J. Indian Inst. Sci., Nov.–Dec. 1996, **76**, 777–792.

A knowledge-based design environment for spacecraft

KAMALINI MARTIN*, A. S. GANESHAN* AND H. N. SHIVASANKAR**
ISRO Satellite Centre (ISAC), Bangalore 560 017, India. *email:martin@isac.ernet.in
**Department of Computer Science and Electronics, Bangalore University, Bangalore 560 001.

Received on November 24, 1994; Revised on July 14, 1995.

Abstract

A knowledge-based environment for configuration design studies is presented which assists in the conceptual stage design of spacecraft. During this stage, major decisions need to be taken on spacecraft design drivers such as spacecraft mass, power, size, launch vehicle, cost, reliability, etc. This requires a deep understanding of the interacting subsystems of the spacecraft and a thorough examination of the feasible design options and tradeoffs. The environment aids the effective integration of heterogenous data and methods from various disciplines. The dynamic sequencing and control of the design procedure is also highly flexible. The tool has been utilised satisfactorily at the ISRO Satellite Centre for the configuration design of a geosynchronous communications and multipurpose spacecraft.

Keywords: Spacecraft design, knowledge base, blackboard architecture, integrated design environment, design and analysis aid.

1. Introduction

The increasing complexity of today's aerospace vehicles requires ever-more sophisticated design approaches and analysis techniques. During the early conceptual design stage, important decisions need to be taken on spacecraft design drivers such as its mass, power, size, launch vehicle choice, cost, reliability, etc. The decisions taken here largely dictate the total cost and technology elements involved that will finally be incorporated into the design. Conceptual design is a recursive process involving many feedback loops and successive refinements. With each iteration more and more of the potential design options are examined and this results in a better understanding of the impact of design decisions. The final quality of the design depends on how much of the design space is explored.

Design tradeoffs involve features that improve one aspect of the design (such as propulsion subsystem in spacecraft) while penalising another (such as structure). Integrated design brings out the benefits and penalties of each design parameter change and its effect on the overall design merit. The efficiency with which configuration design studies can be accomplished is often largely dependent on the coordination of the design programs and their data interchange requirements than on the processing speed of the computer systems.

There is usually some organisation or hierarchical decomposition in complex problems. Generally, the hierarchy is organised along functional decompositions of the task

to be performed. In blackboard systems, a problem is decomposed to maximise the independence of the subsystems (knowledge modules). In particular, modules that generate and fill the solution space (domain dependent) are separated from modules that determine their utility (control). The decomposed domain modules are then reorganised within an integrating control hierarchy. This approach is followed in integrated design approach for spacecraft (IDEAS). This paper presents a design environment which assists spacecraft configuration design and is built using blackboard control architecture. The purpose of IDEAS is to aid in the complex task of integrating a collection of individual, heterogenous design modules into a single system and activating them intelligently in the process of transforming an initial goal into a target design. Examples of IDEAS are included in the paper in the form of design traces.

2. The spacecraft design problem

The design environment must integrate diverse and heterogenous sources of knowledge in each of the subsystem domains. To focus on what a spacecraft design entails, a brief description of a spacecraft is discussed below.

2.1. *Spacecraft systems*

A spacecraft is a product of a variety of engineering disciplines. What distinguishes the application of these engineering principles to a space system from others is the need to realise systems that could operate unattended for long durations and with a high degree of reliability in the hostile environment of space. Berlin[1] discusses the problems encountered by systems in space.

The main objective of spacecraft design is to make the systems lightweight and compact (due to prohibitive launching cost per kg mass), and low power consuming (since all power must be generated onboard the spacecraft). In addition, systems intended to operate in space must face difficult environmental conditions and therefore be extremely rugged and reliable. This calls for a high degree of design optimisation needing tradeoffs of the multiple constraints.

2.2. *Spacecraft configuration*

As discussed by Kasturirangan[2], the configuration of a spacecraft is initiated by the specification of a mission goal(s), usually related to the areas of science, application or technology. In IDEAS, such a mission goal is the area of geosynchronous communications. Once the basic payload, orbit, and mission duration are defined, a spacecraft 'bus' which supports the mission must be designed.

The configuration design of spacecraft being a complex one, the first step is to partition the design process into smaller or simpler design subproblems. Partitioning or problem decomposition could be done in many ways. One method is based on different distinct disciplines that are needed for the spacecraft design. These distinct areas form separable design entities termed subsystems. Any typical spacecraft bus would be composed of a number of distinct disciplines, termed subsystems. The important supporting

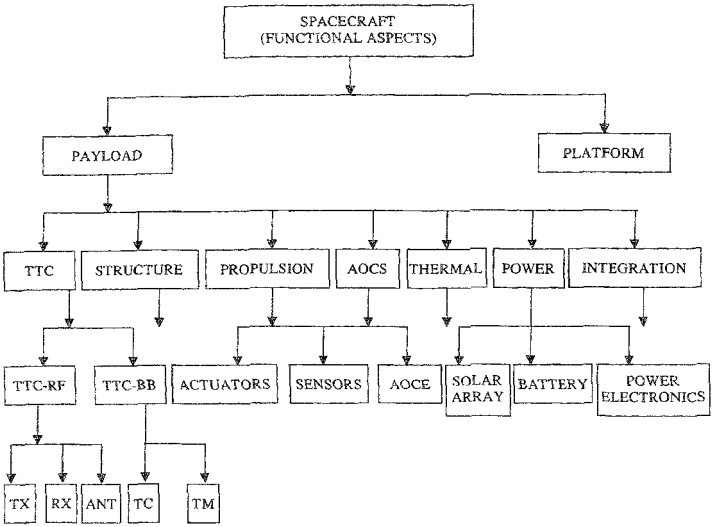


FIG. 1. Tree structure of spacecraft design.

subsystems of the satellite may be listed as power, AOCS (attitude and orbit control), TTC (telemetry, tracking and command), thermal, structures, propulsion, payload, mechanisms and AIT (assembly, integration and test of the spacecraft). Some of these subsystems are themselves multidisciplinary; for example, AOCS needs mechanical, electronics, physics and mathematics expertise. Figure 1 shows the block diagram of the spacecraft configuration process, showing a subset of the disciplines actually involved. Problem partitioning can be repeated at different levels of abstraction until sufficient component details are achieved. For example, the TTC (satellite link to ground) subsystem which is a component of the spacecraft can be subdivided into baseband (TTC-BB) and high-frequency communications (TTC-RF), while the baseband can further be subdivided into uplink (TC) and downlink (TM) and so on.

Each of the above subsystems has its own design considerations and constraints. Such constraints may be ordered in different ways in different subsystems. Each of the subsystems may offer a number of design options, all of which meet the given requirements, but with different mass, power, reliability, cost and other penalties.

Gillam[3] has emphasized the importance of building an integrated design environment which provides a repository of analysis tools, existing models, designs, contextual in-

formation about each design, and stores the expertise gained from long experience by many experts in each of the multidisciplinary domains.

2.3. *Need for knowledge-based implementation*

The individual discipline designs of spacecraft are knowledge modules involving many diverse factors such as:

(i) solving calculations, as in the case of orbital parameters,
(ii) using heuristics, as in the case of locating elements like payload transponders, battery, receivers, etc., on the spacecraft structure,
(iii) interpolations from data tables, as in the case of launch vehicle dynamic and structural constraints on the spacecraft for a chosen launch vehicle,
(iv) solving sets of equations, as in the case of estimation of link margins,
(v) inferencing from data base as in the case of estimation of package sizes and so on.

The design tasks within each design module include one or more of the following:

(i) synthesis, where a feasible design option is picked (internal to the design module), from the set of all possible design configurations for that subsystem,
(ii) optimisation, where the parameter values are sized for the given design requirement,
(iii) decision making, where some parameters may be discarded, while others are included and optimised,
(iv) evaluation of output parameters from input parameters,
(v) analysis such as merit ordering of the design options based on given criteria like mimimum mass, minimum power, etc.

and several supporting activities. Such tasks may utilise methods which are either computational or heuristic or a combination of both.

In this situation, the purely classical programming approaches are not satisfactory since an algorithmic solution to a design problem is not feasible[4].

Since the designs presently achieved by humans involve knowledge which cannot be fully formalised and automated, an environment to aid the design process rather than automate it has been proposed. Similar efforts have been described by Freksa[5]. The human designers using this environment can then focus on the more creative aspects of design and less on the data management or routine computations.

3. The IDEAS environment

It is expected that a software tool based on expert system methodologies should be an active partner in the decision-making process, perhaps guiding the decision maker, while leaving the primacy of judgement with the user. Such an environment[6,7] has been proposed for spacecraft configuration and is described here. The design procedure and the new organisation style adopted for providing a flexible environment is described below.

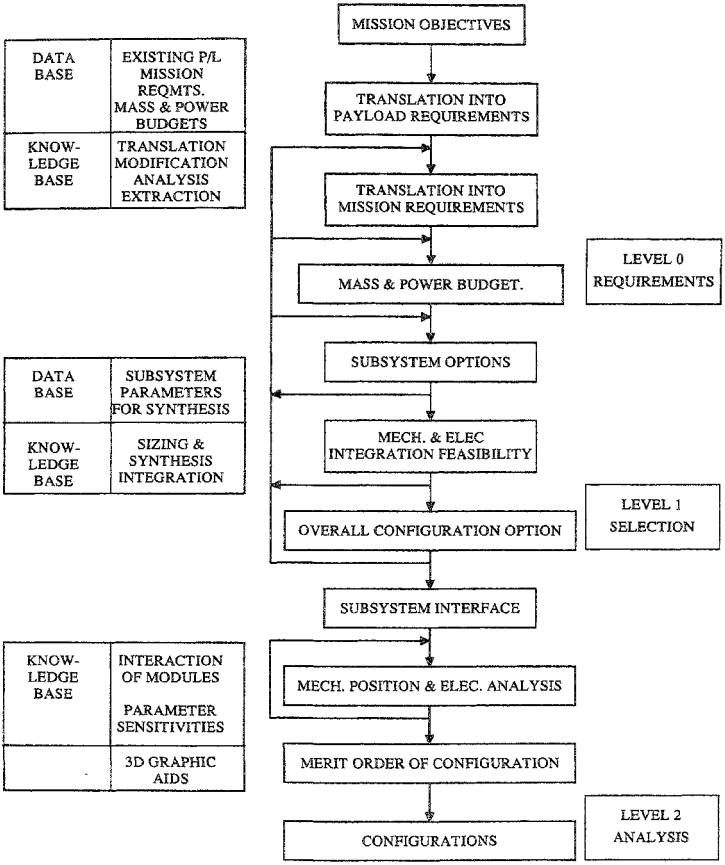


FIG. 2. Design process.

3.1. *Design procedure*

The design procedure within the IDEAS environment transforms the initial goal statement into the final target specifications. In general, the transformation is achieved by

selecting and activating suitable sequences of design modules. Design modules operate on the goal and subsequent requirements by evaluating design parameters employing different methods.

The design procedure for all configurations is organised or partitioned at three levels following the three-phase design model described by Coyne *et al.*[8] The three levels of design are classified in IDEAS as requirements, selection, and analysis phases (Fig. 2).

To achieve the three levels of design, IDEAS is organised as a modular, flexible and intelligent environment in which the human designer can configure spacecraft.

3.2. *Organisation*

Figure 3 shows the organisation of IDEAS. The essence of this modular structure is to partition the problem domain (spacecraft configuration) into a kernel containing domain-specific information and a supervisor containing the control knowledge. The intent of this separation of knowledge is to provide an environment for different applications

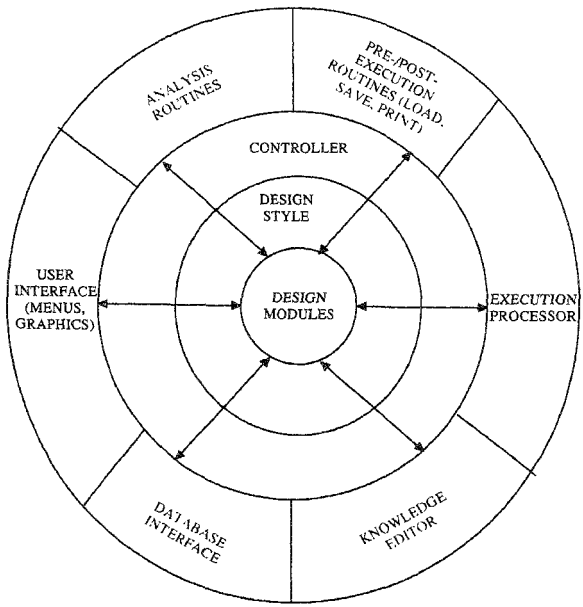


FIG. 3. Features of IDEAS.

(geostationary communication, geostationary mulitipurpose, low-earth remote sensing, scientific applications, etc.) within the same configuration domain.

3.2.1. *The kernel*

The multidisciplinary/subproblem design modules are executable files central to the system. These design modules representing domain knowledge are blackboard objects which are the basic unit of representation. The modules are embedded as suggested by Corkill[9] and may be called upon to solve different kinds of applications, *i.e.*, spacecraft designs. Some modules may be sufficiently general to be used in more than one type of design (*e.g.*, TTC-BB module may be used in other styles of spacecraft designs), whereas some modules are very specific to a particular style (*e.g.*, orbit calculation is obviously quite different in geostationary orbits as against low-Earth orbits). The modules have entirely different types of reasoning, transformation methods, etc., and have therefore been developed in different languages, *e.g.*, C, C++, Fortran, FoxPro, etc. The domain modules are typically of two types—evaluator and design option. Both types transform the input variables into output variables, either by mathematical processing or by knowledge-based reasoning, etc. The evaluator type produces one set of values for the output parameters as in classical programming. The design option type can produce a varying number of sets of output parameters. This number corresponds to the possible number of hypotheses or feasible subsolutions. The number may be more than one, *i.e.*, many subsolutions are feasible, or even zero, *i.e.*, there is no possible subsolution in which case the design inputs must change by some means.

3.2.2. *The design style*

The next layer is called the design style or procedure. This layer is one of the key features of IDEAS and is integral in providing flexibility and extensibility. The knowledge content pertinent to a design style or procedure can be created, edited and maintained in a file entirely independent of the design session itself. The design session then starts with a choice of any desired design style; for example, geostationary communication or remote sensing or scientific experimental style, etc. Each style is a knowledge base in the form of a TurboProlog internal database. The relevant style is selected and loaded into the environment at the start of the design session.

The main details in the design style file relate to either the design module or the design parameter. These details are treated like slots in a frame. Some examples of design parameters are shown in Table I. Thus, design parameters are described by entries in slots corresponding to name, type, unit, if any, default value, if any, and so on. These details are required whenever the value of a design parameter is updated within the design session. Design module details include name, type, input parameter list, output parameter list, parent module name, etc. Typical examples are shown in Table II.

3.2.3. *The controller*

The next layer in the organisation is the controller. This is written in TurboProlog, and by using the design style, activates the subproblem modules either under user guidance

Table I
Design parameter details

Parameter	*Type*	*Unit*	*Default*
Orbit	String		GEO
Mission life	Real	Years	7
Propellent type	String		Bipropellant
TM channels required	Integer		1200
Panel temp. EOL	Real	dec C	55
Bias stability	Real	deg	0.002
Link details	Unstructured list of real		
Mass allocation	Structured list of real		

or automatically to solve the design problem. This control component is critical in any intelligent system, since it determines, implicitly or explicitly, the problem-solving strategies to be applied. Blackboard architecture for control has been well documented by Hayes-Roth and others[10,11]. The last layer is the user interface which provides a large number of facilities to the user as shown in Fig. 3. The architecture of the controller is discussed later.

4. Control architecture

Blackboard architecture provides a general-purpose, powerful and flexible environment for the solution of problems that require a variety of input data and a need to integrate diverse information, as in IDEAS.

4.1. *Control functions*

The environment implements the design process by activating the design modules like a team of cooperating experts coordinated by a supervisor or controller. The control problem includes sequencing of the module to be activated, selection of a feasible option and the evaluation of the configurations.

The function of blackboard control is to invoke the executable modules in an intelligent manner to transform the goal to target. When all executable modules which are

Table II
Design module details

Parameter	*Example 1*	*Example 2*
Module number	11	15
Module type	Evaluator	Design option
Module name	Launch vehicle selection	TTC-baseband
Module parent	Requirements transformation level	TTC subsystem design
Executable file name	lvsel.exe	ttcbb exe
Module inputs	Mission life, dry mass.	Payload types, TM channels, TC commands, technology to be used.
Module outputs	Propellent type, ABM propellant, propellant weight, total mass, TO inclination, launch vehicles, launch vehicle geometric details	Baseband filter BW, TC subcarrier freq. (KHz), TM bit rate (Hz), min. SNR at decoder i/p, TO power requirements.

scheduled have been invoked satisfactorily, the target state has been reached. This target design is represented by the values of the parameters which are purely domain dependent.

4.2. *Blackboard control in IDEAS*

In general, the sequence and design procedure is aimed at evaluating (by numerical methods or otherwise) the design parameters. Such design parameters may be required either by the user or another design module. In response to a request from the user or another module, IDEAS determines the module to be executed and activates it. In turn, if this new module needs further input parameters, they are provided by executing additional routines. As explained by Hewett and Hewett[12], the activation of a design module is the basic unit here, unlike a rule-based system where the rule is the basic unit of execution. The action of a knowledge module makes one or more changes to the blackboard. Typical actions are:

(i) Fill parameter value (if currently unfilled).
(ii) Change parameter value (if currently filled). This results in recursion.
(iii) Add/delete a link between modules in the current design style.
(iv) Add/delete modules in the current design style.

Each change to the blackboard is an event. Each knowledge module is triggered by an event described in its trigger conditions. When such an event occurs, the module is placed on the agenda of potential actions. The agenda has two parts: triggered agenda (scheduled list) and executable agenda (ready list). A knowledge module has state-based preconditions which determine whether it is executable, and only modules on the executable agenda can be invoked.

The component responsible for selecting knowledge modules to be run is the scheduler. It uses control knowledge for selection. Control knowledge is derived from the analysis of the design style. The control loop selection of module to be executed is done as follows. A module is triggered (placed on the scheduled list) when one or more of the following events occur:

(i) User chooses a module for execution.
(ii) Its output parameters are required by another module.
(iii) Its parent is on the scheduled list.

Since the sequencing is represented as a dataflow net, the only precondition to be satisfied for a module to be placed on the ready list is that all its input parameter values be filled. The scheduler chooses or orders the scheduled list according to any one of the following strategies:

(i) Minimum number of input parameters.
(ii) Maximum number of output parameters.
(iii) Least sensitive module in backward direction.
(iv) Most sensitive module in forward direction.

4.3. *Recursion*

In case a parameter is redefined at any stage, recursion must take place. This is achieved by updating the scheduled list continually. Redefinition of any parameter value can occur at any time during the design process due to any one of the following.

4.3.1. *User changes the value of a parameter*

Since modules are already run using a defined set of values, those modules which are affected by the changed parameter must be rerun. For example, after designing the spacecraft for, say, a 10-year lifetime, the user may want to see the effect if it reduces to 5 years. This need not entail a complete redesign of the spacecraft, but certainly involves modification in those subsystems which need mission life as input, *i.e.*, chiefly power, propulsion, etc.

4.3.2. *Conflict resolution*

Different modules may utilise data in different ways, or the design sequence may allow different lines of computation before some parameter can be evaluated. When the two (or more) modules yield different values for a parameter, an arbiter module can decide the most reasonable value. All modules which have used the discarded value must be rerun. This can happen anywhere in the design. For example, dry mass of spacecraft is estimated initially in the design, say, at 1000 kg. This is modified later, say, to 1200 kg. The chosen launch vehicle may not be able take this load. Thus the design is re-iterated wherever necessary.

4.3.3. *Uncertainty in data*

In case of uncertain or incomplete information, different lines of reasoning are followed with a default initial assumption, to form a collective opinion of the most reasonable value of a parameter. This method is chosen instead of the method of attaching 'belief' weights to the parametric values and perhaps evaluating the uncertainty function. For example, the number of solar panels required to generate power is not known before the solar panel design is started. This number decides the temperature of outermost panel in stowed and deployed conditions, in transfer and on orbit conditions. However, the panel design needs the temperature of the outermost panel, and the power loads (among other parameters) before the area of the panel can be assessed. In turn, the area of the solar panel dictates how many number of panels can be accommodated. Thus the design starts with a default number of panels, which is updated when sufficient information is available, and so on.

4.4. *Dynamic sequencing*

To begin with, the design activity needs sequencing to define the process which iteratively calls the building blocks of an entity down to an acceptable component level design, as shown in Fig. 1. This means that in order to run (execute) IDEAS, the three-component knowledge modules, *i.e.*, Levels-0, -1, -2 must be run. In order to run Level-

0, a number of modules of which payload definition is one, must be run. Similarly, to run payload definition, each payload must be defined; for example, communication, remote sensing, and scientific payloads.

Sequencing is also needed to control the dataflow pattern defined by the design style. In this sequence, the procedure can be represented as a network, with the executable knowledge modules forming the nodes and 'design parameters' forming links. For example, to run Level-1, Level-0 must be run so that all necessary design parameters for Level-1 are available. Similarly, to run Level-2, Level-1 should have been run.

$$\text{Level-0} \rightarrow \text{Level-1} \rightarrow \text{Level-2}.$$

In case a parameter needs redefinition at any stage,recursion can take place, so that, for example, a module of Level-0 may be re-executed even while Level-1 is in progress.

In actual sequence, the execution facility prepares the required input parameters from values posted on the domain blackboard and formats them into an input data file, activates the executable file through DOS shell, and accesses the output parameters from the output data file and fills the appropriate parameter values on the blackboard. The input file creation and output file accessing is invisible to the user, who only sees the updating of the blackboard through the execution of the desired process.

5. Design trace

A sample subset of views of the IDEAS screen during an actual run are presented here to exemplify design procedures, particularly iteration and recursion.

5.1. *Example of iteration*

A design session starts with the selection of a module, which represents the part of the spacecraft to be designed. As an example, suppose that the user chooses to design the

SCREEN 1

Module:	Analyse IDEAS	Design	Facilities	Quit
		Execute ✓ Fill parameter values Show design modules Show parameter values Display output file Help		

✓ INDICATES POSITION OF CURSOR AND MENU ACTION

SCREEN 2

```
Module:   Analyse   Design   Facilities   Quit
          IDEAS

                    IDEAS Message
                                  Design modules schedule
                                  1. Requirement phase (L-0)
                                  2. Functional design (L-1)
                                  3. Mechanical design (L-2)

Invoking module 1:
Module name     : Requirement phase (L-0)        Choose
Module parent   : IDEAS                          Continue ✓
Module function : Function defined for submodules only   Skip
Module type     : Not executable                 Do later
                                                 Quit
```

SCREEN 3

```
Module:   Analyse   Design   Facilities   Quit
          IDEAS

                    IDEAS Message
                                  Design modules schedule
                                  1.1.1. Comm. Payload
                                  1.1.2. Met. Payload
                                  1.2. Mass budgets
                                  1.3. Power budgets
                                  1.4. Launch vehicle selection
                                  1.5. Orbit details
                                  1.6. Ground station selection
                                  1.7. S/C dimension
                                  2. Functional design (L-1)
                                  3. Mechanical design (L-2)

Invoking module 1.1.1
Module name     : Communication payload          Choose
Module parent   : Payload definition             Continue ✓
Module function : Defines the communication payload      Skip
Module type     : Designs feasible options based on input parameters   Do later
                                                 Quit
```

entire spacecraft. Then the module selected is the root of the tree in Fig. 1, *i.e.*, IDEAS. When 'Execute' is chosen from the pulldown menu as shown on Screen 1, the scheduler places IDEAS in the scheduled list, and looks up the design style to activate the corresponding object. From the style, the scheduler finds that IDEAS is not an excutable module (no executable file name in the slot for this object) and hence makes a list of the children of IDEAS as shown on Screen 2.

The modules corresponding to the children, *i.e.*, requirement analysis, functional design and physical design replace IDEAS on the scheduled list. The top module on the list is picked up, and the above sequence repeated until an executable module is reached. (Screen 3). An executable module from the scheduled list is ready to run (placed on

SCREEN 4

Analyse Design Facilities Quit
Module: Power budgets \ Requirement phase (L-0) \ IDEAS

IDEAS Message

Design modules trace
1.3. Power budgets: Done
1.2. Mass budgets: NOT done
1.1. Payload definition: Done
1. Requirement phase (L-0): being executed
IDEAS: Chosen

Design modules schedule
1.4. Launch vehicle selection
1.5. Orbit details
1.6. Ground station selection
1.7. S/C Dimension
2. Functional design (L-1)
3. Mechanical design (L-2)

Invoking module 1.4.:
Module name : Launch vehicle selection
Module parent : Requirement phase (L-0)
Module function : Allows user to select feasible launchers
Module type : Evaluates output parameters from input parameters

Choose
Continue ✓
Skip
Do later
Quit

SCREEN 5

Analyse Design Facilities Quit
Module: Mass budgets \ Requirement phase (L-0) \ IDEAS

IDEAS Message

Design modules trace
1.2. Mass budgets: Done
1.4. Launch vehicle selection: being executed
1.3. Power budgets: Done
1.1. Payload definition: Done
1. Requirement phase (L-0): being executed
IDEAS: Chosen

Design modules schedule
1.4. Launch vehicle selection
1.5. Orbit details
1.6. Ground station selection
1.7. S/C Dimension
2. Functional design (L-1)
3. Mechanical design (L-2)

Invoking module 1.4.:
Module name : Launch vehicle selection
Module parent : Requirement phase (L-0)
Module function : Allows user to select feasible launchers
Module type : Evaluates output parameters from input parameters

Choose
Continue ✓
Skip
Do later
Quit

Press Select to Choose, Forward/Backward to move Forward/Backward in the design

SCREEN 6

Analyse Design Facilities Quit
Module: Mass budgets \ Requirement phase (L-0) \ IDEAS

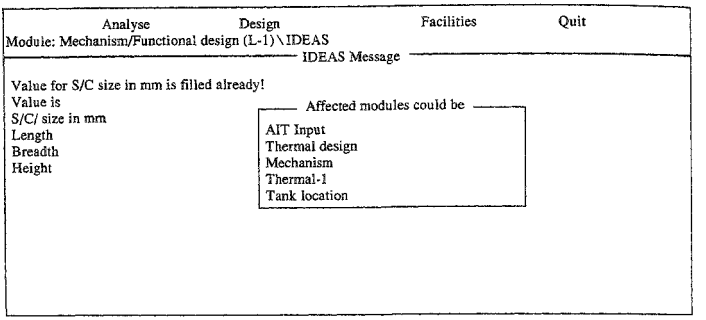
IDEAS Message

Module mechanism has no valid output for this input list

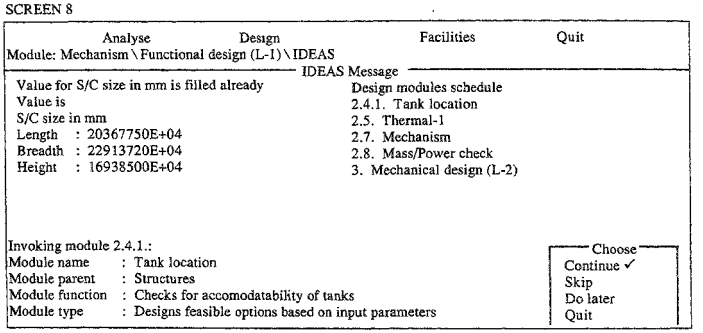
Change input

Envelope dia
S/C size in mm
No. of external antennae
MASS ALLOCATION IN kg
T.O. Area of Solar Array
O.O. Area of Solar Array ✓
No. of S. A. Wings
SA wing positions
Cooler position
Cooler height

SCREEN 7



SCREEN 8



Ready list) if all its input parameters are filled with valid values. For example, if launch vehicle selection is scheduled for execution, its input parameters, mission life and dry mass (Table II, Example 1), must be defined. If any of the parameters (say, dry mass) is not defined (because the user chose to skip that module during execution, as shown on Screen 4) the controller fills the value either by default (Table I, using the value in the 'default' slot of the parameter definition in the design style), by user choice, or running another module which outputs this parameter. For dry mass, the controller finds that mass budget is the module which outputs it, and the scheduler places mass budgets above launch vehicle selection in the scheduled list. After executing mass budgets, the original schedule is resumed (Screen 5). Activation of any module results in defining the corresponding output parameters.

5.2. *Example of recursion*

A design module of Type 2 (Table II, Example 2) may offer a number of options for a given set of input parameters as explained in Section 3.2.1. If an activation of a module results in no possible design options, the input parameters for that module must be redefined by some means for the design to proceed. This results in recursion. Screen 6 shows that a design module (Mechanisms) has been run, and there is no feasible solution. The list of input parameters for this module is looked up and displayed. If the user selects, say, O.O. area of solar panel (*i.e.*, area of the solar panel in on-orbit or deployed state) as the parameter whose value is to be changed, the module which outputs this, *i.e.*, solar panel design, is added to the schedule and rerun (it was already run once in order to fill the O.O area of the solar panel in the first place). In changing the selected option of solar panel in order to accommodate mechanisms, some other parameters may also be changed. In another case, if the user selects (spacecraft size) as the parameter to be changed to get a feasible solution for mechanisms, other modules which also use the spacecraft size as an input parameter will be affected (Screen 7). Thus, all modules which use any parameters whose values change must be rerun and added to the scheduled list as shown on Screen 8. This process continues until parameter values suitable to all modules are achieved. When the scheduled list is exhausted, the design procedure stops.

6. Conclusions

In this paper, we describe a computer-assisted flexible design procedure which is achieved by integrating heterogenous expert design knowledge and by offering heuristic search methods in the design solution space. The methodology has been implemented in IDEAS, an integrated software tool developed as a domain-independent environment based on blackboard architecture. The tool has been used successfully for the design of geosynchronous communications spacecraft. The realistic design process provides balanced solutions for multiple criteria, quantifying the tradeoffs wherever required. Although the software tool has been evolved for designing a spacecraft, the methodology is general and can be used for configuration design of any multidisciplinary artifact and can be represented as a hierarchy of modules. No fundamental limitations to the concept of integrated design tool have been uncovered in two years of intensive use. It is hoped that the methodology adopted will shed light on integrating complex systems and translating or formalising design procedures. The architecture of the tool supports the evolving nature of engineering design by ensuring flexibility, both from the viewpoint of design modules, methods and information types, as well as strategies and policies to be used in guiding the process of design.

Acknowledgements

The authors thank the members of IDEAS task team who were responsible for the development of the subsystem design modules. In particular, the authors express their sincere gratitude to Dr A.S. Prakasa Rao, Deputy Director, ISAC, and to Dr K. Kasturirangan, Chairman, ISRO, for their constant encouragement.

References

1. BERLIN, P. — *The geostationary applications satellite*, 1988, Cambridge University Press.
2. KASTURIRANGAN, K. AND SRIDHARAMURTHY, K. R. — ISRO spacecraft technology evolution, *Sadhana*, 1988, **12**, 251–288.
3. GILLAM, A. — Vehicles knowledge-based design environment, *J. Spacecraft Rockets*, 1993, **30**, 342–347.
4. — *Study on conceptual design for spacecrafts using CAE*, ESA Contract no. 6886/85/NL/PP, Final Report, 1987.
5. FREKSA, C. — Knowledge representation for interactive aircraft design, In *Expert systems and knowledge engineering* (T.Bernold, ed.), 1986, North-Holland.
6. GANESHAN, A. S. AND MARTIN, K. — *IDEAS: Integrated design approach for spacecraft design*, ISAC-TR-44-54, 91-11-22, Nov. 1991, ISAC, Bangalore, India.
7. MARTIN, K. AND GANESHAN, A. S. — *IDEAS: A design and analysis aid*, ISAC-TR-44-47, 94-06-05, June 1994, ISAC, Bangalore, India.
8. COYNE, R. D., ROSENMAN, M. A., RADFORD, A. D., BALACHANDRAN, M. AND GERO, J. S. — *Knowledge-based design systems*, 1990, Addison-Wesley.
9. CORKILL, D. J. — Embeddable problem-solving architecture: A study of integrating OP55 with Umass GBB, *IEEE Trans.*, 1991, **KDE-3**, 18–25.
10. HAYES-ROTH, B. — A blackboard architecture for control, *Artif. Intell.*, 1985, **26**, 251–321.
11. — *Blackboard architecture and applications* (V. Jagannathan, R. Dodhiawala, L.S. Baum, eds), 1989, Academic Press.
12. HEWETT, M. AND HEWETT, R. — A language and architecture for efficient blackboard systems, *9th Int. Conf. on AI for applications, IEEE*, 1993, pp. 34–40.