JOURNAL OF SPACECRAFT AND ROCKETS
Vol. 30, No. 3, May–June 1993

Vehicles Knowledge-Based Design Environment

April Gillam*
The Aerospace Corporation, Los Angeles, California 90009

The *Vehicles* design environment is a knowledge-based system that assists designers in studying tradeoffs, evaluating alternative designs, and identifying design *drivers*. The environment is composed of 1) analysis tools, such as parametric analysis and what-if analysis capabilities; 2) subsystem sizing and performance models; 3) a historical data base (of satellites that have already been built); and 4) an extensible architecture in which new models, tools, and design concepts may be added. We have found that building in flexibility, both for the designer to explore a variety of space system design solutions as well as for the software developers to continually enhance *Vehicles'* capabilities is essential. The analysis tools, which have been very well received by the design community, include the capability to study parametric sensitivities, trace the derivation of results, and compare multiple design concepts. An integrated design environment, one that offers a variety of analysis and information handling tools, gives the designers the ability to view a design from different perspectives and across functional, organizational, and other boundaries.

Requirements

DECISIONS made early in the conceptual design phase of space systems often lock in much of the cost and future allocation of resources. To create spacecraft architectures that cost less and have better performance, it is important to develop design tools that are capable of in-depth evaluation and optimization.

The difficulties in designing artifacts as complex as space systems are well known. To name just a few, 1) the design is created by people with expertise in different areas and with different goals; 2) the models, data, and designs created by the design team must be integrated into a single design concept; 3) there are many competing factors (such as performance requirements vs budgetary constraints); and 4) designs are often context dependent, with the context including operating environments, mission scenarios, threats, political and fiscal realities, and the policies of regulatory agencies.

In addition, concurrent engineering, during the design phase, necessitates that design take into account a growing number of disciplines, requirements and constraints, and tradeoffs. Thus a design environment that meets these needs must support the work of subsystem designers, space system planners, evaluators of competitive designs, and developers of models that characterize subsystems. Increasingly the experience of manufacturing, system maintenance, and marketing also needs to be integrated.

At the heart of a design environment is the need to provide insight into tradeoffs, to identify the effects of decisions so that alternative designs and design drivers can be evaluated, and to study multiple designs within the entire solution space—not just a single solution. Improving insight and understanding involves viewing a design from different perspectives,[1] clarifying correspondences, and identifying interdependencies that might not be obvious. Because studying tradeoffs from many different viewpoints is essential, a systems design approach must look at the entire design and its interfaces, as well as study the design of the individual subsystems. Tracing the impact of decisions throughout an entire design often uncovers unforeseen effects. This is especially so in optimizing a design, where local optimizations do not guarantee a globally optimal solution.[2]

Along with these analysis tools, a design environment will be more effective if it provides a repository of engineering models, existing designs, and contextual information about each design. Without this, work that is done early in a project may not be usable when one returns to it, months or years later. For example, without the context, i.e., the models and assumptions underlying the design decisions, the work may not be credible, understood, or usable as a basis from which to make further studies. In that case, the work may need to be duplicated, starting from the beginning again.

These issues need to be addressed to provide designers with an environment that is sufficiently powerful so that the design process is focused on the creative aspects and less on 1) data management, 2) recasting the design problem to be something the existing design software can handle, and 3) searching for information located in many different places.

Related Research

Chandrasekaran and Brown[3] identify many levels of design, ranging from the fairly routine and well-specified process of making modifications to an existing design to the innovative creation of a totally new design. To date, most work has focused on automating and facilitating the more routine aspects of design. In the case of conceptual design, as in *Vehicles*, a number of challenges arise because the process is a highly creative and innovative one. A challenge arises from manipulating and combining information appropriate to different design levels, such as the functional, structural, and behavioral levels, and from using information specific to models that may be based on different assumptions. This is not as much of an issue for redesign work, since the models are already known to be consistent; however, the conceptual design stage may be the first time that the models and information are integrated.[4] Another issue is that existing design aids are often too rigid to allow alternative design styles. The designer must conform to the style of the software design tool, rather than the software adapting to the strategy taken by the designer.

Design brings many diverse factors into play. Heuristics, detailed calculations, interpolations, precalculated tables, existing programs, sets of equations, constraints, and requirements are all used in the design process. Although models of spacecraft subsystems must draw upon these diverse sources of information, the dynamic nature of the information makes it difficult to represent and maintain the knowledge. For example, the models go through an evolution as the information on which they are based changes, as requirements and capabilities of the artifact being modeled change, and as the models

Received Nov. 14, 1991; revision received July 12, 1992; accepted for publication July 30, 1992.

*Member of the Technical Staff, Computer Systems Research Dept., M/S M1-102, P.O. Box 92957.

themselves are refined. The relationships between the models and the knowledge associated with them evolve as well. It is important, therefore, to track this evolution, to maintain consistency, credibility, and traceability of the information upon which the analysis is based.

Chalfan[5] describes work at Boeing on the analysis of aerospace vehicle design. Chalfan's approach has been to integrate existing programs by explicitly capturing "problem-solving knowledge required to perform the integration" in an expert system. Similarly, we are gathering domain knowledge for the subsystems in addition to knowledge on integrating the subsystem designs. One main problem-solving strategy is to perturb parameters and view the repercussions. Both Chalfan's system and *Vehicles* take this approach. In addition, we find that by using the equations symbolically and having the domain knowledge at hand, we can vary or keep fixed any of the parameters, which is not possible when using code for which the only handle is on the input parameter list.

Buckley et al.[6] have built *Design Sheet*, a tool to support trade studies during conceptual design. They work with algebraic equations and provide tradeoff, optimization, and other analysis capabilities. *Vehicles* has a similar emphasis, representing the equations so they may be manipulated and recast with different sets of independent and dependent parameters. As a foundation for trade studies, this is essential. *Design Sheet* also addresses the difficult task of error propagation and has a constrained optimization capability. A difficulty faced by optimization algorithms, in general, is that the equations may be nonlinear and the values may be discrete, so there is no assurance of differentiability or convergence.

Bouchard[7] has built a powerful design environment, *Engineer's Scratch Pad* (or ESP), not only to create designs, but also to give instructions on how a design that can be used in fabrication is built. While taking a much more process-oriented approach than has been taken in *Vehicles*, both systems focus on how to handle the complexity involved in designing large systems. Strategies include 1) providing a variety of analysis and reporting tools to explore the design space, 2) capturing knowledge about designs and the design process, and 3) handling data and knowledge for ease of access. ESP also provides a number of optimization algorithms.

The methodology used to design and construct an environment for the design of space systems has been developed to address a number of challenging issues. Prominent among these issues are the following: 1) the users have varied design and analysis strategies as well as different viewpoints; 2) additional capability (analysis tools, handling of new types of information) will continually be added throughout the lifetime of the software environment; 3) the information in the environment and the derived results must be credible (accountability); and 4) information from different sources must be integrated to work together and be accessible by analysis tools as well as by the users.

Methodology

The methodology used to address these issues is based on the following:

1) A knowledge-based paradigm for representing and using information in a variety of situations (scenarios, trade studies, simulations, what-ifs).

2) An extensible architecture, which makes it possible to add new tools and new types of information.

3) The availability of analysis tools at any stage or level of a design.

4) The option to start either with high-level requirements and develop a detailed design, or with a new technology or an existing design and analyze new capabilities.

The knowledge-based paradigm used in *Vehicles* includes concepts such as the explicit representation of information, the separation of the knowledge and data from the actions (inferencing and solving), and the declaration of the relationships among information. Explicitly storing the information makes it possible to browse all of the information. This is important when several models by different authors are used together to characterize a satellite. Rarely will any one person be expert in all fields, making it even more important to provide access to the models, their assumptions, and their authors. This approach makes the conglomerate system model more credible, transparent, and accessible. The user can also follow decisions through to their conclusions, or vice versa, trace conclusions to the assumptions and decisions on which they are based.

Another challenging aspect of supporting the design process is that the knowledge and data bases are not static; the information becomes obsolete, is superseded, or may be appropriate only in a particular situation. This makes it important to gather contextual information[8] about the design data and knowledge (which includes the types just mentioned), as well as about constraints and conditions for their use.

Integrating the many different kinds of information is a challenge, because of the different perspectives of the designers, planners, and analysts, as well as the different sources of the engineering models, data bases, and analysis tools. The models may have different levels of detail, different variable names that mean the same thing (or a single variable name that means something different in each model in which it is used), equations based on different assumptions, or different degrees of calculation precision. Most of the integration has been done by hand, which involves working with the authors of the models, defining terms, and specifying assumptions and constraints. We are investigating how the burden on the user might be reduced.[9]

Part of the solution to integrating this information is to declare explicitly the interdependencies in the models. Thus, when a model is changed, the repercussions on other models are evident. It also makes the person who is changing the model aware that changing certain parameters could lead to inconsistencies.

There is a need for extensibility because the software environment will evolve and will require enhancements as new analysis tools and data sets are required. The architecture of the environment must support this growth, while providing analysis capabilities in the short term. The ability of the architecture to accommodate new kinds of information is possible through the use of a "blackboard" architecture, which provides a globally accessible forum to track information that is relevant to a project, the status of the design and analysis studies, and the interdependencies of information that has been partitioned (e.g., into subsystem models). This architecture has been implemented by means of system definition statements (which are predicates within Prolog). The new information may either be contained entirely within the system definition statement itself, or it may be accessed by means of a pointer. The latter method often saves storage space and simplifies the specification of the information.

The system definition is composed of the following: 1) System Name; 2) Subsystem; 3) Type of Information; 4) Values or Pointers; and 5) Version and Date.

The first field is the System Name, such as *landsat* or *dscs*. The second field, Subsystem, is the breakout of the system segments. This might include a *space* and a *ground* segment. The space segment might be further described by satellite subsystems, such as *bus* and *payload*. A similar breakout can be seen in Fig. 1, where both the *bus* and the *payload* have additional breakouts. The Type of Information, given as the third field in the above list, is the category of information, such as *subsystems, parameters*, *equations*, *dependencies*, and so on. The fourth field, Values or Pointers, describes the information. If the type of information were *dependencies*, the Values field would contain parameters that are not generated in the subsystem being described. For instance, in a *radar* subsystem, the *dwell time* is dependent on the *slant range*, which in turn is calculated in the *orbit* subsystem. The last field in the list contains the Version and Date, which is neces-

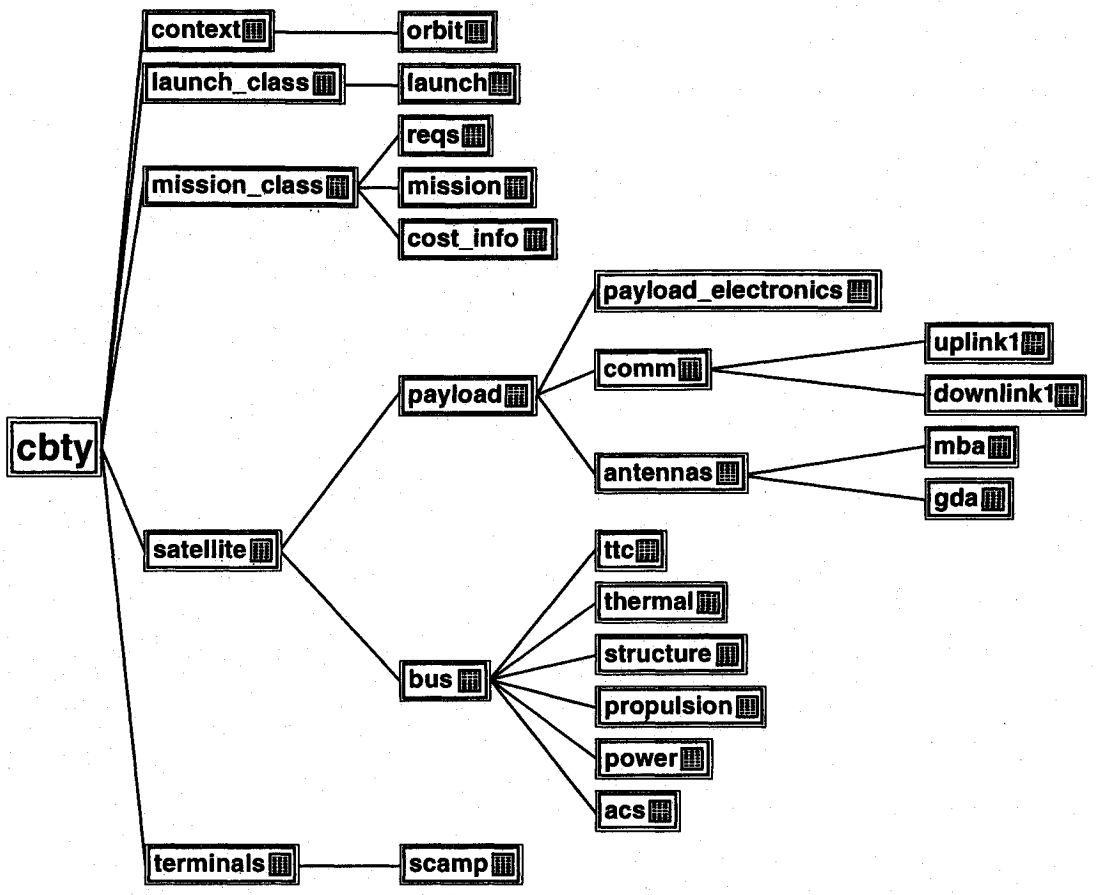

Fig. 1 An overview of the subsystems studied in a communications satellite concept.

sary to maintain consistent models, especially since they may frequently change.

Additional information, even new types, is incorporated simply by adding new system definitions, or by creating new "types" if the information is new. Analysis tools and new reports may be added by writing a preprocessor or postprocessor that accesses the information to input into the tool or incorporates the output of the tool into the appropriate data structures. This is mediated by the system definition statements, which have routines that assist in retrieving or saving the information.

With design paths that may begin (or end) with requirements, a flexible manner of handling equations was sought. We have used a symbolic equation solver that stores the equations symbolically. Then, depending on which variables are known, the equations are solved for any unknown variables. Thus, there is no longer a distinction between the variables on the right- or left-hand side of the equations. The same equations may be used to solve for any variable, depending on the initial conditions.

As noted earlier, the architecture of the software support system, like that of the space system being designed, must permit multiple experts to create, modify, and extend the architecture itself. Substantially different approaches, viewpoints, and information bases necessitate that the architecture address these difficult design problems. To this end, an open and extensible architecture has been investigated.[10] This is being addressed by 1) modularizing the code; 2) separating the knowledge and data bases from the analysis and synthesis tools; 3) exploring schemes to "wrap" (or characterize) analysis tools with metaknowledge about the tool's usage, its inputs, and its outputs; 4) representing information about the structure and interdependencies of design components explicitly; and 5) providing access to programs written in other languages.

Capabilities

We have built a software environment to rapidly model the technology, performance, architectures, and context of artifacts being designed or studied. The artifacts we have dealt with are satellites, constellations of satellites, and satellite subsystems; however, *Vehicles* can support the design of any artifact that can be described as a hierarchy of subsystems. This could include automobiles, printers, or launch vehicles. Figure 2 shows the tools, reports, and kinds of information that are in *Vehicles*. Recently, we have begun to use *Vehicles* as an environment in which to integrate existing software programs into a consolidated simulation.

Analysis Tools

A number of tools have been developed and others are being added to *Vehicles* in order to increase the designer's ability to study the designs, their performance, and their behavior. Among these tools are the following:

1) Equation solvers, allowing the set of independent and dependent variables to be changed, depending on the case being studied.

2) Sensitivity analysis, which gives the relative importance of independent factors in relation to the dependent variable.

3) A parametric study capability, to vary several independent variables and monitor dependent variables, while selecting those designs that meet requirements, constraints, and criteria set by the user.

4) A subsystem characterization report, a report summarizing the critical factors, a communications power link budget, and a weight-breakdown report (among other reports).

5) A comparison and evaluation of design alternatives using such criteria as comparison to a baseline design, constraint and requirements satisfaction, and sanity checks.

6) Presentation of multivariate information,[11] graphically and in tabular format.

Traceability

Considering the complexity and numbers of people working on a project, it is not sufficient simply to provide the *answer*. It is also necessary to provide insight into how solutions were derived and where the information originated; i.e., the data may be due to an estimate, or a measurement, or be from a reference. Along with the explicit representation of knowledge in a knowledge-based approach, information is stamped with the date, time, and the source (or origin). This makes it possible to trace the development of a design and find the user's decisions, when rules were applied, which equations were used, and which external tools or data bases were used. This information is then used to build a dependency diagram of how a parameter value was derived. For example, Fig. 3 shows that the *power_rcvd_signal*, i.e., the power of the received signal, depends on the frequency, the receive and transmit antenna diameters, the power radiated, *power_avg*, and the orbit. In the figure the parameters are in bold print, underneath those are the subsystems, and beneath the subsystems are the values. The lines connecting the parameters indicate that the parameter on the left was derived using the parameter(s) to the right of it. The methods (e.g., rule, equation, external routine) are not displayed, to avoid cluttering the dependency diagram.

Example Trade Study

A trade study is given to exemplify how the above tools are used and the kinds of information that comes into play. For example, in designing a surveillance satellite, models are developed to size and characterize the performance of the sensor payload, the spacecraft bus (attitude control, electrical power, communications, and so on), the environment, the orbit, and the cost and schedule. One trade study compares a scanning sensor to a staring sensor, monitoring the affect of each design on the signal and data processing, the telemetry bandwidth, the weight of the subsystems, and the relative cost of the different alternatives. This may be done within *Vehicles* as opposed to having each individual engineer analyze their subsystem and then pass that information along to the next engineer. The results are stored within *Vehicles* for further analysis and reporting.

Status

Vehicles is an interactive environment built to enhance, assist, simplify, and expedite design activities under the guidance of the designers. We stress its interactive nature, because the knowledge being captured is sufficiently focused on individual tasks so that a higher-level systems view necessitates having the designer in the loop.

The core of *Vehicles* is written in Quintus Prolog and C on a Sun workstation. A new user interface is being built using X-windows, C++, and the Interviews widget set. We have also linked *Vehicles* to existing software tools (in Fortran and C) for graphics and engineering analysis.

We have developed the ability in *Vehicles* 1) to support varied styles of design, 2) to handle and evaluate multiple

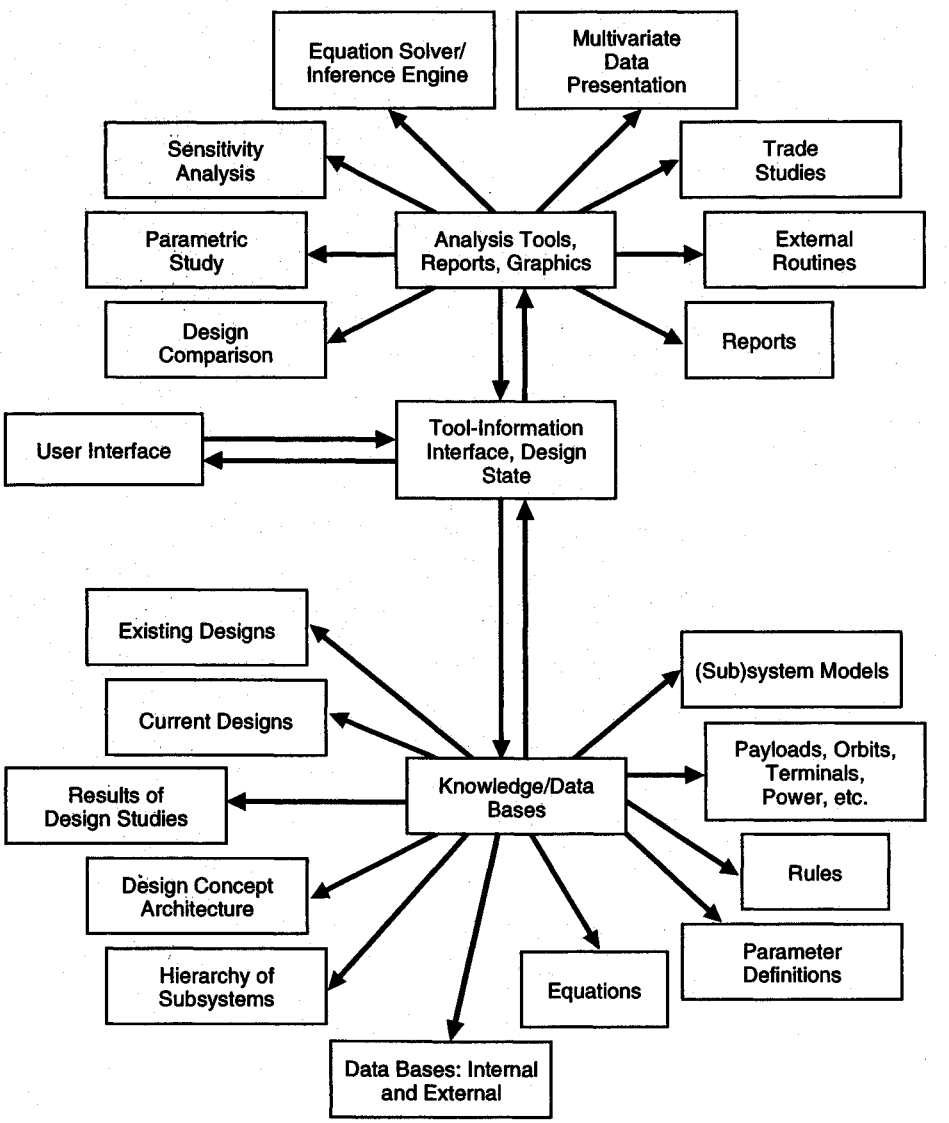

Fig. 2 The *Vehicles* knowledge-based design environment.

designs, 3) to provide meaningful status reports on the results obtained and on the degree of completion of a design, 4) to provide a variety of analysis tools, and 5) to create an open and extensible architecture.

Vehicles has been built and used, in parallel with traditionally coded simulations, to study tradeoffs during the conceptual design phase of several different satellite-architecture studies. The projects we have supported within the last few years include the design of a radar payload, the development of a balanced configuration of satellites to meet mission needs, the design of a modular spacecraft bus that supports interchangeable payloads, the evaluation of designs from competing contractors, and a study modeling the effectiveness of environmental testing of satellites. The system has been delivered to two sites within Aerospace Corporation for use, evaluation, and the iterative process of improving the system based on feedback from its users.

Currently, *Vehicles* has over 300 rules, 900 equations, 2400 parameters, and 150 subsystems. The number of subsystems and parameters are high because the same kind of subsystem, such as "antenna," might have a different name, depending on its type. Thus, some of the antenna subsystems in *Vehicles* are the following: *gda, horn_antenna, mba, mba1, mba2, mba3, scamp,* and *scott*. These represent a gimballed dish antenna, a horn antenna, a few multibeam antennas, and a couple of mobile terminals. Each of these antennas is a separate subsystem characterized by a similar set of parameters. Some parameters include *antenna_type, freq_band, gain,* and *n_beams*. The last parameter, the number of beams, would appear only in the multibeam antenna subsystems. Although the *scamp* and *scott* terminals are based on the ground, they must be modeled in order to characterize the communications links[12] and to study the allocation of resources between the ground stations and the satellite(s).

Lessons Learned

An environment dedicated to supporting designers is powerful because the tools that comprise the environment 1) provide insight into design, 2) enable one to explore the design and solution spaces, 3) integrate diverse engineering models, and 4) capture the design flow, i.e., the decisions made, models used, assumptions, and design approaches. The insight must ensure a better understanding of a design, a set of design alternatives, the tradeoffs, and the design drivers, while keeping in mind the limitations of the modeling process (a model is not the artifact it describes).

The credibility, transparency, and traceability of information are important because the information is complex, originates from a variety of sources, and to be used in ways the authors had not forseen. Engineers, having developed their own analysis programs, know the assumptions and capabilities on which the programs are based, as well as the limitations of the software. As both the information and the analysis tools gain a wider audience, a whole set of meta-information (information about information) is needed to understand how the information (or tool) may be used, when it is appropriate, and what it is based on.

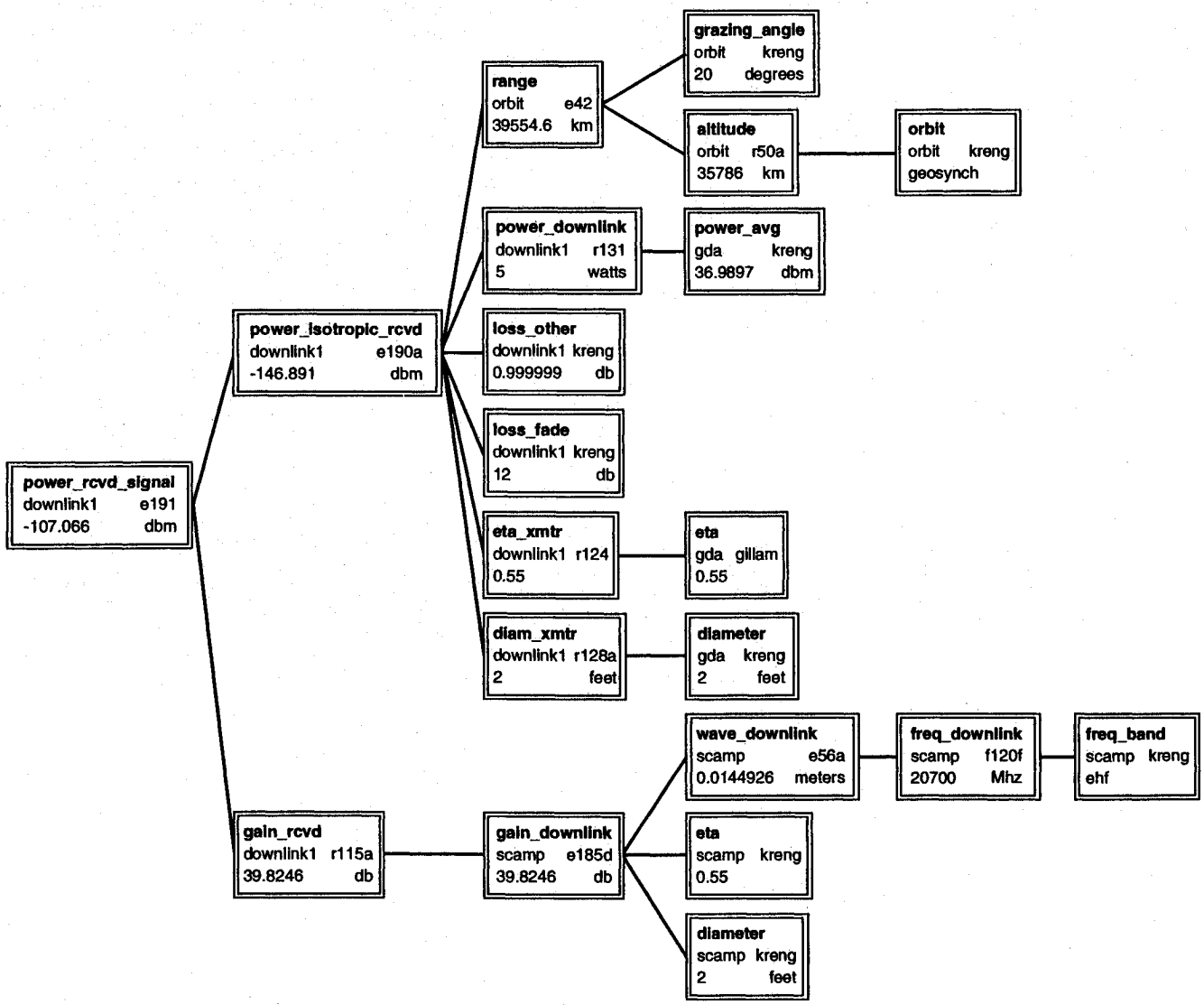

Fig. 3 Traceability (or dependency) diagram for received signal power.

For this reason, as well as to build credibility in the eyes of the designers, it is vital to validate the information and models being used, to enable the user to view what the models are composed of and where the information comes from, as well as to ensure consistency within and across models. The latter issue, that of integrating models[9] that have been authored by different people, is an ongoing area of research.

Information about the design information, such as the sources (e.g., an equation, a report, or a person) or conditions placed on the validity of some information or model, is explicitly captured and represented, and may be displayed. This is coupled with reporting tools to support the transparency and traceability of information. The self-documentation report takes the current state of the design and documents all of the models and information on which that design is based. It includes only the information that is actually relevant to the design. For example, in the design of a phased-array radar, the models appropriate to phased arrays are automatically called into play once the "radar_type" is set to "phased array." Radar models appropriate to a reflector antenna would not be included in the documentation. The second tool, the dependency graph, traces the information that was used to solve for a dependent parameter. This was already presented in Fig. 3.

We have found it essential to involve the users as early as possible, even while the *Vehicles* environment was being developed, so they will feel that they "own" the system; this also ensures that it directly addresses their needs. Building a prototype has made it possible to respond quickly to user needs and get feedback. Often the users themselves did not know what they wanted. However, upon seeing the prototype, it was easier for them to identify what they wanted and to evaluate the current proposed design. The prototype has thus acted as a "strawman," which is how space system designers often begin—initially presenting a preliminary or rough system concept in order to stimulate other ideas. One postscript to this is that the system developers must be open to feedback and to making a variety of changes. This is a very different approach from writing software based on a requirements specification. A high level of trust and open communication channels between the system's users and its designers[13] help ensure that everyone work as a team, rather than as adversaries.

Another lesson we have learned is that it is important to make explicit the interactions among components, subsystems, models, and so on. By explicitly stating what these interactions are, it is possible to trace effects beyond a single module and to perform an impact assessment or trade study. This will also form the basis for a global optimization or satisficing routine to be built. We are actively pursuing the incorporation of such routines.

Engineering designs are always being refined, improved, and expanded. Similarly, the set of design information, analysis capabilities, and kinds of reports needed for design and analysis also change and are enhanced over time. Thus the design platform that is built today will have additional requirements placed on it throughout its lifetime. Supporting this evolution requires that flexibility be built into the design environment, from the perspective of new kinds of engineering models and information as well as from the perspective of software architecture.

Acknowledgments

This work was supported by the Aerospace Sponsored Research Program. The *Vehicles* project has been spearheaded by Kirstie Bellman and the author. I would like to thank C. Christopher Reed and Ronald Johnson for supporting our design environment with funding and guidance, and Frank Hennessey, Jack Kreng, John Maul, and Scott Szogas for their patiently explaining how they design and model space systems. Other researchers at the Aerospace Corporation who have applied their many talents in designing and building the *Vehicles* environment are Grace Chen-Ellis and Robert Lindell. The user interface has been built by Lawrence Miller, Michael O'Brien, Rami Razouk, and Sam Shen.

References

[1]Lendaris, G. G., "On Systemness and the Problem Solver: Tutorial Comments," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 16, No. 4, 1986, pp. 603–610.

[2]Wilde, D. J., *Globally Optimal Design*, Wiley, New York, 1978, pp. 165–200.

[3]Brown, D. C., *Expert Systems for Design Problem-Solving Using Design Refinement with Plan Selection and Redesign*, Ph.D. Dissertation, Ohio State Univ., Columbus, OH, 1984.

[4]Steinberg, L., "Design = Top Down Refinement Plus Constraint Propagation Plus What," *Proceedings of the IEEE Systems Man and Cybernetics Conference* (Fairfax, VA), Oct. 1987, pp. 498–502.

[5]Chalfan, K. M., "A Knowledge System that Integrates Heterogeneous Software for a Design Application," *The AI Magazine*, Summer 1986, pp. 80–84.

[6]Fertig, K. W., Buckley, M. J., and Smith, D. E., "Design Sheet: An Environment for Facilitating Flexible Trade Studies During Conceptual Design." AIAA Paper 92-1191, 1992.

[7]Bouchard, E. E., "Concepts for a Future Aircraft Design Environment," AIAA Paper 92-1188, 1992.

[8]Forsythe, D. A., and Buchanan, B. G., "Knowledge Acquisition for Expert Systems: Some Pitfalls and Suggestions," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 19, No. 3, May/June 1989, pp. 435–442.

[9]Walter, D. O., and Bellman, K., "Some Issues in Model Integration," *Proceedings of the 1990 Eastern MultiConference*, Vol. 22, April 1990, pp. 249–254.

[10]Bellman, K., and Gillam, A., "Achieving Openness and Flexibility in Vehicles," *Proceedings of the 1990 Eastern MultiConference*, Society for Computer Simulation, San Diego, CA, April 1990.

[11]Inselberg, A., and Dimsdale, B., *Human Computer Interaction*, Plenum, New York, 1990.

[12]Sklar, B., *Digital Communications,* Prentice-Hall, Englewood Cliffs, NJ, 1988, Chap. 4.

[13]Rechtin, B., *Systems Architecting-Creating and Building Complex Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1991, Chap. 3.

Alfred L. Vampola
Associate Editor