N90-27327

VIP: A Knowledge-Based Design Aid for the Engineering of Space Systems

Steven M. Lewis and Kirstie L. Bellman
Computer Science Laboratory
The Aerospace Corporation
Los Angeles, CA 90009-2957

Abstract

This paper describes the Vehicles Implementation Project (VIP), a knowledge-based design aid for the engineering of space systems. VIP combines qualitative knowledge in the form of rules, quantitative knowledge in the form of equations, and other mathematical modeling tools. The system allows users rapidly to develop and experiment with models of spacecraft system designs. As information becomes available to the system, appropriate equations are solved symbolically and the results are displayed. Users may browse through the system, observing dependencies and the effects of altering specific parameters. The system can also suggest approaches to the derivation of specific parameter values.

In addition to providing a tool for the development of specific designs, VIP aims at increasing the user's understanding of the design process. Users may rapidly examine the sensitivity of a given parameter to others in the system and perform tradeoffs or optimizations of specific parameters. A second major goal of VIP is to integrate the existing corporate knowledge base of models and rules into a central, symbolic form.

Introduction

VIP is a portion of Vehicles, a long-term research effort on the part of The Aerospace Corporation to develop artificial-intelligence tools for the conceptual design of spacecraft and other systems. Currently, much of the knowledge and tools used for such design are scattered throughout the corporation. The tools, while sophisticated, are often poorly documented and require a detailed knowledge of the tool itself, as well as of the system under consideration; in order to use the current tools effectively, the user of a model of spacecraft batteries needs a detailed knowledge of both the model as well as batteries. Complex and detailed setup procedures and relatively rigid structures often discourage engineers from examining alternative choices. This complexity precludes planners from using many of the existing models in the early stages of a system's design.

As older engineers retire and younger people enter the organization, a major problem has arisen regarding the maintainability of existing tools. Too often, organizations find themselves dependent on sophisticated tools that were developed by personnel no longer with the organization. These tools become difficult to maintain and, without the experts who developed the package, equally difficult to rewrite. A long-term goal is to reorganize the corporate knowledge base to make it more accessible and maintainable. Vehicles is an attempt to develop tools that allow engineers to concentrate on the rules and equations of a model, while leaving to a central system the details of implementation, solution, and interface. The hope is that models will be translated into a central, easily represented and documented form that can be useful to both experienced and casual users.

An object within the system, e.g. a frame or a slot, consists essentially of two parts, a **class** and a specific **instance**. The class holds information that is common to all instances. For example, a spacecraft may have multiple antennas. Each instance points to a common frame, of the class antenna. Within this class there is a **slot** that holds the antenna's diameter. The slot points to a template that holds common information: name, default values, limits, definitions, and default print units. The instance holds information specific to one frame: the value, the default print units, information about how the value was derived, and any user-supplied annotation. Similarly, relationships such as equations also exist in two forms: a template attached to the frame template, and a specific instance in which variable names are replaced by pointers to the variable instances.

Actions

All actions in the system are initiated by the user. Three basic user actions are possible: (1) The user may enter a value for an attribute, (2) the user may add a new subsystem to the design, or (3) the user may request a view of the current state of the design. When the user enters a new value, the effects of that change throughout the system are handled by the **propagator**.

The Propagator

When a new value is entered by the user, that value is tested against the limits imposed by the attribute template. If acceptable, the new value is passed to the relationships that are attached to that slot. Equations may be solved for any unknown; however, equations are reorganized and solved for a particular unknown only when all other unknowns have acquired values. Once an equation is solved for a given variable, subsequent changes in the independent parameters are automatically propagated through the equation to change the derived value. The derived value may be altered by the user in two ways. First, one of the independent values may be set to **UNKNOWN**; this causes the solution of the equation to be retracted and the equation to revert to a state of having two unknown values. In that state, the assertion of a value for the previously derived parameter causes the equation to be solved for the retracted value.

A second means of setting the value of a derived parameter is available under the tradeoff view. The system of equations in the model is repeatedly solved for differing values of an independent parameter. The resultant values of the dependent parameter are collected and displayed as an *x,y* graph showing how the independent parameter varies with the dependent parameter. The user may then use the mouse to select a value of the independent parameter that results in the desired value of the derived parameter. We prefer this latter approach because it preserves the causal ordering of derivation. That is, we do not say that the cost of a spacecraft was determined by the cost of a subsystem, but rather that the spacecraft's cost was determined by the weight of that subsystem, whose weight was in turn selected to give a specific cost [Simon 84].

Views

VIP contains a set of tools that allow users to visualize data. These tools provide the user with multiple **views** of a complex design. Different tools are available at the design, subsystem, and attribute levels of the hierarchy. Tools at the design level allow the user to visualize an overview of the components tree (Figure 3). Clicking on any subsystem displays that subsystem.

Attribute-level tools allow users to view the properties of individual attributes. These tools are accessed by clicking on the name of the attribute in the subsystem window (Figure 4). Where the views are appropriate to the current value of the slot, a menu of possible views is then displayed.

Choices Available for All Slots

• **Information**: This choice gives the definition of the item, tells how it was derived (if a value is known), and displays all the slots that depend on that value as well as all slots used to compute the value. In addition, any annotations are displayed. Information is displayed in a locked, scrollable text window.

• **Annotate:** This choice allows the user to view and edit annotation related to a given value. This is usually used to allow the user to add notes justifying his decision.

• **How:** If the item is set, this feature causes the system to tell how it is set, or, if unset, how it might be set; if unset, this choice lists all rules and equations that could be used to derive the item; if none exist, this choice tells the user that he must input a value for the item, as there is no other way to derive one. Note that many of the equations in the system normally use some items as inputs only, and thus may be inappropriate for deriving these items. For example, the cost of a satellite is normally proportional to its weight. It is rarely useful to suggest to the user that the satellite's weight could be estimated if the cost were known.

Choices Available Only When Appropriate

• **Sensitivity**: This choice displays the sensitivity of a derived value to all parameters used in its derivation. Each of the deriving parameters is varied by 5% of its current value and the resultant effects on the dependent parameter are computed as numerical derivatives. The derivatives are normalized as $\delta\log(y)/\delta\log(x)$ [Landauer].

•**Tradeoff:** This choice allows users to visualize the effects on a derived value of altering one parameter. Users select a parameter from a list of parameters used to derive the current value. The system automatically varies the selected parameter over a selected range and generates a graph showing the effects on the dependent parameter. Users may select on the dependency graph a point that causes that combination to become the new current values.

• **Recommend:** If a value is unknown but VIP can find a collection of equations based on known parameters designated "usually user entered" or on values derived from the above, this choice displays the potential dependency tree, highlighting the unknown values and allowing the user to enter the remaining unknowns.

• **Apportion:** If the value is the sum of N terms, this choice displays the relative importance of the additive terms as a pie chart.

Interface

A major feature of the philosophy underlying VIP is to enable users to interact with the system in a simple and intuitive fashion. The **interface** consists of a collection of windows; in the current implementation, only one window is displayed at a time. Some of these windows are described below. The two basic windows are the **subsystem**

The Vehicles project has the long-range goal of developing new capabilities and knowledge representations for supporting the conceptual design of spacecraft. VIP has the more immediate goal of utilizing methods developed in this project to provide an intelligent system that allows users to develop and manipulate space-system designs. To make VIP widely available, we developed it for use on a personal workstation (the Apple Macintosh II). This project has been concerned with developing a system that is simple and user-friendly enough to shift much of the use and extension of the knowledge base from the computer science lab to the working engineer.

VIP Architecture

The architecture of VIP follows a common practice in expert-system design, by making a clear separation between the knowledge in the system and the procedures used to manipulate that knowledge. VIP is divided into two basic parts -- an engine for manipulating knowledge and user selections, and a collection of relevant knowledge sources.

Figure 1 displays the architecture of VIP. The seven major components of the engine are as follows:

- **Knowledge Base:** This stores general knowledge about the systems the user is developing. Data from the knowledge base (currently stored as a series of textual files) are parsed by the system into working code. Storing knowledge as text allows the user to interact with easily understandable forms of the knowledge base, facilitating the reading, editing, and correction of the data. The generated code is used by the system but is not normally accessed by either users or developers. The knowledge base is described in detail below.

- **Knowledge Editor:** This tool allows the knowledge base to be updated. Users may add new subsystems, rules, or equations to the database. Knowledge that is added is tested for syntax and consistency with the current contents of the database. The editor tags each piece of information with the name of the person entering the knowledge, and encourages users to attach annotation detailing the technical underpinnings of each item.

- **Working Database:** This stores current designs. Designs are represented as frames that have slots containing information about parameter values, information about component hierarchies, information about how parameter values were derived, and user-supplied comments Designs in the working database may be created and manipulated by the user. A design (for example, of a proposed spacecraft) may be stored in the knowledge base, permitting future manipulation.

- **Propagator:** This tool accepts changes in parameter values. Any change in a parameter's value is propagated through all constraints, rules, and equations in the system.

- **Report Generator:** The tool generates a report on the current design. The report is user-readable text formatted to allow VIP to read in the report, regenerating the working design.

• **Interface:** VIP has a built-in user interface that allows users to browse and alter information rapidly. The interface is described in detail below.

• **Toolbox:** A major component of the interface is the toolbox, which gives the user a number of views into the data. The toolbox also allows users to read, add, alter, or annotate any portion of the design.

Knowledge Base

The knowledge base stores four classes of information: templates, fixed systems, historical designs, and working designs. These are described below.

Templates hold knowledge about how to generate specific elements of an object, storing what is possible to know about a specific subsystem. Such information includes possible parent systems, possible component systems, attribute names, equations, rules, and constraints. Attribute templates store the type of attribute (if numeric), the type of measurement (e.g. distance), and default units (e.g. km). The attribute template may also store default values and upper and lower bounds. If an attribute involves a choice (e.g. a choice of material), then the template holds all possible choices.

Fixed systems are instances of subsystems. As such the attributes are assigned specific values. Fixed systems represent knowledge that may not be altered in the current design. A communications band such as X-band would be considered as a fixed system that holds the assignments for uplink and downlink frequencies, bandwidths, and the effects of adverse weather on the signal. The choice of launch vehicle is another class of fixed system that contains cost, availability, possible orbits, and throw weight.

Historical systems comprise data from complete systems that have already been developed. Like fixed systems, they comprise read-only data. Historical systems may be accessed to give the user perspective about what is possible. The system can access the historical system most similar to the current design and, in the absence of other information, can use parameter values from this system as defaults in the current design.

Working designs are also complete systems. Unlike historical systems, working designs may be modified.

The knowledge base is stored as a collection of textual files that hold information as keyword, value pairs. Figure 2 shows a portion of the current knowledge base describing a power subsystem. The current system parses these text files into Prolog code; the files are then added to the working database as required.

Working Database

A design such as a spacecraft or space system may be considered as a hierarchical frame system. The design is a root frame whose slots contain specific values called **attributes** and frames called **components**. The component hierarchy may be visualized as a tree (Figure 2). The design and all its components are a class of frame called a **subsystem.** In addition to components and attributes, a subsystem has slots for relationships: equations, rules, and constraints. All attributes and relationships are attached to a specific subsystem frame. Relationships can link to attributes in other frames; these links allow information to propagate through the design.

window, which gives a detailed view of a specific subsystem, and the **navigation window**, which shows all subsystems as parts of a tree.

• **Subsystem Window:** The subsystem window (Figure 4) displays all the slot values in a scrolling field. Each slot has four components: **Name, Value, Units** (if applicable), and **Source**. The name is simply the name of the slot. Values may be strings, numbers, or tuples. Every slot type points to code that generates an appropriate display of the slot's contents. If these contents are undefined, a blank rectangle is displayed. Numeric slots may have units associated with the slot's value. If these units are defined, they are displayed next to the data. The source type is displayed as an icon at the right of the slot. Currently, we support icons for **user, default, equation, rule,** and **system** (copied from a predefined system, such as the throw weight of the shuttle).

Navigation Window

The navigation window (Figure 3) displays the components as a tree, with the design at the root, various subsystems as the branches, and the components as the leaves. The user may open a selected subsystem by clicking on the appropriate box. The tree gives the user a sense of where the current system lies relative to the entire design. This window enables the user to develop insight into the architecture of the design under consideration. The navigation window also enables the user to move throughout the design.

Dependency Window

The dependency window displays trees that are generated for an individual parameter. Two possible trees can be derived. When the value of a parameter is derived, the dependency tree shows the current dependencies; for example, cost was derived from weight and availability, weight was derived from power and coverage, and coverage was entered by the user. The leaves in the dependency tree represent default values as well as values entered by the user. Dependency trees may be constructed for any derived parameter. When a parameter has no value, a potential dependency tree can be generated by seeking parameters that are noted as "usually user entered."

Other Windows

Other windows are generated to display graphical information, presented as an *x,y* graph. The relative contribution of terms to a sum is presented as a pie graph, and the sensitivity of a dependent parameter is presented along with its dependencies. Here the derivatives are normalized as described above and are presented as a bar graph.

Hardware

The system is implemented on an Apple Macintosh II with 5 MB of memory and using AAIS Prolog. A version runs on the Sun/3 under Quintus Prolog, but this version does not have the full user interface that is available on the Mac. The system requires approximately 2.5 MB of working memory.

Problems Studied

We have applied the system to a number of conceptual studies within the corporation. For example, we have developed models for electrical power systems,

constellations of communications, and configurations for SDI-related designs. The most complex models were able to handle 20 subsystems and approximately 100 equations.

In several cases the models were generated by engineers working directly with the system, rather than by personnel within the Computer Science Laboratory. The success of the system with relatively naive users was encouraging. The problems these people commonly encountered (misspelling the names of system parameters or referring to parameters not explicitly imported from the owning subsystem) have highlighted the need to provide more tools to verify the input data, both syntactically and in terms on the internal consistency of the generated model.

References

Bellman, K. and A. Gillam, "A knowledge-based approach to the conceptual design of space systems," *AI Papers 20, Proceedings of the Conference on AI and Simulation, 1988* , SCS International.

Landauer, C., "Sensitivity Analysis," Technical Report, The Aerospace Corporation (in process).

Iwasaki, Y. and H. Simon, "Causality in Device Behavior," *Artificial Intelligence 29:3-32* (1984).

Parsaye, K., M. Chignell, S. Khoshafian, and H. Wong, *Intelligent Databases* (Wiley, New York, 1989).

Figure Captions

Figure 1. VIP Architecture

Figure 2. Navigation Window. The components of a design may be viewed as a tree. The system highlights the currently active system. When this window is displayed, clicking on any box causes that system to become active and displays a subsystem window for that subsystem. This enable the user to change the active subsystem.

Figure 3. Subsystem Window. The basic view of a subsystem, this window allows a user to view and modify specific attributes. The Traverse button brings up the navigation window. Clicking on the name of any attribute (e.g. Time_Delay) pops up a menu of applicable tools for that attribute. Clicking on the value box allows the user to type in a new value. If the attribute has only a fixed set of alphanumeric values, clicking on the box pops up a menu of choices. The icon indicates the source, and clicking on the icon displays information about how the parameter was derived.

Figure 4. Knowledge Base. This section of the knowledge base illustrates how knowledge is entered as keyword text. A major goal of VIP was to keep the working knowledge base in a form that can be accessed and edited by the user.

Figure 1

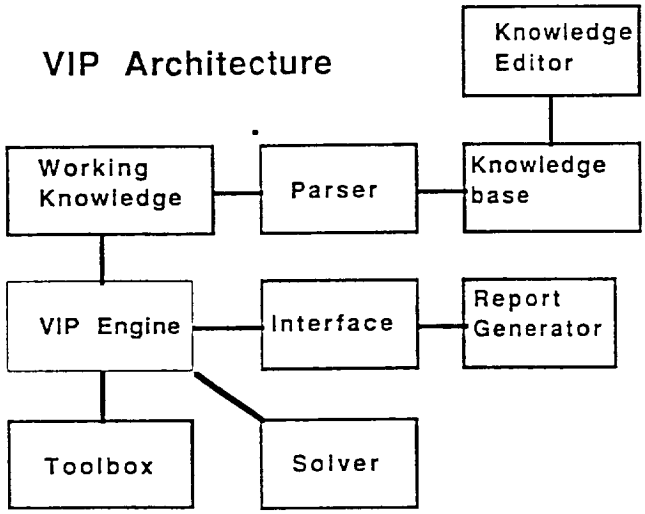

Figure 2
Sample System Specification File

Note sections in ***bold italics*** are the designator. Remaining text is merely commentary.

```
% ***************************************
% power
%
% Begin Subsystem Specification
%
subsystem  power
  possible parents buss  % must be a subsystem of the buss
  % designate possible components
    possible  components  regulator  solar_panel  power_buss

% ************************************************
 %
 %  Begin Designating Attributes
 %
 % ************************************************
  attribute  type      choice solar_panel solar_body nuclear solar_concentrator
                default  solar_panel

 %
 %  Specify a numeric attribute
 %
 %   name     type  units metric english

  attribute  eol_power numeric power watts watts
             lower  limit  100  watts
             upper  limit  5000  watts
             what  "  eol_power is the end of life power. That is the power the power
             system can deliver at the end of the design life of the spacecraft. Because
             components in the system, specifically solar cells degrade over the
             spacecraft lifetime, this will be significantly lower then the bol_power or
             beginning of life power. Power systems are usually designed to deliver a
             specific end of life power.  "


attribute battery  choice nicad nih lithium lead_acid
```

Figure 3 Navigation Window

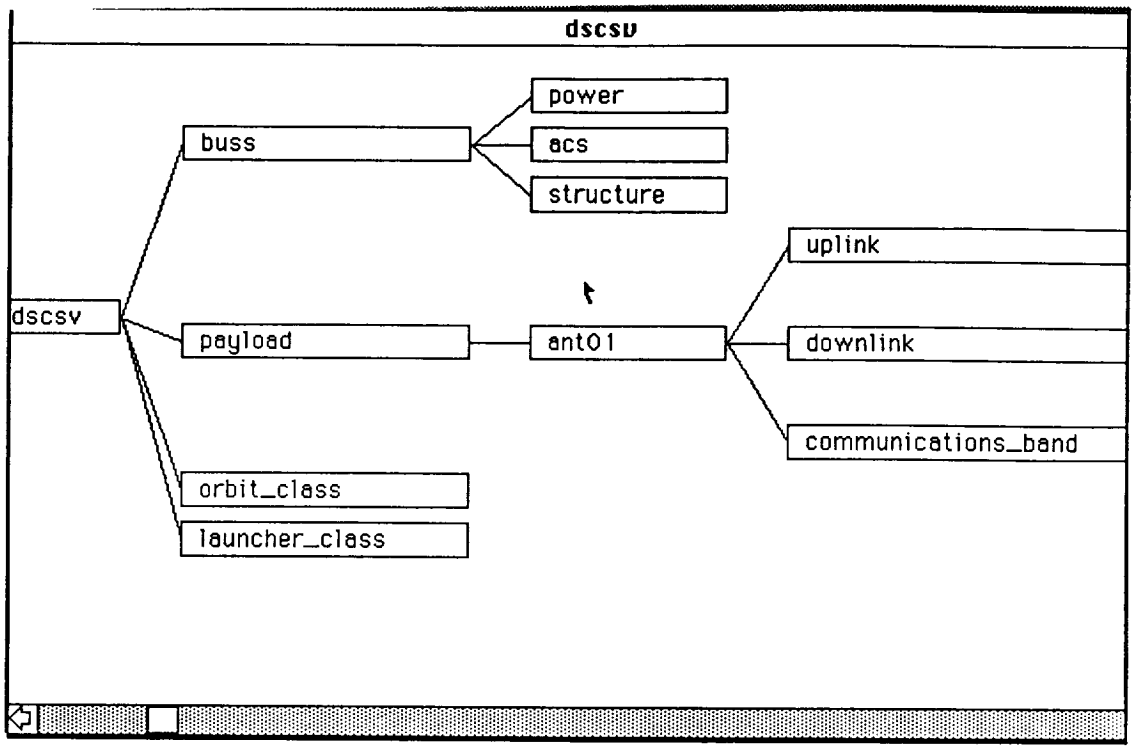

Figure 4 Subsystem Window