# NASA Technical Memorandum 101575

## DeMAID - A Design Manager's Aide for Intelligent Decomposition User's Guide

(NASA-TM-101575) DeMAID: A DESIGN MANAGER'S AIDE FOR INTELLIGENT DECOMPOSITION USER'S GUIDE (NASA. Langley Research Center) 26 p CSCL 09B G3/61

N89-22358 Unclas 0204476

James L. Rogers

MARCH 1989

National Aeronautics and Space Administration

**Langley Research Center**
Hampton, Virginia 23665-5225

TABLE OF CONTENTS


| | |
|---|---|
| **1. Introduction** | **3** |
| **2. A Proposed Model of the Design Process** | **4** |
| **3. Components of DeMaid** | **5** |
| 3.1 The Input Data | 6 |
| 3.2 The Main Program | 9 |
| 3.3 The Graphics Package | 10 |
| 3.4 The Knowledge-based System | 11 |
| **4. Functions of DeMAID** | **12** |
| 4.1 Planning | 13 |
| 4.2 Scheduling | 15 |
| 4.3 The NxN Matrix Display | 17 |
| 4.4 Multilevel Decomposition | 20 |
| 4.5 Examine Parallelism | 22 |
| 4.6 The Dependency Matrix | 23 |
| **5. References** | **25** |

# 1. INTRODUCTION

Many engineering systems are large and multidisciplinary. Before the design of such complex systems can begin, much time and money are invested in determining the possible couplings among the participating subsystems and their parts. For designs based on existing concepts, like commercial aircraft design, the subsystems and their couplings are usually well-established. However, for designs based on novel concepts, like large space platforms, the determination of the subsystems, couplings, and participating disciplines is an important task. Moreover, this task must be repeated as new information becomes available or as the design specifications change. Determining the subsystems is not an easy, straightforward process and often important couplings are overlooked. The design manager must know how to divide the design work among the design teams so that changes in one subsystem will have predictable effects on other subsystems. The resulting subsystems must be ordered into a hierarchical structure before the planning documents and milestones of the design project are set. The success of a design project often depends on the wise choice of design variables, constraints, objective functions, and the partitioning of these among the design teams. Very few tools are available to aid the design manager in determining the hierarchical structure of a design problem and assist in making these decisions.

Recently Sobieski (ref. 1) showed the value of multilevel optimization as an approach to solving complex design problems. But to use this approach, a novel design problem must be decomposed to identify its hierarchical structure. Although much work has been done in applying AI tools and techniques to problems in different engineering disciplines, only recently has the application of AI tools begun to spread to the decomposition of complex design problems. Steward (ref. 2) developed a project management tool to organize and display the couplings among tasks in an NxN matrix format using matrix manipulations. A new tool called the Design Manager's Aide for Intelligent Decomposition (DeMAID) has been developed to implement a decomposition scheme suitable for multilevel optimization. It displays the data in an NxN matrix format and replaces the matrix manipulations with a knowledge base to provide more flexibility. In addition, if the design project has a time constraint placed on it, the design manager may have to examine the trade-offs between sequential and parallel processing. DeMAID provides that capability. Rogers (ref. 3) presents a more detailed discussion DeMAID.

PRECEDING PAGE BLANK NOT FILMED

PAGE 2 INTENTIONALLY BLANK

## 2. A PROPOSED MODEL OF THE DESIGN PROCESS

DeMAID incorporates only one model of the many possible models of the design process. This model parallels Steward's (ref. 2) model of a system which defines the *structure* of a system as the way in which some parts of a system affect other parts of a system. These effects differentiate a system from just a collection of parts. The *semantics* of the system describe how and why these effects occur. The structure and semantics together completely describe the system. To attain a desirable structure, the design manager needs more formal tools to gain understanding of both the structure and the semantics of the system.

Typically, a desirable structure has a limited number of feedback links because they increase the cost of the solution. Feedback links imply that information is required before it is available which, in turn, implies that guesses must be made to initiate the process and iterations are necessary. One method of reducing feedback links is multilevel decomposition where the modules and their couplings are ordered in such a way that a number of smaller uncoupled optimization problems can be identified.

DeMAID partitions the modules of a system into circuits which represent subsystems where each module is simultaneously dependent on all of the other modules within the same circuit. Feedback links are contained within the circuits indicating that an iteration is required. Circuits are connected to each other only by feedforward links. This indicates that there is no iteration among circuits and they can be ordered in a multilevel format. Thus a complex design process can be decomposed into a hierarchical set of tasks.

## 3. COMPONENTS OF DeMAID

The four primary components of DeMAID are the input data, the main program, the graphics package, and the knowledge-based system. A diagram of DeMAID is shown in figure 1.

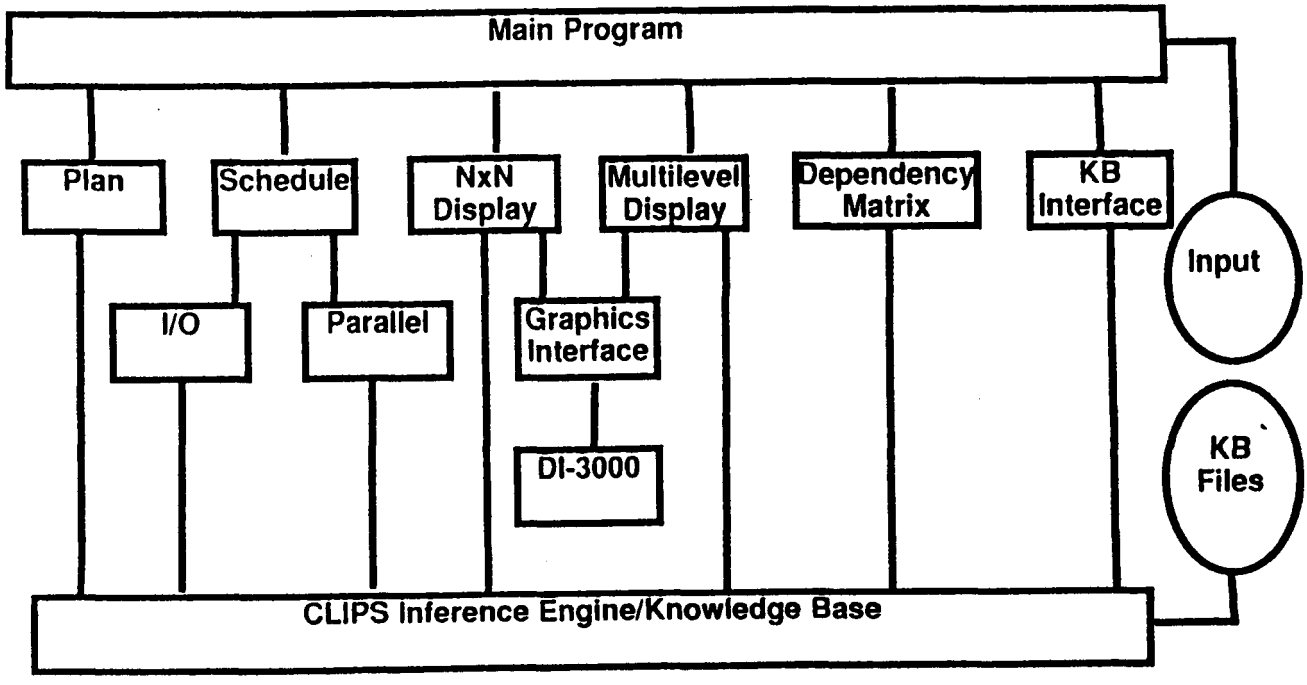


Figure 1 - A diagram of DeMAID.

### 3.1 The Input Data

The user begins the design of a complex system that is divisible into modules by determining the outputs that contribute to the objective and constraint functions of the system. The user divides the system into these modules and determines the input and output of each module creating a data file to be read by the main program. Table 1 is an example of the relationships that might be found among the different design elements. From this list of relationships, a data file is created that contains the number of modules followed by a list of the modules. The input data for the sample problem described in table 1 is shown in table 2.

The format of a single line in the list is:

**module** number name weight time output **unknown** input-list

The **module** and **unknown** items are not to be changed by the user. The **module** item is used to designate the list as module data for the inference engine. The **unknown** item is a status marker for the inference engine and changes as different stages of the reordering are passed.

The items to be changed by the user are:

*number* - a unique number, beginning with 1 and numbered consecutively. These numbers are rearranged as the modules are reordered.

*name* - a unique name.

*weight* - a weight factor. Typically the factors are 4 for an objective function, 3 for a design variable, 2 for a behavior variable, and 1 for a constraint function. The modules are ordered within a circuit based on these factors.

*time* - a time which is an estimate of how long the module will take to complete execution.

*output* - a unique name for the output of the module. Regardless of whether the output is a single value or many values, it is given a single, unique name.

*input-list* - a list of the input requirements for the module. If a module requires no input from within the system (i.e. the input is from an external source or the module is for initialization purposes) the user inputs "no-input" which must be in lower case and have the hyphen.

**Design variables**

1. DV01=f(G011,G012,G013
2. DV02=f(G011,G012,G013)
3. DV03=f(G011,G012,G013)
4. DV04=f(G003)
5. DV05=f(G003)
6. DV06=f(G014,G015)
7. DV07=f(G014,G015)
8. DV08=f(G014,G015)
9. DV09=f(G004,G005)
10. DV10=f(G004,G005)
11. DV11=f(G004,G005)
12. DV12=f(G016,G017)
13. DV13=f(G016,G017)
14. DV14=f(G016,G017)
15. DV15=f(G016,G017)
16. DV16=f(G006,G007,G008)
17. DV17=f(G006,G007,G008)
18. DV18=f(G006,G007,G008)
19. DV19=f(G009,G010)
20. DV20=f(G009,G010)
21. DV21=f(G001,G002)
22. DV22=f(G001,G002)
23. DV23=f(OB01)
24. DV99=f(G001,G003,G030)

**Constraint functions**

1. G001=f(DV16,DV17,BV02)
2. G002=f(DV18,BV02)
3. G003=f(DV01,DV02,DV03,DV04,DV05)
4. G004=f(DV06,DV10,DV11,BV04)
5. G005=f(DV07,DV08,DV10,DV11,BV04)
6. G006=f(DV12,DV13,DV16,DV17,DV18)
7. G007=f(DV12,DV13,DV16,DV17,DV18)
8. G008=f(DV12,DV13,DV16,DV17,DV18)
9. G009=f(DV14,DV19,DV20)
10. G010=f(DV15,DV19,DV20)
11. G011=f(DV01,DV02,DV03)
12. G012=f(DV23,BV03)
13. G013=f(DV23,BV03)
14. G014=f(DV06,DV07,DV08,DV23)
15. G015=f(DV06,DV07,DV08,DV23)
16. G016=f(DV23,BV01)
17. G017=f(DV23,BV01)
18. G098=f(DV18,DV26,DV32)

**Objective function**

1. OB01=f(DV23)

**Behavior variables**

1. BV01=f(DV12,DV13,DV14,DV15)
2. BV02=f(DV21,DV22)
3. BV03=f(DV01,DV02,DV03)
4. BV04=f(DV09,DV10,DV11)
5. BV50=f(DV01,DV02,DV03)

Table 1 - Relationships among the design elements of the sample problem.
The notation = f(...) means *is a function of.*

| 46 | *No.* | *Name* | *Wt.* | *Tm.* | *Out.* | *Status* | *Input* |
|---|---|---|---|---|---|---|---|
| module | 1 | TASKC10 | 1 | 27 | G010 | unknown | DV15 DV19 DV20 |
| module | 2 | TASKD07 | 3 | 12 | DV07 | unknown | G014 G015 |
| module | 3 | TASKD17 | 3 | 12 | DV17 | unknown | G006 G007 G008 |
| module | 4 | TASKD23 | 3 | 81 | DV23 | unknown | OB01 |
| module | 5 | TASKD20 | 3 | 17 | DV20 | unknown | G009 G010 |
| module | 6 | TASKD15 | 3 | 23 | DV15 | unknown | G016 G017 |
| module | 7 | TASKB03 | 2 | 53 | BV03 | unknown | DV01 DV02 DV03 |
| module | 8 | TASKC14 | 1 | 18 | G014 | unknown | DV06 DV07 DV08 DV23 |
| module | 9 | TASKC07 | 1 | 18 | G007 | unknown | DV12 DV13 DV16 DV17 DV18 |
| module | 10 | TASKC15 | 1 | 62 | G015 | unknown | DV06 DV07 DV08 DV23 |
| module | 11 | TASKD21 | 3 | 12 | DV21 | unknown | G001 G002 |
| module | 12 | TASKC04 | 1 | 15 | G004 | unknown | DV06 DV10 DV11 BV04 |
| module | 13 | TASKC17 | 1 | 35 | G017 | unknown | DV23 BV01 |
| module | 14 | TASKC06 | 1 | 23 | G006 | unknown | DV12 DV13 DV16 DV17 DV18 |
| module | 15 | TASKC03 | 1 | 24 | G003 | unknown | DV01 DV02 DV03 DV04 DV05 |
| module | 16 | TASKC13 | 1 | 17 | G013 | unknown | DV23 BV03 |
| module | 17 | TASKB04 | 2 | 70 | BV04 | unknown | DV09 DV10 DV11 |
| *module | 18 | TASKD99 | 3 | 84 | DV99 | unknown | G001 G003 G030 |
| module | 19 | TASKD11 | 3 | 22 | DV11 | unknown | G004 G005 |
| module | 20 | TASKD02 | 3 | 12 | DV02 | unknown | G011 G012 G013 |
| module | 21 | TASKC01 | 1 | 35 | G001 | unknown | DV16 DV17 BV02 |
| *module | 22 | TASKC98 | 1 | 10 | G098 | unknown | DV18 DV26 DV32 |
| module | 23 | TASKC16 | 1 | 55 | G016 | unknown | DV23 BV01 |
| module | 24 | TASKD13 | 3 | 15 | DV13 | unknown | G016 G017 |
| module | 25 | TASKD05 | 3 | 53 | DV05 | unknown | G003 |
| module | 26 | TASKD14 | 3 | 74 | DV14 | unknown | G016 G017 |
| module | 27 | TASKC08 | 1 | 53 | G008 | unknown | DV12 DV13 DV16 DV17 DV18 |
| module | 28 | TASKB02 | 2 | 17 | BV02 | unknown | DV21 DV22 |
| module | 29 | TASKD10 | 3 | 42 | DV10 | unknown | G004 G005 |
| module | 30 | TASKC09 | 1 | 98 | G009 | unknown | DV14 DV19 DV20 |
| module | 31 | TASKC11 | 1 | 14 | G011 | unknown | DV01 DV02 DV03 |
| module | 32 | TASKD16 | 3 | 67 | DV16 | unknown | G006 G007 G008 |
| module | 33 | TASKD06 | 3 | 68 | DV06 | unknown | G014 G015 |
| module | 34 | TASKD19 | 3 | 74 | DV19 | unknown | G009 G010 |
| module | 35 | TASKD03 | 3 | 46 | DV03 | unknown | G011 G012 G013 |
| module | 36 | TASKD09 | 3 | 57 | DV09 | unknown | G004 G005 |
| module | 37 | TASKD12 | 3 | 25 | DV12 | unknown | G016 G017 |
| module | 38 | TASKC12 | 1 | 19 | G012 | unknown | DV23 BV03 |
| module | 39 | TASKD22 | 3 | 57 | DV22 | unknown | G001 G002 |
| module | 40 | TASKD18 | 3 | 84 | DV18 | unknown | G006 G007 G008 |
| module | 41 | TASKD01 | 3 | 40 | DV01 | unknown | G011 G012 G013 |
| module | 42 | TASKD08 | 3 | 93 | DV08 | unknown | G014 G015 |
| module | 43 | TASKF01 | 4 | 44 | OB01 | unknown | DV23 |
| module | 44 | TASKB01 | 2 | 62 | BV01 | unknown | DV12 DV13 DV14 DV15 |
| module | 45 | TASKC05 | 1 | 12 | G005 | unknown | DV07 DV08 DV10 DV11 BV04 |
| *module | 46 | TASKB50 | 2 | 39 | BV50 | unknown | DV01 DV02 DV03 |

Table 2 - Original input data. (* indicates modules not contributing to solution which are removed during planning and the italicized labels are not part of the input file)

### 3.2 The Main Program

The main program is written entirely in FORTRAN. The other components are added by linking existing software to the main program. The other components include:

DI-3000 - the graphics package (ref. 4), the linking must include whether the display is black and white (Tektronix 4014 display) or color (VT240 display).

CLIPS (C Language Production System) - the knowledge-based system (ref. 5).

RUNCLIPS - sets up the subroutine (KBANS1) to receive data from the rules (ref. 5).

FORTNTER - the subroutines to interface the main program with the knowledge-based system (ref. 5).

The main program controls the flow of execution through a system of menus. The user first has a choice of displaying the output on the screen or writing it to a file for post-processing. The user then must select whether the display is black and white or color. Through the main menu, the user has the choice of planning, scheduling, displaying the NxN matrix, displaying the multilevel organization, examining parallelism, or displaying the dependency matrix.

The menu appears to the user as follows:

1 - Planning stage
2 - Scheduling stage
3 - Display NxN matrix
4 - Stages 1-2
5 - Stages 2-3
6 - Stages 1-2-3
7 - Multilevel display
8 - Examine parallelism
9 - Display dependency matrix
0 - Stop

Each of these areas is discussed in more detail in the section on functions of the system.

### 3.3 The Graphics Package

DI-3000 is a device independent graphics system and is the primary graphics package used at NASA Langley Research Center. The data can be displayed on a Tektronix 4014 window of the DEC VaxStation or a DEC VT240 monitor. The graphics window is divided into two parts. One is for a dialogue area for user interface and the other is for the graphics display. The graphics calls in DeMAID are very simple operations such as moving the cursor, drawing lines and circles, text, and receiving data from the mouse or arrow keys. Since all of the calls to the DI-3000 graphics package are made from a single subroutine, GRFXEC, it should pose no problem for the user to replace DI-3000 with another graphics package.

All of the calls to GRFXEC have an option parameter. The options are:

0 - initialize DI-3000.
1 - use mouse to find interface between two modules.
5 - draw graphics window.
6 - draw modules.
7 - draw boxes around circuits.
8 - draw links between modules.
10 - move cursor to upper left hand corner of dialogue area.
11 - move cursor to lower left hand corner of dialogue area.
20 - draw levels for multilevel display.
21 - draw multilevel boxes.
22 - draw multilevel links.
100 - close down graphics.

### 3.4 The Knowledge-based System

CLIPS (ref. 5) is a knowledge-based system that was developed at NASA Johnson Space Center. It is written in C, performs forward chaining based on the Rete pattern matching algorithm, and has a FORTRAN interface. There are three main parts to this knowledge-based system, the facts, the rules, and the inference engine.

*Facts* are the basic form of data in the knowledge base and are contained in a facts-list. A fact is composed of one or more fields with each field separated by a space. A field can contain a number, a word, or a string. Facts can be asserted into the facts-list by an assert command in the calling program before the inference engine is executed.

The knowledge base also contains *rules*. A rule states that specific actions are to be taken if certain conditions are met. An action may be to return data to the calling program through the FORTRAN interface or assert a new fact into the facts-list. A rule executes based on the existence or non-existence of facts in the facts-list. Currently there are 156 rules divided among seven files and are loaded into the knowledge base as needed.

The *inference engine* applies the knowledge (rules) to the data (facts) by pattern matching the facts in the facts-list against the conditions of the rule. The basic execution cycle begins by examining the knowledge base to determine if the conditions of any rules have been met. All rules with currently met conditions are placed on to the *agenda* which is essentially a push down stack. Once the agenda is complete, the top rule is selected and its actions are executed. As a result of the action(s) of the rule, new rules may be placed on the agenda and rules already on the agenda may be removed. This cycle repeats until all rules that can execute have done so. The calling program passes control to CLIPS for execution of the inference engine and CLIPS returns control back to the calling program after all the rules have been executed.

Data is returned from the knowledge base to the KBANS1 subroutine of the main program. KBANS1 has four parameters for returning data from the knowledge base. The first parameter is CHOICE and is alphanumeric. CHOICE is a pointer to the kind of data that is being stored. The actual data parameters are ALPHA (alphanumeric), and XNUM1 and XNUM2 (floating point). These parameters must agree with the parameters set up in the RUNCLIPS file. Based on the CHOICE parameter the ALPHA, XNUM1, and XNUM2 parameters are stored in variables in the common blocks SHARE and MULTIL for later usage.

## 4. FUNCTIONS OF DeMAID

DeMAID performs several useful functions to aid the user in attaining a desirable structure. These functions are (1) planning, (2) scheduling, (3) displaying the modules and their couplings in an NxN matrix format, (4) displaying the subsystems in a multilevel format, examining parallelism, and (6) displaying the dependency matrix. Each of these functions is contained in a subroutine of the main program (figure 1). The planning function is always done first followed by the scheduling function. Calling the other functions depends upon the needs of the user.

After each function is completed a file is written containing the current list of modules. This allows the user to restart the process without having to go back to the start each time. The files are assigned by executing the DESIGN.COM file before executing the program. The DESIGN.COM file assigns the folowing files to their respective units:

input file - unit 7 (input to the planning stage, user defines the name)
PLAN.OUT - unit 8 (output from the planning stage)
SCHEDULE.OUT - unit 9 (a log file for the changes made by the scheduler)
LOGFILE.OUT - unit 10 (output from the scheduling stage)
DISPLAY.OUT - unit 11 (output from the display stage)

The various rule files are loaded into the program on an as needed basis. They need not be assigned by the user before executing the program.

The functions of DeMAID are discussed in the remainder of this section using a generic design problem as a sample problem. The input to this problem is shown in Table 1. The problem has 46 modules. These modules perform one of the following tasks: (1) set the value of one or more design variables, (2) evaluate one or more constraints functions, (3) calculate intermediate results and behavior variables, and (4) evaluate the objective function. The problem is defined in terms of relationships among these four design tasks. The dependency of the objective and constraint functions on the design and behavior variables can be defined explicitly by mathematical equations. The same is true for defining the dependency of the behavior variables on the design variables. However, the question of whether new values of the design variables can be set without knowing the outcome of the function evaluations depends on the design manager's view of the problem, therefore engineering judgement is required when determining these dependencies. The main requirement is that a design variable can only depend on a function evaluation if that function is dependent on the design variable.

## 4.1 Planning

The term *planning* within the context of DeMAID means determining which modules contribute to the solution of the problem. The user begins with a list of modules as the input data. This list should contain all modules that might possibly be used in the problem. The main program calls subroutine PLANIT. The rule file PLAN.CLP is loaded into the knowledge base. The input data is read from unit 7 and asserted as facts into the knowledge base. The inference engine is executed.

The first step in the planner is to determine whether or not a module contributes to the problem by checking the output of each module against the input requirements of the other modules. If the output of the module is contained in the input list of at least one other module then that module contributes to the solution of the problem. If a module is found not to be a contributor then it is removed from the list of modules, but saved for possible use later. A message is printed to inform the user. If two modules have the same output name, then the user is asked to either remove one of the modules or to change the name of the output on one of the modules. Messages are printed to inform the user of actions that have been taken.

In the second step, the planner examines the input lists of all the modules to determine if all input requirements are satisfied by the output of other modules. Some modules have no input requirements. These modules are used for initialization purposes and represent external inputs. If an input requirement to a module is not satisfied, then the user must interactively add a new module to the list or remove the input requirement. To add a new module the user must respond to a series of queries about the name, weight, time and output associated with that module. The user must then type the module input beginning with the word "input" and a number supplied by the program. (For example: input 29 G023 G034) The word input designates the list as input for the inference engine. If there are no input requirements the user types " input no-input".

If a new module is added, its input requirements are also checked. If one or more of its input requirements are not met, then the modules removed from the list earlier are checked first to determine if they satisfy the requirement, if not, then another module must be added. The program checks the modules that have been removed during the first step to see if they satisfy the requirements. If so, they are restored to the list and a message is printed for the user. This step continues until all input requirements are satisfied. At the conclusion of this stage, a file, PLAN.OUT, is written to unit 8 for input to the scheduling stage. Table 3 is a list of the modules from table 2 after the planning function.

| 45 | *No.* | *Name* | *Wt.* | *Tm.* | *Out.* | *Status* | *Input* |
|---|---|---|---|---|---|---|---|
| module | 25 | TASKD05 | 3 | 53 | DV05 | ok | G003 |
| *module | 45 | TASKD04 | 3 | 44 | DV04 | ok | G003 |
| module | 35 | TASKD03 | 3 | 46 | DV03 | ok | G011 G012 G013 |
| module | 33 | TASKD06 | 3 | 68 | DV06 | ok | G014 G015 |
| module | 34 | TASKD19 | 3 | 74 | DV19 | ok | G009 G010 |
| module | 17 | TASKB04 | 2 | 70 | BV04 | ok | DV09 DV10 DV11 |
| module | 6 | TASKD15 | 3 | 23 | DV15 | ok | G016 G017 |
| module | 37 | TASKD12 | 3 | 25 | DV12 | ok | G016 G017 |
| module | 21 | TASKC01 | 1 | 35 | G001 | ok | DV16 DV17 BV02 |
| module | 38 | TASKC12 | 1 | 19 | G012 | ok | DV23 BV03 |
| module | 36 | TASKD09 | 3 | 57 | DV09 | ok | G004 G005 |
| module | 32 | TASKD16 | 3 | 67 | DV16 | ok | G006 G007 G008 |
| module | 31 | TASKC11 | 1 | 14 | G011 | ok | DV01 DV02 DV03 |
| module | 30 | TASKC09 | 1 | 98 | G009 | ok | DV14 DV19 DV20 |
| module | 29 | TASKD10 | 3 | 42 | DV10 | ok | G004 G005 |
| module | 28 | TASKB02 | 2 | 17 | BV02 | ok | DV21 DV22 |
| module | 27 | TASKC08 | 1 | 53 | G008 | ok | DV12 DV13 DV16 DV17 DV18 |
| module | 24 | TASKD13 | 3 | 15 | DV13 | ok | G016 G017 |
| module | 23 | TASKC16 | 1 | 55 | G016 | ok | DV23 BV01 |
| module | 20 | TASKD02 | 3 | 12 | DV02 | ok | G011 G012 G013 |
| module | 19 | TASKD11 | 3 | 22 | DV11 | ok | G004 G005 |
| module | 15 | TASKC03 | 1 | 24 | G003 | ok | DV01 DV02 DV03 DV04 DV05 |
| module | 14 | TASKC06 | 1 | 23 | G006 | ok | DV12 DV13 DV16 DV17 DV18 |
| module | 13 | TASKC17 | 1 | 35 | G017 | ok | DV23 BV01 |
| module | 10 | TASKC15 | 1 | 62 | G015 | ok | DV06 DV07 DV08 DV23 |
| module | 9 | TASKC07 | 1 | 18 | G007 | ok | DV12 DV13 DV16 DV17 DV18 |
| module | 8 | TASKC14 | 1 | 18 | G014 | ok | DV06 DV07 DV08 DV23 |
| module | 7 | TASKB03 | 2 | 53 | BV03 | ok | DV01 DV02 DV03 |
| module | 5 | TASKD20 | 3 | 17 | DV20 | ok | G009 G010 |
| module | 4 | TASKD23 | 3 | 81 | DV23 | ok | OB01 |
| module | 2 | TASKD07 | 3 | 12 | DV07 | ok | G014 G015 |
| module | 26 | TASKD14 | 3 | 74 | DV14 | ok | G016 G017 |
| module | 16 | TASKC13 | 1 | 17 | G013 | ok | DV23 BV03 |
| module | 12 | TASKC04 | 1 | 15 | G004 | ok | DV06 DV10 DV11 BV04 |
| module | 1 | TASKC10 | 1 | 27 | G010 | ok | DV15 DV19 DV20 |
| module | 11 | TASKD21 | 3 | 12 | DV21 | ok | G001 G002 |
| module | 3 | TASKD17 | 3 | 12 | DV17 | ok | G006 G007 G008 |
| module | 22 | TASKC05 | 1 | 12 | G005 | ok | DV07 DV08 DV10 DV11 BV04 |
| module | 18 | TASKB01 | 2 | 62 | BV01 | ok | DV12 DV13 DV14 DV15 |
| module | 39 | TASKD22 | 3 | 57 | DV22 | ok | G001 G002 |
| module | 40 | TASKD18 | 3 | 84 | DV18 | ok | G006 G007 G008 |
| module | 41 | TASKD01 | 3 | 40 | DV01 | ok | G011 G012 G013 |
| module | 42 | TASKD08 | 3 | 93 | DV08 | ok | G014 G015 |
| module | 43 | TASKF01 | 4 | 44 | OB01 | ok | DV23 |
| *module | 44 | TASKC02 | 1 | 40 | G002 | ok | DV18 BV02 |

Table 3 - Modules after planning. (* indicates modules added during planning function and the italicized labels are not part of the file)

## 4.2 Scheduling

The scheduling function is the heart of DeMAID. Within this context, *scheduling* means the ordering of the modules into a meaningful solution sequence while limiting the number of feedback links among the modules. The main program calls subroutine SKEDIT. The user then has a choice of scheduling by I/O requirements or by parallel requirements. Depending on the choice, the rule file SKED.CLP or PARA.CLP is loaded into the knowledge base. The data file PLAN.OUT which was output from the planning stage is read from unit 8 and asserted as facts into the knowledge base. The inference engine is executed.

If the user has selected the option to schedule based on I/O requirements, then the scheduling function reorders the modules based on their couplings. If the modules and their couplings are placed into the matrix without any regard to their ordering, then very little information regarding the desirable structure of the system is available to the design manager because the modules are probably very disorganized and contain a substantial number of feedback links. Limiting the feedback links among the modules is done by examining the couplings and grouping the modules into circuits. DeMAID also orders the modules within the circuits and orders the circuits within the design process. While Steward (ref. 2) implements the grouping into circuits with matrix manipulations, DeMAID follows the same steps but replaces the matrix manipulations for grouping by applying rules contained in a knowledge base. Because this stage requires several minutes to complete, a message is periodically issued to the user to indicate that the program is still executing.

One of the advantages of using a knowledge-based tool over matrix manipulations is the ease with which new rules can be added. This gives the knowledge-based tool more flexibility. Additional rules that were not in Steward's (ref. 2) procedure were developed to control the ordering of the modules within circuits and the ordering of circuits within the design process. The ordering is done based on the weight assigned to the modules. This step reorders the modules within a circuit by moving the modules with the highest weight to the beginning of the circuit. The modules with ever decreasing weights are moved to be below but near the top priority modules to which they are coupled. Using this method, tasks can begin as soon as possible but the modules with the highest weights are given priority.

Since limiting feedbacks is not the only means for improving the design process, the user may also wish to examine the trade-offs between limiting the feedback links and the potential gains from parallel processing. To do this the user selects the option to schedule based on parallel requirements. Even though the natural order of processing for some tasks may be sequential, these tasks can be ordered for parallel processing by artificially introducing feedbacks links and therefore iterations. This is done by assuming that certain pieces of information are available when, in reality, they have not yet been computed.

When scheduling by parallel requirements, the user must input the number of processors that are available. In this type of scheduling the processors replace the circuits found in scheduling by I/O requirements. The program computes the amount of time that would be required to execute all modules in sequence and divides that time by the number of processors available. This yields an average time per processor. In the ideal situation, all processors would complete processing at the same time which is why this average is computed as a starting point. The program begins filling the processors with the modules beginning with the modules requiring the most time. This time is then subtracted from the time available in the processor. The remaining time slots by finding the module with the maximum time that is less than the time remaining available in a processor. This process continues until all modules have been placed or there is no time slot available in any of the processors. If any module has not been place in a processor, then its time is divided by the number of processors available and added to the average time of the processors and the placement process is repeated. If all modules have been placed, then the scheduling function based on parallel requirements is complete. The program lists the number of iterations using parallel processing that can be completed before the time is equal to the sequential time. This information can be used in examining the trade-offs between sequential and parallel processing.

At the conclusion of this stage, a file, SCHEDULE.OUT, is written to unit 9. This module data in this file has two new items not found in the original data or the PLAN.OUT file. It has two sets of numbers. One is the original number of the module found in the original data, while the second is the current module number found by reordering the modules in the scheduling function. The other item is a number indicating which circuit the module is in. This circuit number is the number of the first module in that circuit. During this function a log is written to file LOGFILE.OUT on unit 10. This log keeps track of all the steps taken to find the circuits.

### 4.3 The NxN Matrix Display

Once the scheduling function is completed, the user can display the NxN matrix (figure 2). The display of the modules and their couplings in DeMAID is slightly different from that of Steward (ref. 2). The modules of the problem are placed on the diagonal of the matrix. The couplings are lines connected horizontally to a box to indicate an output from that module and vertically to indicate an input. A circle a the juncture of the horizontal and vertical lines indicates a coupling between two modules. A circle below the diagonal indicates a feedback link.

The main program calls subroutine DISPLY. The rule file DISP.CLP is loaded into the knowledge base. The data file SCHEDULE.OUT which was created by the scheduling function is read from unit 9 and asserted as facts into the knowledge base. The inference engine is executed.

The following menu appears:

1 - new top
2 - links
3 - circuits
4 - links and circuits
5 - module data
6 - interface data
7 - list modules
8 - move a module
9 - main menu

Choose one
Follow choice with
1 - 25 modules
2 - 50 modules

where:

*new top* - moves a new module to the top left hand corner of the matrix. Since only a finite number of modules (25 or 50) can be displayed at a time, the user has the option of moving up or down the diagonal to display a different group of modules. When this option is chosen, the user will be asked for a number. The number that is input will be the number of the module the user wants to appear in the top left hand corner of the matrix.

*links* - displays the links between the modules.

*circuits* - displays the circuits found by the scheduler.

*links and circuits* - displays both links and circuits.

*module data* - displays all the important data about a particular module. When this option is chosen, the user is asked for the number of the desired module. The data includes the number, name, weight, output, inputs, the modules which use the output, and the modules which supply the input.

*interface data* - displays data about the interface between two modules. When this option is chosen, the user is asked to place the crosshairs on the circle indicating an interface between two modules. The data displayed indicates the name of the interface data and the names and numbers of the two modules.

*list modules* - displays a list of all the modules with their current numbers, their original numbers, their names, and their weights.

*move a module* - allows the user to move a module to a new location in the matrix. When this option is selected, the user is asked whether or not the links and circuits should be redrawn. This helps to determine where a module can be moved, since no module should be moved outside of its circuit. The user is then asked the number of the module to be moved and the module number after which it is to be moved. The circuits and links are redrawn after the module is moved.

*main menu* - returns to the main menu. In addition, a data file DISPLAY.OUT is written to unit 11 in the same format as SCHEDULE.OUT. If changes have been made to the matrix since the scheduling function completed, then the user should replace the SCHEDULE.OUT file with the DISPLAY.OUT file before making another display.

*25 or 50 modules* - allows the user to display 25 (1) or 50 (2) modules at a time. The selection is appended on to one of the above selections. For example, the choice 42 implies that the user wishes to display the links and circuits of 50 modules.

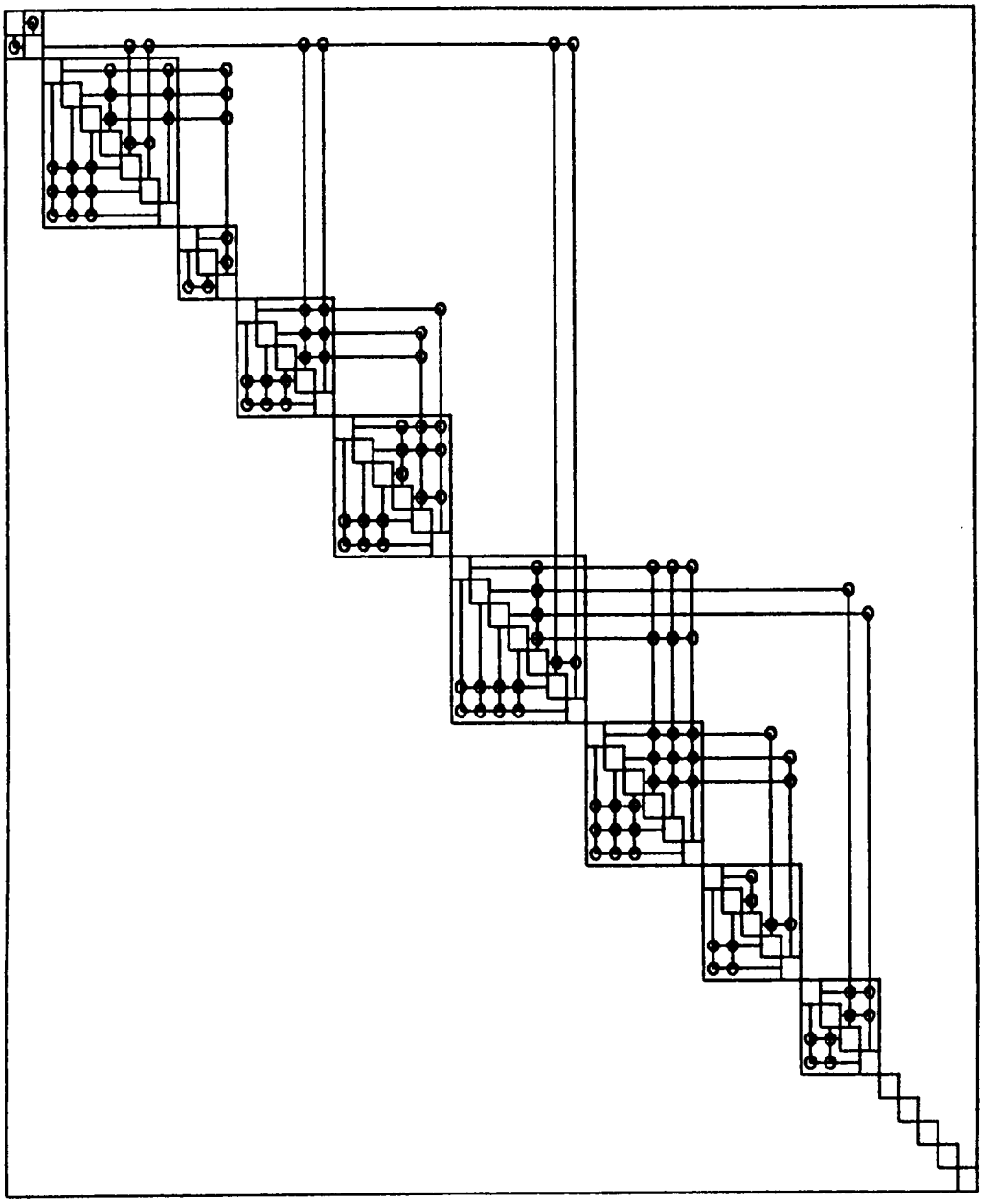

Figure 2 - NxN display of modules, circuits, and links after scheduling.

### 4.4 Multilevel Decomposition

The circuits and their couplings can also be displayed in an NxN matrix format (figure 3). By examining the circuits, it is apparent that there are no feedback links among the circuits, therefore there is no iteration among the circuits. The only iterations are contained within the circuits. Thus, once the circuits have been found during the scheduling function, it is simple to achieve a multilevel organization of the problem.

The main program calls subroutine MLTLVL. The rule file MLVL.CLP is loaded into the knowledge base. The data file DISPLAY.OUT created by the display function is read from unit 11 and asserted as facts into the knowledge base. The inference engine is executed to determine the multilevel hierarchy. As circuits with satisfied input requirements are found, they are placed on a level. A circuit is placed on the level below the lowest level containing a circuit which generates input for the circuit being placed The multilevel display of the circuits is shown in figure 4.

The following menu appears:

1 - NxN matrix display of circuits
2 - Hierarchical display of levels
3 - Main menu

where:

*NxN display* - displays the circuits only and their links.

*Hierarchical display* - displays the circuits in a multilevel format.

*Main menu* - returns to main menu

The time required for completing a circuit is computed by adding the times of the individual modules in that circuit. The time for each circuit is listed. The time to complete each level is computed by taking the maximum time to complete execution of any one circuit on that level. These times are also listed. The total time to sequentially execute each module once is computed by adding the times of all the modules. The approximate time savings by executing the levels in parallel can then be computed to determine the amount of savings of parallel execution over sequential execution. The number of processors required is determined by the maximum number of circuits on any level. For the sample problem, this information is listed in table 4.

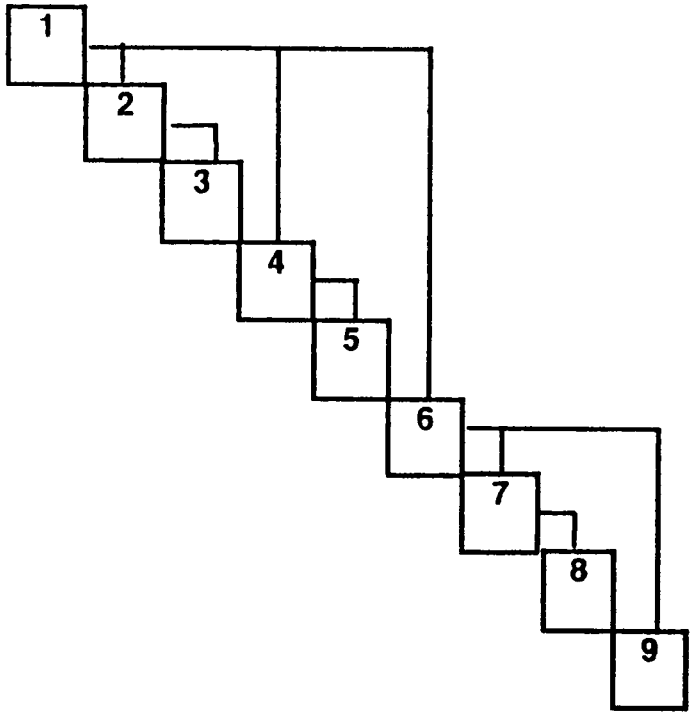


Figure 3 - NxN display of circuits and links.

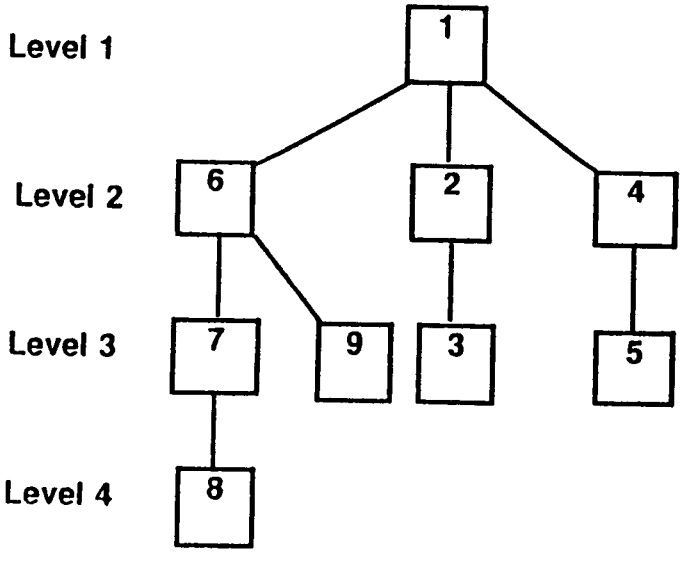


Figure 4 - Multilevel display of circuits.

### 4.5 Examine Parallelism

This function determines the savings that can be obtained from executing the modules within a circuit in parallel. This implies that scheduling was done based on the I/O requirements. The main program calls subroutine PARALL. The rule file TPAR.CLP is loaded into the knowledge base. The data file DISPLAY.OUT created by the display function is read from unit 11 and asserted as facts into the knowledge base. The inference engine is executed.

The modules within the circuits are examined to determine how they might be executed in parallel. Modules can begin execution if modules that satisfy their input requirements have completed execution. A list of all modules executing in parallel is kept along with their time requirements. The maximum number of modules executing at any one time indicates the number of processors that will be required. The amount of time that can be saved by executing certain modules in parallel and the number of processors that will be needed is listed.

Since executing the system in parallel would probably require more iterations that a sequential execution, the user needs some information on the anticipated trade-off. This function determines the amount of time it would take to complete one iteration if each module was given its own processor. This is determined by examining the time requirements of each processor and retaining the maximum time. This number is then used to determine the number of iterations that would be required before the parallel time would equal the sequential time.

| Circuit | Level | Time |
|---|---|---|
| 1 | 1 | 125* |
| 2 | 2 | 201 |
| 3 | 3 | 121 |
| 4 | 2 | 253 |
| 5 | 3 | 218 |
| 6 | 2 | 289* |
| 7 | 3 | 257* |
| 8 | 4 | 161* |
| 9 | 3 | 216 |

Table 4 - Circuit and level times for parallel execution.
The * indicates the maximum time for each level.

### 4.6 The Dependency Matrix

Another function of DeMAID is to build the dependency matrix of the problem. It is an ordered table that identifies the functional dependence between constraints and independent design variables. The main program calls subroutine DEPMAT. The rule file DMAT.CLP is loaded into the knowledge base. The data file DISPLAY.OUT which was created by the display function is read from unit 11 and asserted as facts into the knowledge base. The inference engine is executed.

Behavior variables can be evaluated using design variables, therefore each behavior variable can be replaced by a list of independent design variables. Each constraint is examined to determine its dependency on design and behavior variables. Whenever a constraint depends on a behavior variable, the dependency of that behavior variable on the independent design variables is substituted. This produces a rectangular matrix with constraint functions listed along the rows and the independent design variables along the columns (figure 5). An X marks the dependency. The numbers along the rows and columns are the numbers after the reordering of the modules. Building the dependency matrix after the planning and scheduling functions reveals dependency patterns that may prove advantageous when developing multilevel optimization algorithms.

| MOD | 1 | 2 | 3 | 4 | 5 | 10 | 11 | 13 | 14 | 15 | 18 | 19 | 20 | 24 | 25 | 26 | 27 | 31 | 32 | 33 | 37 | 38 | 42 | 43 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 1 | X | X | X | X | | | | | | | | | | | | | | | | | | | |
| 8 | 1 | X | X | X | X | | | | | | | | | | | | | | | | | | | |
| 9 | 1 | | X | X | X | | | | | | | | | | | | | | | | | | | |
| 12 | 1 | | X | X | X | X | X | | | | | | | | | | | | | | | | | |
| 16 | 1 | X | | | | | | X | X | X | | | | | | | | | | | | | | |
| 17 | 1 | X | | | | | | X | X | X | | | | | | | | | | | | | | |
| 22 | 1 | | | | | | | | X | X | X | X | X | | | | | | | | | | | |
| 23 | 1 | | | | | | | X | | | X | X | X | | | | | | | | | | | |
| 29 | 1 | X | | | | | | | | | | | | X | X | X | X | | | | | | | |
| 30 | 1 | X | | | | | | | | | | | | X | X | X | X | | | | | | | |
| 34 | 1 | | | | | | | | | | | | | X | | | X | X | X | X | | | | |
| 35 | 1 | | | | | | | | | | | | | X | | | X | X | X | X | | | | |
| 36 | 1 | | | | | | | | | | | | | X | | | X | X | X | X | | | | |
| 40 | 1 | | | | | | | | | | | | | | | | | X | | | X | X | | |
| 41 | 1 | | | | | | | | | | | | | | | | | | X | X | X | X | | |
| 44 | 1 | | | | | | | | | | | | | | X | | | | | | | | X | X |
| 45 | 1 | | | | | | | | | | | | | | | X | | | | | | | X | X |

Figure 5 - Display of dependency matrix.

## 5. REFERENCES


1. Sobieszczanski-Sobieski, J.: "A Linear Decomposition Method for Large Optimization Problems - Blueprint for Development. "NASA TM 83248, February 1982.

2. Steward, D. V.: Systems Analysis and Management. Petrocelli Books, Inc., New York, NY, 1981.

3. Rogers, J. L.: "A Knowledge-based Tool for Multilevel Decomposition of a Complex Design Problem." NASA TP 2903, January 1989.

4. DI-3000 User's Guide, Precision Visuals Inc. Document Number DI3817, Release Number 4, March 1984.

5. Riley, G.; Culbert, C.; and Savely, R. T.: "CLIPS : An Expert System Tool for Delivery and Training." Proceeding of the Third Conference on AI for Space Applications, November 1987.

# Report Documentation Page

| 1. Report No. | 2. Government Accession No. | 3. Recipient's Catalog No. |
|---|---|---|
| NASA TM-101575 | | |

| 4. Title and Subtitle | 5. Report Date |
|---|---|
| DeMAID - A Design Manager's Aide for Intelligent Decomposition User's Guide | March 1989 |
| | 6. Performing Organization Code |
| 7. Author(s) | 8. Performing Organization Report No. |
| James L. Rogers | |
| | 10. Work Unit No. 505-63-11-01 |
| 9. Performing Organization Name and Address | 11. Contract or Grant No. |
| NASA Langley Research Center<br>Hampton, VA 23665-5225 | |
| | 13. Type of Report and Period Covered |
| 12. Sponsoring Agency Name and Address | Technical Memorandum |
| National Aeronautics and Space Administration<br>Washington, DC 20546-0001 | 14. Sponsoring Agency Code |

15. Supplementary Notes

16. Abstract

A design problem is viewed as a complex system divisible into modules. Before the design of a complex system can begin, much time and money are spent in determining the couplings among modules and the presence of iterative loops. This is important because the design manager must know how to group the modules into subsystems and how to assign subsystems to design teams so that changes in one subsystem will have predictable effects on other subsystems. Determining these subsystems is not an easy, straightforward process and often important couplings are overlooked. Moreover, the planning task must be repeated as new information becomes available or as the design specifications change. The purpose of this research is to develop a knowledge-based tool called the Design Manager's Aide for Intelligent Decomposition (DeMAID)to act as an intelligent advisor for the design manager. DeMaid identifies the subsystems of a complex design problem, orders them into a well-structured format, and marks the couplings among the subsystems to facilitate the use of multilevel tools. DeMaid also provides the design manager with the capability of examining the trade-offs between sequential and parallel processing. This type of approach could lead to a substantial savings or organizing and displaying a complex problem as a sequence of subsystems easily divisible among design teams. This report serves as a User's Guide for the program.

| 17. Key Words (Suggested by Author(s)) | 18. Distribution Statement |
|---|---|
| Knowledge-based system<br>Design<br>Multilevel decomposition<br>Sequential vs. parallel processing | Unclassified-Unlimited<br>Subject Category 61 |

| 19. Security Classif. (of this report) | 20. Security Classif. (of this page) | 21. No. of pages | 22. Price |
|---|---|---|---|
| Unclassifed | Unclassifed | 26 | A03 |