A General Synthesis Engine: Making MICON Domain-Independent

by

William P. Birmingham, Anurag P. Gupta,
Daniel P. Siewiorek

EDRC18-07-89 S

A General Synthesis Engine

Making MICON Domain-Independent[1]

William P. Birmingham
Advanced Computer Architecture Laboratory
Electrical Engineering and Computer Science Department
University of Michigan
Ann Arbor, Michigan 48109
313-936-1590
wpb@crim.eecs.umich.edu

Anurag P. Gupta
Daniel P. Siewiorek
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, Pennsylvania 15217

November 4, 1988

[1]This work was funded by in part by National Science Foundation grant DMC-8405136 to the Demeter Project, and the Engineering Design Research Center, Carnegie Mellon University, an NSF engineering research center supported by grant CDR-8522616.

Abstract

MICON's synthesis and knowledge-acquisition tools. Ml and CGEN, provide a design paradigm and knowledge representation scheme currently used to create computer systems. However, the techniques employed by these tools can be generalized to create a *synttiesis engine* which can be used for a variety of engineering design problems. This paper describes the design of a *domain independent* Ml and CGEN. Two example applications in mechanical design and design environments are presented.

1 Introduction

MICON[3] is a complete system for designing small computer systems. The heart of MICON consists of two tool: Ml and CGEN. Ml[6] is a knowledge-based synthesis tool, and CGEN[2] is an automated knowledge-acquisition tool. These tools are designed and implemented in a manner that facilitates their extension into new domains with appropriate modifications. Ml and CGEN form a *synthesis engine*, capable of design in a wide variety of domains.

The domain-independence of Ml/CGEN arises from two sources. First, the tool's problem-solving techniques[2] work for a class of engineering design problems which are common to many disciplines. Second, Ml is written as a knowledge-based system (KBS), and a properly engineered KBS is inherently portable to new domains. The key to portability stems from the rigid separation of the problem-solver from domain knowledge[5]. It is possible to categorize every rule in Ml into either the problem-solver or the knowledge-base, but not both. Knowledge-bases for new domains can then *plugged* into the problem-solver. This scheme is successful only if the problem-solver has integrated support for creating knowledge-bases, which CGEN provides for Ml. Figure 1 illustrates the principle: in Part (a) the systems contain a problem-solver and a knowledge-base specific to computer design design. In Part (b) of Figure 1, the problem-solders remain the same, but the knowledge-bases for domain X are used.

The idea of domain-independent computer-aided design (CAD) tools is intriguing. Consider that the *engine* underlying CAD tools, be they problem-solvers or algorithms, are usually developed for a specific application and technology. For example, printed-circuit board routers can not be applied to integrated circuit (IC) routing[3]. Essentially, each new application of existing algorithms requires a new implementation, impeding the development of new tools. A domain-independent tool, however, could be quickly configured for a new application domain, without requiring significant re-coding of the problem-solver. A second advantage of a domain-independent synthesis tool is that it provides a consistent framework for studying the design process across many domains. A consistent framework provides a reference point for analyzing design by eliminating *domain bias*, or the *my-domain-is-harder-than-yours*[4] attitude faced by researchers.

The idea of portability problem-solvers has been considered by several researchers. Mitchell [7] describes the application of the VEXED IC design system[9] to mechanical engineering problems. Maher[8] presents another example of a domain independent synthesis shell. The work described here emphasizes both the general applicability of the problem-solver and the integrated automated knowledge-acquisition system, not addressed by the other work.

This paper describes the evolution of Ml and CGEN into a domain independent synthesis system. A brief review of the original architecture of both tools is presented. The process of deriving the domain-independent version is described. The Ml/CGEN system is applicable to many design problems, but not all. A characterization of eligible problems is provided. Finally, example applications in the domains of mechanical design and design environments are described.

Throughout the remainder of this paper the following notation is used:

CGENc/>: The version of CGEN used for computer design.

MICD-• The version of Ml used for computer design.

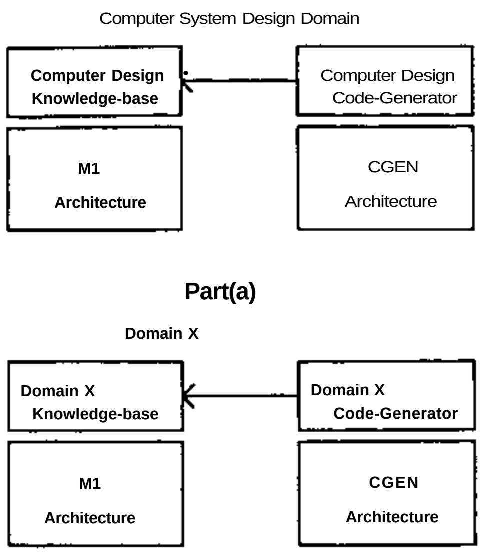
$CGEN_D$/: The domain independent version of CGEN.

$M1_D$/i The domain independent version of Ml.

[2]This includes the design cycle, functional hierarchy, templates, and knowledge-base partitions.

[3]Setliff and Rutenbarfll] are exploring solutions to this problem.

[4]Related to the infamous *NIH* problem.



Part(b)

Figure 1: Porting M1 and CGEN to new domains.

$CGEN_X$: $CGEN_{DI}$ applied to domain X.

$M1_X$: $M1_{DI}$ applied to domain X.

2 The $M1_{DI}$ and $CGEN_{DI}$ Architectures

The domain independent versions of M1 and CGEN are abstracted versions of the computer design versions of M1 and CGEN. This section reviews the original architectures of $M1_{CD}$ and $CGEN_{CD}$ and describes the how each was abstracted. A general characterization of suitable problems is given at the section's end.

2.1 M1 Architecture

As presented in a companion paper[6], the M1 problem-solver is built on the following elements:

functional hierarchy: a lattice which organizes parts by function, by indicating how functionally abstract parts are related to physical parts. An effect of the functional hierarchy is to break the design of a large system into a set of smaller, sub-problems where each sub-problem is the design of a part on the functional hierarchy.

templates: represent knowledge of how parts structurally interact with each other.

part models: a collection of information about a part including: specifications, characteristics, pins, and functional hierarchy information (called links).

design cycle: the methodology used by $M1_{CD}$ to perform design. The design cycle is composed of five basic steps:

Specification (d_{spec}): values for the specifications of a part are generated.

Selection (d_{select}): the children of a part, as described by the functional hierarchy, become *candidates*. The characteristics of the candidate's (children) part model are compared to the specifications (of the parent) generated in design step d_{spec}. The candidate that matches most closely is chosen.

Part Expansion (d_{casc}): cascadable parts (e.g. memory) are built out to the proper width.

Structure Design (d_{struct}): a template is chosen and asserted.

Calculation (d_ci_c): various calculations are performed, including: updating the values of constraints and generating design information (e.g. calculating access times).

Each step is applied iterativcly to all abstract parts in an evolving design.

Functional hierarchies appear to be a natural and generally applicable mechanism to organize components and design sub-tasks. Many other design systems have employed **a** similar technique (see Mitchell[9] and Maher[8] for examples). To be applied to Ml©/, the definition of the functional hierarchy must be slightly modified in the following two ways. First, the constituents of the hierarchy do not necessarily have to be parts (in the original sense of electronic components), but can be any object which is manipulated in the domain. For example, if Ml were applied to software design, the hierarchy could be composed of procedures or functions. The second change is to relax the implied organizational criterion of functionality, allowing any criterion, thus providing greater flexibility in the hierarchy. Since MICD does not literally interpret the meaning of the functional hierarchy these changes will not adversely affect $Ml_{D/}$'s operation.

A similar argument can be made for part models. Since part models are used more as an extensible listing of relevant attributes of a part, it can be easily extended into a new domain. To reflect the flexibility, part models are henceforth referred to as object models.

Templates and the design cycle require modification in Ml©/. Templates, its associated design step *6_Umpiau>* and the design step *dcasc* presuppose that structural knowledge is necessary for the design task. This is not always true. Revisiting the software example, structural knowledge is not used when configuring software modules. However, templates and *dcasc* can be considered forms of *design actions* for the computer design domain. Specifically, these actions are to: instantiate a part into the design and create connections between parts[5]. In the software design domain, design actions might include setting up parameters for procedure or function calls.

The design cycle for *Mlor* is generalized to:

Specification ***(d_{sp4C}):*** same as above.

Selection *(d_{S4}uet)** same as above.

Action *(daction)** the actions associated with domain are executed.

Calculation (d_{ca}/c): same as above.

The Mlco knowledge-base supports the design cycle by providing the expertise needed to execute every step for each design situation. The knowledge-base is organized into partitions, where each partition corresponds to a design cycle step. The partitions are: $k_{sp€C}$, $k_{seUc}t$* *Kaso ^template* Kaic** and k_{archi} where k^* is the procedures and rules used for various functions of Ml, such as input/output and conflict resolution.

To conform with Mlo/'s design cycle, the new knowledge-base partitions are: $k_{sp€C}$, k_{seUct9} k_a*ctu>n** $k_{ca}/_c$, and *karch-* The partition *kaction* is defined relative to the actions required for the domain. This has an impact on the knowledge-acquisition system as described in Section 2.2. The differences in the contents of partitions k_{arch} for MlCD and *MIQI* are minor, meaning that the problem-solving engine (particularly the inferencing method) is general purpose, emphasizing Mi's domain independence. The only changes necessary are related to domain-specific data structure maintenance. For example, software design does not require netlists so the code associated with the netlist data structures would be removed.

To summarize, the elements of the *MlDI* architecture are:

- object hierarchy
- object models
- design cycle: $d_{sp€C}$, <\$_{S€}u_c$u *faction,* $d_{ca}/_c$
- knowledge-base: $k_{sp€C}$, k_{stU}*cu kaction,* kc«ic, $\mathbf{k_{arch}}$

[5]Making pan connections is itself **a complex task requiring the interpretation** of **buses and** the application of several consistency checking rules.

2.2 CGEN Architecture

The CGEN architecture, as presented in a related paper[2], is built from three elements:

acquisition cycle: controls the execution of CGEN by specifying and sequencing the knowledge-acquisition tasks.

knowledge-base model: describes purpose of each knowledge partition in M1. This includes the form of rules in each partition so that syntactically and semantically correct code can be created.

design cycle model: describes how the design cycle is related to the knowledge acquisition task. In particular, this model defines the correct amount of knowledge to be collected during an acquisition session.

Changes to $M1_{CD}$'s original design cycle, specifically the step d_{action}, have a profound effect on CGEN. $CGEN_{CD}$ was designed to provide a high-level, domain specific interface to a hardware design expert. The purpose of this interface is to allow the expert to express domain *actions* without having to write code. For example,templates are described by the domain expert via a schematic drawing, which $CGEN_{CD}$ parses and casts into a form usable by $M1_{CD}$. This type of interface implies that CGEN understands how design actions should be interpreted and translated into $M1_{DI}$'s internal operators and data structures. Therefore, when $CGEN_{DI}$ is ported to a new domain modifications allowing it to interpret the actions specific to that domain must be made. These modifications apply to both the domain-expert interface and the code generator section of $CGEN_{DI}$. At present, this requires some re-coding of CGEN[6] and modifications to the original expert interface. The removal of step d_{case} does not have significant impact since it is a special case of d_{action}.

The acquisition cycle, which is partially derived from the interaction between the design cycle and the knowledge-base, is left intact. The $M1_{DI}$ design cycle does not fundamentally change its relationship to the knowledge-base. In addition, the $M1_{DI}$ knowledge-base is not substantially different from the $M1_{DI}$ definition (except as described in the preceding paragraph). This means that the way in which $CGEN_{CD}$ goes about gathering knowledge, its internal representation of acquired knowledge, and the code it generates are valid for the domain independent version.

2.3 Problem Characterization

A set of characteristics of suitable design problems is listed below. This characterization is evolving, but should provide reasonable guidelines. The characteristics are:

Design Object Hierarchy*: a well-defined hierarchy of the domain's design objects can be constructed through abstraction of each object's function or other relevant criteria.

Complex Actions: the design actions may be complex, requiring specific domain knowledge.

Ill-structured Domain*: an algorithmic solution to the design task does not exist. Furthermore, identifiable and compilable design expertise must exist.

Large Design Space: a large, fragmented, and sparse space of feasible designs exist.

Large Knowledge-base: due to the complexity of the domain related to the number of design objects and potentially complex design actions, a large amount of domain knowledge is needed.

Design Constraints*: the design space is bounded by a set of well-defined design constraints and techniques (algorithms, heuristics) for calculating their values.

Growth: an increasing number of new design objects and design styles (means of design with these objects), with an increasing vocabulary of terms to describe the design objects is typical for the domain.

Problem-solving Approach* the design task fits the design cycle.

Conditions marked with * are necessary.

An example of a design problem not suitable for $M1_{DI}$ and $CGEN_{DI}$ is a sizing problem, such as alloy design. In this problem class, the design objects, such as various metals, do not have a reasonable hierarchical relationship. Furthermore, the design cycle is not appropriate since the d_{select} step does not have an obvious function.

[6]Research is presently underway to develop a high-level language to describe design actions. This language will be interpreted by $CGEN_{DI}$ allowing automatic configuration to a new domain.

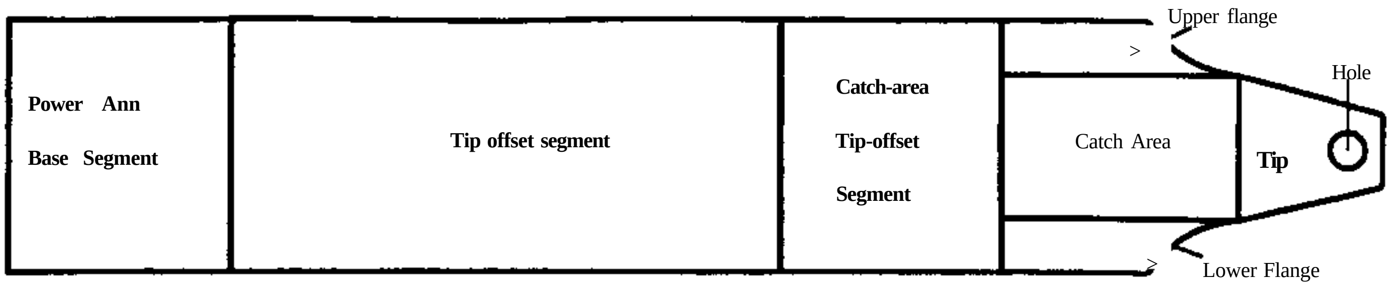


Figure 2: Lift-arm of a window-regulator

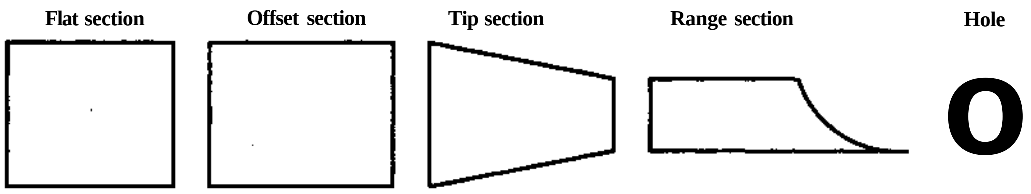


Figure 3: Primitive sections in the window-regulator design

3 Applications

Two applications of Ml/>/ and CGENo/ are described in this section. The applications areas are mechanical design and design environments[4].

3.1 Mechanical Design

Several problems in mechanical design involve selecting and integrating a set of primitives components to produce an artifact that satisfies input specifications. A particular instance of such a problem is the design of a manual window-regulator for an automobile. The window-regulator is installed between the inner and outer car door panels and serves to raise and lower the window glass. The CASE system[10] provides an integrated framework of synthesis and analysis tools for the complete problem. The MICON system was extended to automate the design of a portion of the window-regulator.

A window-regulator has three main components - backplate, lift-arm, and sector. The lift-arm, shown in Figure 2, consists of a power-arm-base segment, a tip-offset segment, a catch-area-tip-offset segment, a catch-area, upper and lower flanges, and a tip segment with a hole. Each of these, in turn, can be one of five primitive sections shown in Figure 3. The corresponding object hierarchy is shown in Figure 4. There is a direct analogy between parts and functional abstraction of Mlco and this domain. The *design process* finds a set of sized primitive sections that arc integrated to satisfy a particular functionality.

Corresponding to the nodes in the hierarchy are part models. For example, LIFT_ARM, the lift-arm abstract part, has a specification called LIFT_ARM_LENGTH; FLAT__SECTION, the flat primitive section, has characteristics LENGTH and WIDTH.

The Ml*DI* design cycle is the same as described in Section 2.1. The step *daction* just instantiates the successor

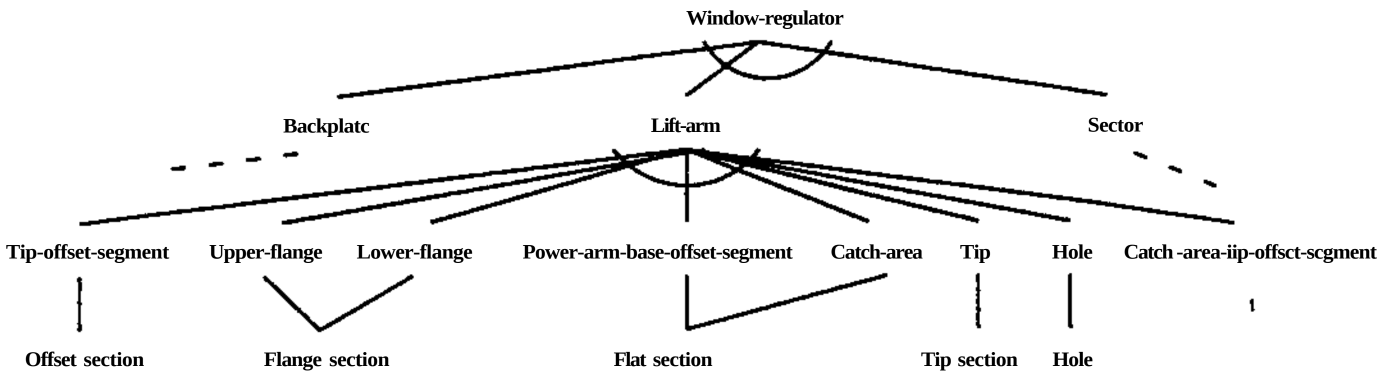


Figure 4: Hierarchy for Window-regulator design

part in the design hierarchy. However, while *MICD* produced an interconnection of pins on parts, the design process here results in the sizing of primitive parts. The sizes are calculated in the $d_{ea}ic$ step. For example, the outer width of the tip-section is computed using the equation:

TIP_OUTER_WIDTH = 2 * HOLE_RADIUS + SAFETY_FACTOR * 2

A special procedure was added to *Mlor* to print out the characteristics of primitive sections when the design was completed.

The knowledge-acquisition tool CGEN^/ in this domain captures knowledge about which parts must be used and how their size specifications and characteristics are to computed. The domain-expert interface consists of a list of parts and a set of equations listed in the *CGENor methods* input formaL

The *MIDI* and CGENo/ tool-set was easily extended and effectively handled the design of the window regulator. This was anticipated due to remarkable similarity of the window-regulator and computer system design tasks,

3.2 Design Environment

A design environment is an automated mechanism for selecting and sequencing design tools to achieve a design goal[12,4]. Design environments are useful when many tools are needed to accomplish a design task. The MICON system is a good application for design environments. An *MIDI* and CGENo/ system to control MICON, hence MI*CD* and *CGEN*$_C$*D* *is* described Jiere.

There are many tasks associated with the MICON environment, such as: design synthesis using *MICD,* database update with the ENTRY program, and knowledge-acquisition using CGENc/>. There tasks are complex and require tool usage expertise. For example, consider Figure 5 which describes the data and control flow during the knowledge acquisition task using CGENCD.

As illustrated in Figure 5, a user of the MICON tool set would benefit from the automation provided by a design environment. The mapping of various parts of the domain independent versions of CGEN and Ml into Ml£>£ and *CGENDE* are described in the remainder of this section.

The design objects in this domain consist of *design tasks* which map into *design tools*[7]. The *design process* is to find, select and invoke one or more design tools appropriate for a design task. The task/tool hierarchy is shown in Figure 6[8]. The hierarchy is organized by abstracting "tasks", as opposed to functions.

[7]This corresponds to abstract parts and physical parts respectively in *MlCD* parlance.

[8]Figures 6 and 7 arc adopted from [1].

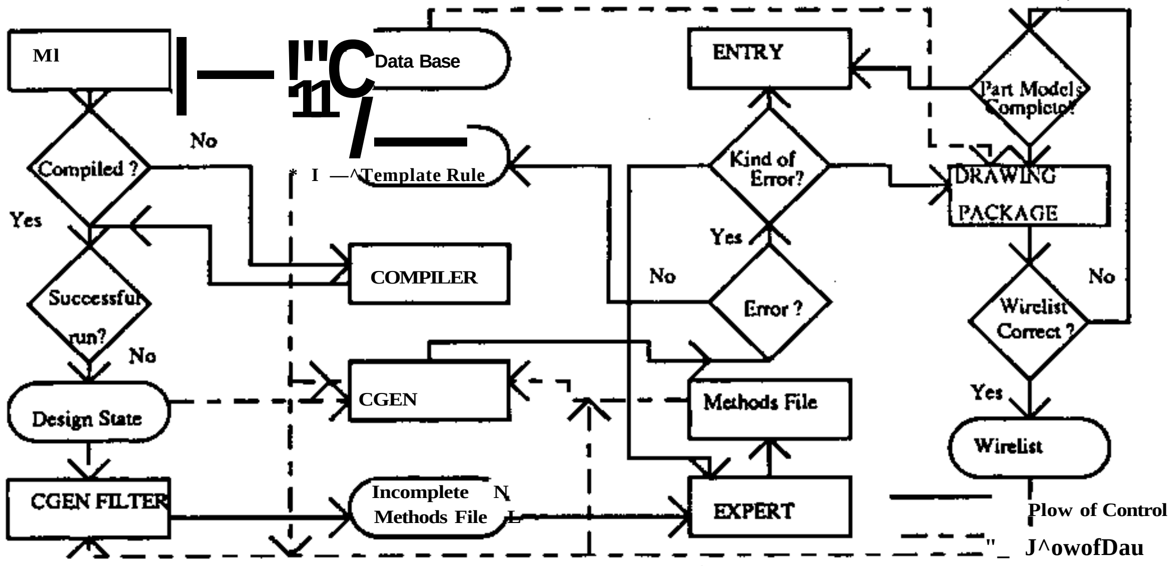


Figure 5: Knowledge-acquisition task description.

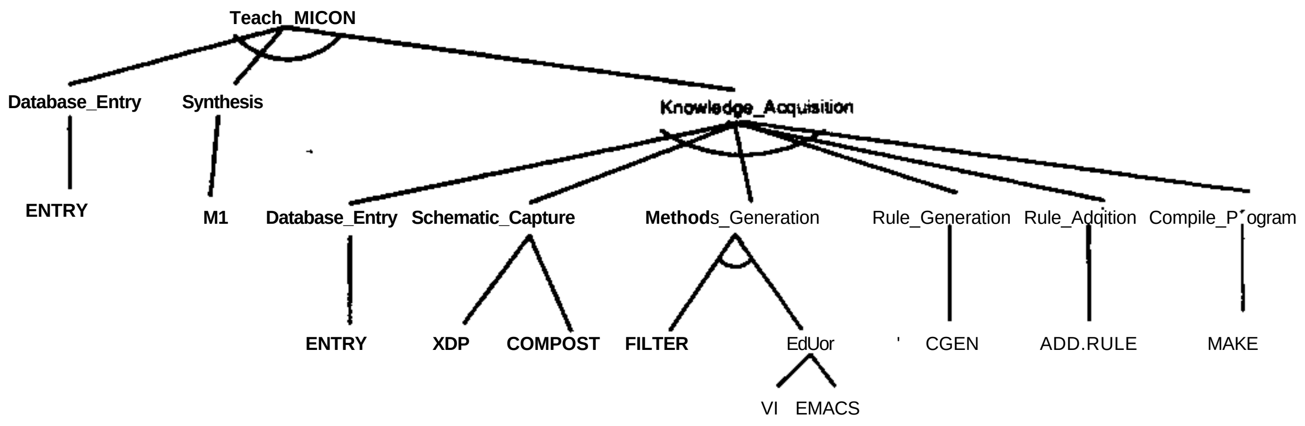


Figure 6: MICON tool/task hierarchy.

Action	Function
Query User	Replace tool by user
Delete File	remove file
Create File	create file
File Exists	test existence of file
Process Exists	test existence of process
Execution Error	fault occurred during execution

Figure 7: Design actions for design environments.

Corresponding to nodes in hierarchy are *task/tool* models. For example, the node CGENco has characteristic (DOMAIN, Computer-design); the node KNOWLEDGE_ACQUISITION has specifications Domain, Synthesis_failed, DB_Server_running *et cetera.*

The *Mlor* design cycle remains as described in Section 2.1. Example design actions for the step *Action* are shown in Figure 7.

The knowledge-acquisition task for CGEND/ in this domain consists of capture *tool usage* knowledge. For example, as Ml/)/ descends the tool/task hierarchy it needs to know how to select between various tools and how to invoke a tool[9]. This knowledge is acquired by CGENo/, converted into the design operators shown in Figure 7, and integrated into Mla/'s knowledge-base. The CGENo/ domain expert interface consists of a list of design-actions and constraints expressed in the high level CGEN *methods* input format.

The Mlo/ and CGEND/ tool set maps well into the design environment domain. The similarity of tasks between this domain and computer design is interesting to observe.

4 Summary

The Ml and CGEN systems are based on a design paradigm and knowledge representation technique which transcend a single domain. After removing some domain specific (computer design) features, a domain independent synthesis system has evolved. Thus, these tools provide a synthesis platform which can be quickly configured for a wide range of problems. Note, that when *MIQI* and CGEN^/ are instantiated for a particular application, some minor modifications are required.

Work is continuing on Ml/)/ **and** CGEN$_D$/. The tools are being applied to more difficult design problems in an attempt to find the tool limits and to better characterize appropriate problem-domains.

5 Acknowledgements

The authors would like to thank Jim Rehg and Sarosh Talukdar for their assistance with the mechanical engineering domain. In addition, the help of Ajai Kapoor and Nino Vidovic is appreciated in the design environment area.

References

[1] William P. Birmingham, Ajai Kapoor, Daniel P. Siewiorek, and Nino Vidovic. The design of an integrated environment for the automated synthesis of small computer systems. In *To appear in the Hawaii International Conference on System Sciences - 22,* IEEE Computer Society, January 1989.

[9]**Tool invocations is a compound operation composed of: ensuring inputs exist and are valid;** providing ihe correct command (sequence) to run **the** tool; **and, checking that the tool executed properly.**

[2] William P. Birmingham and Daniel P. Sicwiorck. Capturing designer expertise - the cgen system. In *Submitted to The 26th Design Automation Conference,* IEEE and ACM-SIGDA, IEEE Computer Society, 1989.

[3] W.P. Birmingham, A.P. Gupta, and D.P. Siewiorek. The micon system for computer design. In *Submitted to The 26th Design Automation Conference** IEEE Computer Society, 1989.

[4] W.P. Birmingham, A. Kapoor, D.P. Siewiorek, and N. Vidovic. The design of an integrated environment for the automated synthesis of small computer systems. In *To appear in the Hawaii International Conference on System Sciences - 22,* IEEE Computer Society, January 1989.

[5] R. Davis. Interactive transfer of expertise: acquisition of new inference rules. In *Readings in Artificial Intelligence,* Tioga, 1981.

[6] A.P. Gupta and D.P. Siewiorck. A hierarchical problem-solving architecture for synthesis. In *Submitted to The 26th Design Automation Conference,* IEEE Computer Society, 1989.

[7] N.A. Langrana, T.M. Mitchell, and N. Ramachandran. Progress toward a knowledge-based aid for mechanical design. In *Symposium on Integrated and Intelligent Manufacturing,* ASME, 1986.

[8] M.L. Maher and P. Longinos. Development of an expert system shell for engineering design. *International Journal of AI for Engineering,* Summer 1987.

[9] T.M. Mitchell, L.I. Steinberg, and J.S. Shulman. A knowledge-based approach to design. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* PAMI-7(5), September 1985.

[10] J. Rehg, A. Elfes, S.Talukdar,R. Woodbury, M. Eisenberger, and R. Edahl. Case: computer-aided simultaneous engineering. In *Proceedings of the Third International Conference on Applications of Artificial Intelligence in Engineering,* Computatioal Mechanics Institute, IEEE Computer Society, 1988.

[11] D. Setliff and R. Rutenbar. Knowledge-based synthesis of custom vlsi physical design tools: first steps. In *The Fourth Conference on Artificial Intelligence Applications,* IEEE Computer Society, March 1988.

[12] D.P. Siewiorek, D. Giuse, and W.P. Birmingham. *Proposal for Research on DEMETER: A Design Methodology and Environment.* Technical Report CMUCAD-83-5, Carnegie Mellon University Deparment of Electrical and Computer Engineering, January 1983.