Dominic II: Meta-Level Control in Iterative Redesign

Mark F. Orelup
Graduate Research Assistant

John R. Dixon
Professor
Mechanical Design Automation Laboratory
Department of Mechanical Engineering
University of Massachusetts
Amherst, MA 01003

Paul R. Cohen
Assistant Professor
Experimental Knowledge Systems Laboratory
Department of Computer and Information Science
University of Massachusetts

Melvin K. Simmons
Artificial Intelligence Branch
General Electric Corporate Research and Development
Schenectady, New York 12301

Abstract

This paper describes the meta-level control system of a program (Dominic) for parametric design of mechanical components by iterative redesign. We view parametric design as search, and thus Dominic is a hill climbing algorithm. However, from experience with Dominic we concluded that modeling engineering design as hill climbing has several limitations. Therefore, a need for meta-level control knowledge exists. To implement meta-level control, we have taken the approach of dynamically modifying the way hill climbing is performed for this task, rather than requiring the addition of domain-specific control knowledge. We have identified the limitations of hill climbing, constructed various generic hill climbing strategies, and developed a meta-strategy to guide the application of the strategies. The program monitors its own performance for unproductive efforts and selects among different strategies to improve its performance as it designs. This meta-level control significantly improves the performance of the program over the performance of an earlier version.

1 Introduction

Engineering design is widely recognized to be an iterative process; thus the control of iterative processes is critical to progress in developing computational models of design. This paper describes a meta-level control system for a program (Dominic) that performs parametric design of mechanical components by iterative redesign. We view parametric design as search through a design space where each point on a hill is a design. From this view, we have developed Dominic as a hill climbing algorithm for the task of parametric mechanical design.

Dominic I [Howe 86a, Dixon 87] is a domain-independent program generalized from two expert systems [Dixon 84b, Kulkarni 85]. It solves that large class of parametric component design problems (i.e., the design variables are known, but their values are not) which require essentially no conceptual innovation (an initial trial design is readily obtainable, and all non-metric decisions have been made, such as manufacturing process and material). The inference engine of Dominic I implements the iterative redesign model of the design process [Dixon 85, Dixon 84a] shown in Figure 1, which is a hill climbing algorithm. Though redesign can follow a number of distinct strategies, only a single conservative strategy was employed in Dominic I. Dominic I demonstrates a first step towards domain-independence and can be characterized as a task-level architecture [Gruber 87b]. However, it failed to produce acceptable designs in approximately twenty percent of the cases on which it was tested. We believe this was primarily due to the limitations of modeling mechanical design as hill climbing.

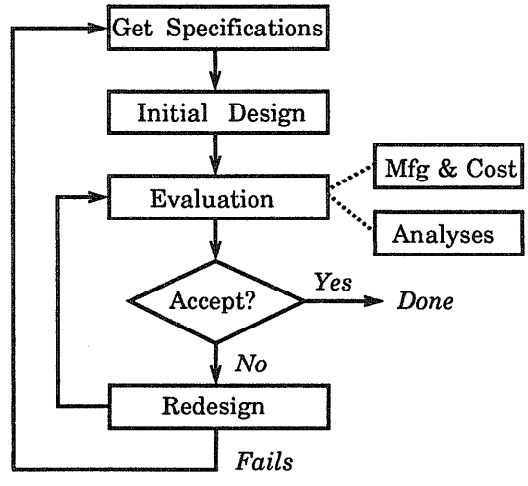

Figure 1. Iterative redesign model

Rather than add *domain-specific* control knowledge to improve performance, we have implemented a domain-independent meta-level control system that controls the application of different *generic* hill climbing strategies. Thus, the second version of the program (Dominic II) has meta-level control as depicted in Figure 2. When an examination of a history of the design effort by the Performance Monitor reveals an unproductive effort, the Strategy Selector selects a new redesign strategy from the Library of Redesign Strategies to be used in the inference engine. Strategy selection is made by a meta-strategy, based in part on a history of previous strategies. With this structure, Dominic II can recognize when one hill climbing strategy is unproductive and will substitute another, more appropriate one. This implementation of meta-level control maintains the generality of Dominic's architecture since the meta-level control system is independent of any domain.

In the remainder of the paper, after a brief overview of Dominic's methodology, the details of Dominic II's meta-level control system are described, its performance on a variety of problems in five different domains is presented, and its relation to other research in meta-level control is discussed.

2 Overview of Dominic

Before the Dominic II meta-level control system can be described, the basic features of the iterative redesign problem formulation embodied in Dominic must be explained (for more details see [Howe 86a]). In this section we describe how redesign problems are formulated and what constitutes a "redesign strategy."

2.1 Problem Formulation

Redesign class problems are structured in the following terms:

1) *Problem Parameters* define a problem instance in a domain;
2) *Design Variables* define a candidate solution. They are controlled and their values selected (within their limits) by the designer;
3) *Performance Parameters* assess the quality of a design;
4) *Constraints* are required relationships among design variables and problem parameters;
5) A *Dependency* expresses the expected effect a change in a design variable will have on a performance parameter.

2.2 Quality Assessment

For each design, levels of satisfaction (excellent, good, fair, unacceptable) determine how well a performance parameter meets the desired performance. A quality level (excellent, good, fair, poor, unacceptable) for a performance parameter is then determined by combining its level of satisfaction with its degree of importance (high, moderate, low). This method places more emphasis on the more important performance parameters. The lowest quality level of all the performance parameters becomes overall design quality level.

2.3 Redesign Strategy

A redesign strategy specifies the set of methods that perform the following actions at each step of the iterative redesign phase:

1) Select a performance parameter for attention;
2) Determine a target amount to change the selected performance parameter;
3) Select the design variable to effect the desired change in the performance;
4) Determine the amount to change the selected design variable;
5) Decide whether or not to implement the selected design variable change.

Any combination of methods for each step forms a redesign strategy. For example, one redesign strategy could be replaced by another by simply changing the method that determines how much to change the selected performance parameter, say, from "increase one quality level" to "increase to the excellent quality level." Therefore, many redesign strategies are possible, and all search the design space differently. It should be noted that the redesign strategies are defined in generic terms.

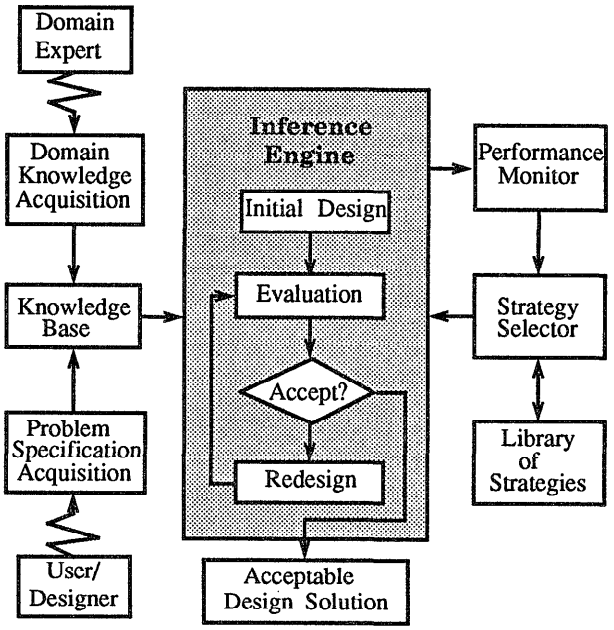

Figure 2. Dominic II architecture

3 Dominic II

In this section the meta-level knowledge and control system implemented in Dominic II are described in terms of: (1) performance monitoring; (2) the unproductive design efforts; (3) the redesign strategies; and (4) the meta-strategy.

3.1 Overview

In Dominic, design is viewed as a search for solutions in a design space whose axes are the design variables and whose topology is determined by the varying degrees of quality of the performance parameters. Dominic searches the design space via an operator (to change one or more design variables) and the redesign strategies control the application of the operator by determining which design variable(s) to change and by how much. Hence, a form of meta-level control for Dominic is the control of redesign strategies. Dominic II monitors its performance by checking if its designs are improving. If not, normally one of the unproductive design efforts (the problems of hill climbing quantified for this task) are present, and Dominic II selects a different redesign strategy to control its search based on the unproductive design effort present and the history of the strategies tried.

With this structure, at the meta-level Dominic is controlling the application of redesign strategies, or the methods by which it designs. In essence, Dominic is adapting

its problem solving approach to the current state of the problem as the following paragraphs describe.

3.2 Performance Monitoring

The current history of the design effort is the information Dominic II uses to assess its performance. For each iteration in the redesign-analysis-evaluate loop Dominic II stores the following information to record the current design effort:

1) the design variable that was modified, and by how much;
2) the performance parameter attended to, and the consequent quality change;
3) any active constraints;
4) the current design;
5) the resulting overall design quality level.

A time record of the design effort is made by collecting the above information for a number of iterations. In practice, a sufficient number was twice the number of design variables. We found that all important information for performance monitoring could be found in a time window of this duration.

Performance monitoring is done by making a performance assessment every third iteration - unless a strategy change has just been invoked, in which case a grace period equal to the duration of a time record is given. A performance assessment is made by first analyzing the sequence of overall design quality levels within a time record. If the quality has increased by at least one level since the beginning of the time record, the program continues with the current strategy. If not, Dominic II ascertains if any unproductive design efforts are present by comparing the information gathered in the time record to the criteria of each unproductive design effort described below. If no unproductive design effort is discovered, the program continues without a strategy change.

3.3 Unproductive Design Efforts

Six unproductive design efforts have been identified by observing failure cases in Dominic I. They are searched for in the time record when lack of overall progress is detected by a performance check. The unproductive design efforts are described below in terms of the design space.

1) *Creeping*: One or more design variables are being changed in steady, but ineffectual amounts. This is akin to designing along a ridge in the design space, and singular changes in the design variables lead off the ridge.
2) *Cycling*: The program is producing the same designs fairly frequently. This usually signifies that an optimum has been reached.
3) *Floundering*: The design variable changes improve the selected performance parameter, but not the overall design. This state can be described as a plateau relative to the overall design quality or as a highly nonlinear design space.
4) *Confined*: The program is making ineffectual design variable changes. The proposed design variable changes are being limited by active constraints, or by mutually exclusive performance parameters.
5) *Blocked*: Constraints are active, or performance parameters conflict such that no design variable change is allowable. This normally signifies that an optimum has been reached. This is also where Dominic I must quit.
6) *Constraint-bound*: One or more constraints are active, which inhibits the program from moving into the part of the design space it wants to explore. Constraint-bound can be the cause of Confined, Blocked, Creeping and Cycling.

All these unproductive design efforts have operational definitions [Orelup 87]. For example, the operational definition for Creeping is: a design variable has been modified monotonically in at least 30% of the iterations in the time record, and all its increments/decrements are nearly equal.

If an unproductive design effort is detected, the Strategy Selection module selects one of the strategies described below.

3.4 Strategies

Dominic II has six redesign strategies described below in terms of hill climbing. Operational definitions in terms of the components of a redesign strategy are given in [Orelup 87].

1) *Vanilla*: This is Dominic I's original strategy. It is a conservative one which seems to work well for many applications. It climbs a hill one quality level at a time and is never allowed to move down.
2) *Aggressive*: This strategy is takes much larger steps than Vanilla, and does not consider effects on the overall design (i.e., it may climb down or even off the hill).
3) *Semi-aggressive*: It takes large steps and may move down, but not off, a hill.
4) *Re-order-performance-parameters*: This strategy is used to change the order in which the performance parameters are selected for attention.
5) *Constraint-adapter*: This strategy, used in conjunction with an existing strategy, allows Dominic II to change more than one design variable at a time so that an active constraint is not violated.
6) *Big-jump*: This strategy is used to achieve much larger design variable changes than previously allowed.

Another strategy used, New-initial-design, is not strictly a redesign strategy. It is invoked when the strategy selector concludes that Dominic II is in a space which is no longer useful.

3.5 Meta-Strategies

The mapping of unproductive design efforts to strategies is not one-to-one. Deciding which strategy to implement given an unproductive design effort depends on the strategy history, specifically on whether or not the unproductive design effort has been detected before and, if so, what strategy or strategies have been previously implemented. Therefore Dominic II not only monitors its performance in design, but also its performance in terms of strategy use.

Heuristic rules of meta-strategy have been developed for selecting a strategy given an unproductive design

effort. These rules are described in general terms below with the unproductive design efforts in which they are used.

1) *Creeping*: To help Dominic II converge more quickly, but avoid bouncing back and forth over the desired value, use Big-jump until the design variable is creeping in the other direction; at that time invoke Aggressive.
2) *Cycling*: To ensure the design found is an optimum before trying elsewhere, select for attention any performance parameters not previously selected in the current time record. If this does not break the cycle, use the New-initial-design strategy.
3) *Floundering*: From the information gathered in the time record it is very difficult to tell whether a plateau is present or the design space is very non-linear. To remedy the plateau case, larger steps should be taken (e.g., from Vanilla to Semi-aggressive). For the other case, smaller steps should be taken to be more sensitive to the design space (e.g., from Aggressive to Semi-aggressive). If trying both cases did not help, make a new initial design.
4) *Confined*: To handle conflicting performance parameters, the program should take larger (but not reckless) steps in hopes that the trade-offs will work out more quickly. If this is not the case, then Dominic II is better off in another part of the design space (by invoking New-initial-design).
5) *Blocked*: If a constraint prevented a design variable change, then Constraint-adapter should be used. If not, or if Constraint-adapter did not work, make a new initial design.
6) *Constraint-bound*: Activate Constraint-adapter for the active constraints unless Constraint-adapter was activated for the same constraints in the previous time record. For that case invoke New-initial-design.

As noted before, Constraint-bound can emulate other unproductive design efforts. When this occurs, the Constraint-bound response is run. If the Constraint-bound design variables are not the ones creating the other unproductive design effort, then the response for the other unproductive design effort is run as well.

4 Results

Dominic II was tested and compared against Dominic I in twenty-seven test cases spanning five different domains: hydraulic cylinder (3 cases); I-beam (3 cases); post and beam (3 cases); v-belt (10 cases); and solar heating system (8 cases). Both programs started from the same initial design, in the same strategy, and were given the same number of iterations to work a problem. In time comparisons, Dominic II normally ran faster (up to twenty-four percent faster) than Dominic I, and in the worst case was within three percent of Dominic I's time.

In each of the test cases for hydraulic cylinder design, Dominic I failed to produce an acceptable design. Dominic II, however, was able to find at least one acceptable design in two of the three cases.

In I-beam design, in all cases Dominic II converged faster than Dominic I (from 15 - 75% fewer iterations to reach comparable designs), and Dominic II found an "excellent" design, while Dominic I found a "fair" design at best.

In the post and beam domain, Dominic II performed much like Dominic I, but was able to find one design more in all cases. The post and beam domain is more readily solved by decomposition (into a post designer, a beam designer, and a manager) than by iterative redesign [Verrilli 87].

V-belt drive design is a good test domain because all its design variables are discrete, and the design space has many local maxima. In nine out of the ten cases run, Dominic II found two or more designs while Dominic I could only find one, became Blocked, and quit. In two cases, Dominic I failed while Dominic II was able to find a "fair" design. In six of the ten cases, Dominic II found better designs than Dominic I.

The solar heating system domain consisted of designing a space heating system using the F-chart method [Beckman 77, Kreith 78]. This domain was chosen for its large number of design variables (eleven) and directly conflicting performance parameters (initial cost and annual savings). In five of the eight cases Dominic II found at least one more design than Dominic I, and in three cases Dominic II found better designs.

In summary, by adapting its approach to a design problem while designing, Dominic II clearly performs better than Dominic I. Dominic II finds more solutions, finds better solutions, converges more quickly, and can succeed where Dominic I may fail.

5 Comparison to Related Work

Work in task-level architectures includes [Clancey 85, 86, Chandrasekaran 86, Cohen 87, Gruber 87b, Marcus 85]. These are architectures that are more general than domain-specific problems but more specific than weak methods - shells with control knowledge for particular tasks. The power of these generic architectures lies in their trade-off between powerful problem solving and wide applicability, and their explicit representation of control knowledge. Dominic embodies a task level architecture for performing parametric design of mechanical components [Gruber 87b]. Though the meta-level control system in Dominic II is by no means a generic architecture for monitoring and adapting to a problem space, it does express the power/applicability trade-off. Rather than adding domain specific control knowledge to improve performance, the meta-level control in Dominic II is independent of any domain the system is applied to.

SOAR [Laird 87] is a task-level architecture based on search. Depending on the search control knowledge added to the base system, SOAR realizes the weak methods such as hill climbing, means-ends analysis, alpha-beta search, etc. Since many of its subgoals address how to make control decisions, SOAR can reflect on its own problem solving behavior. SOAR also has a chunking mechanism for learning to add to its search control knowledge while it is running. The meta-level control system of Dominic II is in the same spirit as SOAR in that they are both trying to mold the search process to the search space. SOAR performs this through universal subgoaling, while Dominic II does it by monitoring its performance and changing its

strategies for search accordingly.

Georgeff [Georgeff 83] demonstrates the utility of problem specific strategies in heuristic search. He also discusses various methods for constructing strategies and how meta-level strategies can be used to guide the application of object level strategies. In his discussion of meta-level strategies, Georgeff anticipates Dominic II: "More general meta-level strategies could take account of information derived during the search, and could allow for dynamically changing lines of reasoning. For example, the lack of success of a strategy may suggest a corrective strategy with which to continue."

An elegant approach to control is BB1 [Hayes-Roth 85], a domain independent blackboard architecture for control. The total system consists of two blackboards, one for control and the other for the domain. The control blackboard has six levels of abstraction and controls the execution of the domain knowledge sources as well as its own. The cost-effectiveness of control reasoning using BB1 has been demonstrated [Garvey 87] though in one domain through the addition of domain-specific control knowledge. Dominic II has shown the viability of dynamically modifying the control aspects of a weak method (and therefore domain-independent control knowledge) and has demonstrated its cost-effectiveness in twenty-seven problems spanning five different domains.

Hudlicka and Lesser [Hudlicka 84] describe a system which monitors and corrects a program that performs vehicle monitoring through acoustic signals, called the DMVT [Lesser 83]. The system uses a causal model of the DMVT, which is based on a blackboard architecture, to guide diagnosis when the DMVT deviates from expected behavior. The aim of the diagnosis is to detect faulty control parameter settings or faulty hardware components. The meta-level control system in Dominic II is very similar to Hudlicka and Lesser's work, though their system is more sophisticated. Rather than using a causal model to diagnose, Dominic II simply follows a predefined decision tree, and the tuning of the system is not as flexible or adaptable since individual control parameters are not adjusted but the entire problem solving strategy is changed. However, from what was learned during this research, we believe it is possible to develop a causal model between the general symptoms of unproductive design efforts and the elements of redesign. This would then lead to the fine tuning of strategies as well.

Domineering [Howe 86b] applies Dominic I to itself, that is, to design its own configuration. The five steps of redesign are the design variables and the performance parameters are measures of performance for Dominic I, such as the time required to find a design or the best design found. Domineering converges on redesign strategies that produce good performance by Dominic I in a specific domain. Unfortunately, no formal comparison has been made between Dominic II and Domineering. For the purposes of discussion, however, a few comparisons and speculations can be made. Dominic II has implicit some of the explicit features of Domineering: the performance parameters of Domineering are implicitly specified in the definitions of the unproductive design efforts in Dominic II; and the dependencies between the aspects of performance and the steps of redesign are explicit in Domineering whereas they are implicit in Dominic II in the form of the meta-strategy. Also note that Domineering attempts to converge on one redesign strategy that yields good performance in a domain while Dominic II dynamically changes the strategies according to the characteristics of the space it is in. Though no experiments have been done, we speculate that Dominic II would perform better than Dominic I running the strategy that Domineering selects for a domain because Dominic II is more flexible and adaptable to the design space than Dominic I. A single redesign strategy probably could not do well throughout the entire design space.

6 Summary

Dominic is a general hill-climbing algorithm that finds satisficing [Simon 81] solutions to design problems in the iterative redesign class. Since engineering design is not well represented as hill climbing, the original Dominic needed extra control knowledge to improve its performance. This could have been done by adding domain specific knowledge, but to maintain the generality of Dominic's architecture for parametric design of mechanical components, we added meta-level control knowledge based on the idea that the problems that can occur in hill climbing can be solved by modifying how the hill climbing is performed. In other words, when Dominic determines it is not designing well, it changes the method by which it designs. To implement this idea, we have modeled the limitations of hill climbing in the form of unproductive design efforts; identified numerous, generic strategies for hill-climbing in this task; and have constructed a strategy to perform the mapping between the two. With the addition of this knowledge, Dominic II is able to monitor its performance for unproductive design efforts and select among different strategies to try to improve its performance as it designs. Thus Dominic II reacts and adapts to its environment and shows marked improvement over the performance of Dominic I by converging faster, finding more designs, finding better designs, and succeeding where Dominic I may fail.

Acknowledgments

The work reported in this paper was partially funded by grants from the General Electric Company and the National Science Foundation to the University of Massachusetts.

References

[Beckman 77] W. A. Beckman, S. A. Klein, and A. D. Duffie, Solar Heating Design by the F-Chart Method, Wiley Inter-science Publication, 1977.

[Chandrasekaran 86] B. Chandrasekaran, "Generic Tasks in Knowledge Based Reasoning: High-Level Building Blocks for Expert System Design", *IEEE Expert*, Fall 1986, pages 23-30.

[Clancey 85] W. J. Clancey, "Heuristic Classification", *Artificial Intelligence*, Vol. 27 (1985) pages 289-350.

[Clancey 86] W. J. Clancey, "From GUIDON to NEOMYCIN and HERACLES in Twenty short Lessons: ONR final Report 1979-1985", *The AI Magazine*, August 1986, pages 40-60.

[Cohen 87] P. R. Cohen, M. Greenberg, and J. DeLiso, "MU: A Development Environment for Prospective Reasoning Systems", *Proceedings of the National Conference on Artificial Intelligence*, pages 783-788, August 1987.

[Dixon 84a] J. R. Dixon, M. K. Simmons, and P. R. Cohen, "An Architecture for Applying Artificial Intelligence to Design", *Proceedings of the 21st ACM/IEEE Design Automation Conference*, Albuquerque, NM, June 25-27, 1984.

[Dixon 84b] J. R. Dixon, and M. K. Simmons, "Expert Systems for Design: Standard V-Belt Drive Design as an Example of the Design-Evaluate-Redesign Architecture", *Proceedings of the ASME Computers in Engineering Conference*, Las Vegas, NV, August 12-16, 1984.

[Dixon 85] J. R. Dixon, and M. K. Simmons, "Expert Systems for Design: A Program of Research", *ASME Paper NO. 85-det-78, presented at the ASME Design Engineering Conference*, Cincinnati, OH, September 10-13, 1985.

[Dixon 87] J. R. Dixon, A. E. Howe, P. R. Cohen, and M. K. Simmons, "Dominic I: Progress Towards Domain Independence in Design by Iterative Redesign", *Engineering with Computers*, Vol. 2 (1987), pages 137-145.

[Garvey 87] A. Garvey, C. Cornelius, and B. Hayes-Roth, "Computational Costs versus Benefits of Control Reasoning", *Proceeding of the National Conference on Artificial Intelligence*, pages 110-115, August 1987.

[Georgeff 83] M. P. Georgeff, "Strategies in Heuristic Search", *Artificial Intelligence*, Vol. 20 (1983) 393-425.

[Gruber 87a] T. Gruber, and P. Cohen, "Knowledge Engineering Tools at the Architecture Level", *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 100-103, August, 1987.

[Gruber 87b] T. R. Gruber, and P. R. Cohen, "Design for acquisition: Principles of Knowledge-system Design to Facilitate Knowledge Acquisition", *International Journal for Man-Machine Studies*, Vol. 26 (1987) 143-159.

[Hayes-Roth 85] B. Hayes-Roth, "A Blackboard Architecture for Control", *Artificial Intelligence*, Vol. 26 (1985) pages 251-321.

[Howe 86a] A. E. Howe, P. R. Cohen, J. R. Dixon, and M. K. Simmons, "Dominic: a Domain Independent Program for Mechanical Design", *The International Journal for Artificial Intelligence in Engineering*, Vol. 1, No. 1, 1986.

[Howe 86b] A. E. Howe, "Learning to Design Mechanical Engineering Problems", *EKSL Working Paper 86-01*, Department of Computer and Information Science, University of Massachusetts, Amherst, 1986.

[Hudlicka 84] E. Hudlicka, and V. Lesser, "Meta-Level Control through Fault Detection and Diagnosis", *Proceedings of the National Conference on Artificial Intelligence*, August 1984.

[Kreith 78] F. Kreith, and J. Kreider, Principles of Solar Engineering, Hempisphere Publishing Corp., 1978.

[Kulkarni 85] V. M. Kulkarni, J. R. Dixon, J. E., Sunderland, and M. K. Simmons, "Expert .Systems for Design: The Design of Heat Fins as an Example of Conflicting Sub-goals and the Use of Dependencies", *Proceedings of the ASME Computers in Engineering Conference*, Boston, MA, August 4-8, 1985.

[Laird 87] J. E. Laird, A. Newell, and P. S. Rosenbloom, "SOAR: An Architecture for General Intelligence", *Artificial Intelligence*, Vol. 33 (1987) pages 1-64.

[Lesser 83] V. Lesser, and D. D. Corkill, "The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks", *AI Magazine*, Vol. 4, No. 3, Fall 1983.

[Marcus 85] S. Marcus, J. McDermott and T. Wang, "Knowledge Acquisition for Constructive Systems", *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 637-639, August 1985.

[Orelup 87] M. F. Orelup, "Meta-Level Control in Domain Independent Design by Iterative Redesign", *Master's Thesis*, Department of Mechanical Engineering, University of Massachusetts, Amherst MA, 1987.

[Simon 81] H. A. Simon, The Sciences of the Artificial, 2nd edition, Cambridge, MA, The MIT Press, 1981.

[Verrilli 87] R. J. Verrilli, K. L. Meunier, J. R. Dixon, and M. K. Simmons, "A Model for Management of Problem-Solving Networks in Mechanical Design", *Proceedings of the ASME Computers in Engineering Conference*, New York, NY, August, 1987.