov/Dec 1987).

4. Larry A. Stauffer, An Empirical Study on the Process of Mechanical Engineering Design, Ph.D. Thesis, Oregon State University, Corvallis, OR 1987.

5. Allen Newell, Herbert A. Simon (1972), *Human Problem Solving*, Prentice Hall, Englewood Cliffs, NJ.

6. John Laird, Paul Rosenbloom, Allen Newell (1986), *Universal Subgoaling and Chunking, The Automatic Generation and Learning of Goal Hierarchies*, Kluwer Academic Publishers.

7. Fundamental Processes of Mechanical Designers Based on Empirical Data, DPRG -5, 1987, Design Process Research Group, Oregon State University.

Appendix 1

This appendix presents a section of the protocol and its analysis in terms of the SOAR model for the 'battery contacts' design problem. The battery contacts design problem can be briefly described as -

Design a plastic envelope (dimensions provided) and the electrical contacts to accept the three batteries to power the time clock in a new computer. The batteries (detailed dimensions provided) must be connected in series and to an adjacent printed circuit board. The external dimensions of the envelope are provided as are needed contact pressures. The volume is 50,000 units/month for three years and the assembly will use a robot.

The following is the excerpt from the protocol ('*'s stand for pauses) -

So for dimensions of those new contacts, starting with the two that look -- it'd probably be better if they were angled, there's more space that way, without hitting the supports which * would be somewhere near there * with a wire to connect those. The diameter of that would need to be – the top part of the diameter of the battery is .228 which is the smallest part, so it would have to be smaller than that diameter. And, ** pick a number from .228, smaller than that, say, look at .2 diameter and that would give you forty thousandths clearance all around. That's probably not important, but we are going for a number. * The distance from the center there to there will have to be the diameter of the battery plus a little clearance, to have enough room to solder the wire on. * Or again, you could just * connect those metal to metal and just have it all one part. ** * This should be a different height than the rest of the material if this is gonna be the contact. So it could be all made of one part, instead of soldering it. Okay, say if we made that all one part, and connect that across there and this nickel plate * the two circles and the increased height that you'd need. And the same part we are going to use on the bottom and top envelope O9. And that part will look like -- let's see, without the nickel plating *** would look similar to this and the dimension from there to there * is given 2.5 to there * plus or minus four thousandths. The dimension from there to there, the maximum would be half the diameter of the battery plus .047 which is .228, plus .047 or .245. * The distance, no we probably want to set distances. * That radius, .02, half of that is the diameter, which is .010 radius. So if we made this whole part the same thickness, * .020, and this nickel plate, just those two areas, and the rest of the material would be made out of, say copper alloy, which is a good electrical conductor. * A beryllium copper is commonly used for springs, but you don't really need a spring force on these so it's not really a question, if you just use copper for its conducting properties. One thing I'd have to check is if we can nickel plate to a copper alloy, that I can use. So, the thickness of this, if you look at it from the side, would be pretty narrow. We don't have much room, anyway. We have 025 and 22 1/2 for the thickness of the plastic, plus the contact. So if you made that *** I have that 020 thickness there for the wall, probably have to be smaller than that, considering we'll need to make * that would leave that 025 left for the contacts. But this was made out of a thin sheet of copper. I'll have to check sizes to see what kind of stock sizes, so I'll guess that that would come between

E : What sizes are you thinking of?

S : It would have to be around .005 thousandths. I don't know if that's a standard size or not.

E : Take it as if it were of a standard size.

The progress of design fromthe start of the protocol section given above to its end is presented in the next page in terms of the SOAR model and the TEA model.

The abbreviations used are -

CP - Create Proposals

GPC - Gather Proposals Constraints

EP - Evaluate Proposals

GEF - Gather Evaluation Factors

DP - Decide Proposals

The state at the beginning of this section of the protocol is represented in the SOAR model as

```
(State S4  ^Formtree M5  ^Functiontree N6  ^Task-agenda T6 ^Available A1)
    (Formtree M5  ^Name Battery-contact ^Node MN1 MN2 ^Constraint Con0)
        (Node MN1 ^Name Contact ^Node CN1 CN2 CN3 ^Constraint Con1 Con2)
            (Node CN1 ^Name Left-contact ^Constraint Con3)
                (Constraint ^Con3 ^Name Ends-on-battery&pcb)
            (Node CN2 ^Name Middle-contact ^Constraint Con4)
                (Constraint ^Con4 ^Name Ends-on-adjacent-batteries)
            (Node CN3 ^Name Right-contact ^Constraint Con5)
                (Constraint ^Con5^Name Ends-on-battery&pcb)
            (Constraint ^Con1 ^Interfaces MN2)
            (Constraint ^Con2 ^Name shortest-length-possible)
        (Node MN2 ^Name Envelope ^Node EN1 EN2)
                (Node EN1 ^Name Upper-half)
                (Node EN1 ^Name Lower-half)
        (Constraint Con0 ^Name prevent-short-out)
    (Functiontree N6 ^Node NN1 NN2 NN3)
        (Node NN1 ^Name Covering)
        (Node NN2 ^Name Holding)
        (Node NN3 ^Name Electric-connection)
    (Available A1 ^Name Battery ^Node BN1)
        (Node BN1 ^Surface-diameter 0.02)
```

SOAR model
(indentations indicate creation of subgoal)

Task - Dimensions of middle contact

Episode 1 - Shape of middle contact

CP - (need to gather the relevant constraints)
GPC (the constraints that deal with the middle contact are gathered)
- one end on one battery and the other end on the other battery
- take shortest distance between the two batteries
CP - contact shape is horizontal.

EP
GEF - interference with position of side wall of the envelope

EP - bad

(create a new problem space to do a repair on the proposal)

PATCH

- sides of contact angled back

DP - accept

(Add the information proposed to the new state - that is, the shape of contact involves a curve around the side of the wall)

Episode 2 - Diameter of contact surface

CP -

GPC- prevent short-out

- diameter of contact surface should be less than diameter of battery surface

TEA model

Task - Dimensions of middle contact

Episode 1 - Documentation - Overall shape of middle contact

select [proposal1 : sides of contact are horizontal]

draw (proposal1)

select [constraint1 : relative location of support wall interferes]

simulate [proposal1 to constraint1]

compare [proposal1 to constraint1]

Episode 2 - Repair - angle of contact's sides

patch [proposal1 => proposal2 : sides of contact are angled back]

accept [proposal2]

Episode 3 - Specification - Diameter of contact surface

select [proposal3 : diameter of contacting surfaces in drawing]

draw (reference for dimension)

Episode 4 - Assimilation - limitations on diameter

select [proposal4 : diameter of battery is .28]

- diameter of battery surface is 0.28

- get a round number

CP - diameter is 0.20 inches

EP -

GEF- clearance must be 0.040

EP- good

DP- accept

(Add the information proposed to the new state - that is, the diameter of contact surface is 0.02 inches)

Episode 3 - Size of Contact

CP - Length of the contacts is diameter of battery plus clearance for soldering wire
- Solder wire to contact

EP -
GEF - minimize number of parts [assumed]

EP - bad

PATCH
- Have the two contacts and wire as one metal piece
- use same part on top and bottom

EP -

GEF - height of contact surface greater than rest of contact

create [constraint2 : contact canot short-out]!

calculate [constraint3 : diameter of contacting surface must be < .28]

accept [constraint3]

select [constraint4 : round off number to a "nice" number]!

create [constraint5 : clearance between contact diameter and battery diameter must be 0.040]

compare [proposal3 to constraint3 - 5]

refine [proposal3 => proposal5 : diameter is 0.2 inches]

accept [proposal5]

draw [proposal5]

Episode 5 - Specification - length of side

select [proposal6 : length of side in drawing]
draw (reference for dimension)
select [proposal7 : diameter of battery]
draw (proposal7)
create [constraint6 : need clearance for soldering wire]
simulate [proposal6 to proposal7 subject to constraint6]
suspend [make contact out of one piece]

Episode 6 - Specification - contact out of one piece

select [constraint7 : minimize number of parts]!
select [proposal2]
compare [proposal2 to constraint6,7]
patch [proposal2 => proposal8 : contact is made from one piece]
accept [proposal8]

Episode 7 - Verification - Contact out of one piece

create [constraint8 : height of contacting surface must be greater

EP - good
DP - accept

Episode 4 - Dimension between contact points

CP -
GPC - distance between batteries is 0.530 +/- 0.004
- contacting points should contact centers of batteries
CP - distance between contact points is 0.53 +/- 0.004
EP -
GEF -
DP - accept

Episode 5 - Height of contact

CP -

GPC - diameter of battery is 0.228
- clearance from battery to back wall of envelope is 0.017
- height of battery should be less than 0.228 + 0.017

CP - height of contact is less than 0.245
EP -
GEF -
DP - accept

Episode 6 - Width of contact

CP -
GPC- diameter of battery is 0.2
- width of contact can be same as diameter of battery
CP - width of contact is 0.2
EP -
GEF -
DP - accept

than rest of contact]
compare [proposal8 to constraint8]
accept [proposal8]
draw (proposal8)

Episode 8 - Specification - Dimension between contact points

select [proposal9 : distance between contact points in drawing]
select [proposal10 : center distance between batteries is .530 +/- .004]
calculate [constraint12 : contacting points should centers of batteries]
compare [proposal9 to proposal10 subject to constraint12]
refine [proposal9 => proposal11 : distance between contact points is .530 +/- .004]
accept [proposal11]

Episode 10 - Specification - Height of contact

select [proposal12 : height of contact in drawing]
draw (reference lines)
Episode 11 - Assimilation - constraints needed for height
create [proposal13 : half diameter of battery is 0.228]
select [proposal14 : clearance from battery to back wall of envelope is 0.017]
calculate [constraint13 : height can be less than proposal13 plus proposal14]
accept [constraint13]
compare [proposal12 to constraint13]
refine [proposal12 => proposal15 : height of contact is .245]
accept [proposal15]
draw (proposal15)

Episode 13 - Specification - Width of contact

select [proposal5]
draw (proposal3)
select [proposal17 : width of contact in drawing]
create [constraint15 : diameter and width can be same size]!
compare [proposal17 to proposal5 subject to constraint15]
refine [proposal17 => proposal18 : width of contact is .020]
accept [proposal18]

Episode 7 - Materials for contact

CP -
GPC- good conductance

CP - proposal 1 - contact made of nickel and rest of the material from copper alloy
- proposal 2 - material used could be beryllium copper

COMPARE -

GEF (factor 1) - spring force not needed
(factor 2) - adherence (can nickel adhere to copper)
EP - (factor 1) - proposal 1 good
- proposal 2 bad
- (factor 2) - suspend { not enough information, so create a task to check this later}

DP - accept proposal 1

Episode 8 - Thickness of contact

CP -

GPC - thickness of envelope plus contact is 0.0225
- thickness of envelope is 0.02
- thickness of contact less than or around 0.0025
- take a standard size

CP - thickness of contact is 0.005

DP - accept

Episode 14 - Specification - Materials for contact

select [proposal5]
select [proposal19 : proposal5 is made of nickel]
select [proposal20 : material of contact except for plated surface is unknown]
select [constraint16 : conductance of contacts must be good]
create [constraint17 : copper is a good conductor]
compare [proposal20 to constraint17]
refine [proposal20 => proposal21 : material of contact except for plated surfaceis copper]
create [proposal22 : material of contact is beryllium copper]
create [constraint18 : beryllium copper is commonly used for springs]
calculate [constraint19 : contact does not need spring force]
reject [proposal22]
create [constraint20 : plating must adhere to contact material]
calculate [strategy2 : check later if nickel will adhere to copper]
accept [strategy2]
accept [propoasal21]

Episode 15 - Specification - thickness of contact

select [proposal23 : thickness of contact in drawing]
draw (proposal23)
select [constraint21 : not much space available for contact]
Episode 16 - Assimilation - Information on available space
select [proposal24 : thickness of envelope & contact is .0225]
select [proposal25 : thickness of the envelope is .020]
calculate [constraint22 : thickness of contact must be < .0025]
accept [constraint22]
create [constraint23 : use only standard-size materials]
create [constraint24 : a standard size is 0.005]
compare [proposal23 to constraint22 - 24]
refine [proposal23 => proposal26 : contact thickness is .005]

accept [proposal26]

The state at the end of this section of the protocol is represented in the SOAR model as

```
(State S4  ^Formtree M5  ^Functiontree N6  ^Task-agenda T6 ^Available A1)
          (Formtree M5 ^Name Battery-contact ^Node MN1 MN2 ^Constraint Con0)
              (Node MN1 ^Name Contact  ^Node CN1 CN2 CN3 ^Dimensions  DN1  DN2  DN3  DN4
                    ^Material   MT1 MT2  ^Constraint Con1 Con2)
                 (Node CN1 ^Name Left-contact ^Constraint Con3)
                    (Constraint ^Con3 ^Name Ends-on-battery&pcb)
                 (Node CN2 ^Name Middle-contact ^Shape SN21  ^Constraint Con4   ^Node  MC1 MC2 MC3
                    ^Dimensons DN5)
                    (DImensions DN5  ^Distance-between-pts  0.53 +/- 0.004)
                    (Node MC1 ^Name Left-piece )
                    (Node MC2 ^Name Wire   ^Description  Soldered  ^Interfaces MC1 MC2)
                    (Node MC3 ^Name Right-piece)
                    (Constraint ^Con4 ^Name Ends-on-adjacent-batteries)
                    (Shape  SN21   ^Name Sides-are-angled-back)
                 (Node CN3 ^Name Right-contact ^Constraint Con5)
                    (Constraint  Con5 ^Name Ends-on-battery&pcb)
                 (Dimensions DN1  ^Length-of-contact-surface  0.02)
                 (Dimensions DN2   ^height-of-contact 0.245)
                 (Dimensions DN3   ^width-of-contact 0.2)
                 (Dimensions DN4   ^Thickness-of-contact 0.005)
                 (Material  MT1  ^Name Nickel)
                 (Material  MT2  ^Name  Copper-alloy)
                 (Constraint ^Con1 ^Interfaces MN2)
                 (Constraint ^Con2 ^Name shortest-length-possible)
              (Node MN2 ^Name Envelope ^Node EN1 EN2)
                 (Node EN1 ^Name Upper-half)
                 (Node EN1 ^Name Lower-half)
              (Constraint Con0 ^Name prevent-short-out)
          (Functiontree N6 ^Node NN1 NN2 NN3)
              (Node NN1 ^Name Covering)
              (Node NN2 ^Name Holding)
              (Node NN3 ^Name Electric-connection)
          (Available A1 ^Name Battery ^Node BN1)
              (Node BN1 ^Surface-diameter 0.02in)
```