

# A Knowledge System that Integrates Heterogeneous Software for a Design Application

Kathryn M. Chalfan

*Artificial Intelligence Center, Boeing Computer Services, Seattle, Washington 98124*

In the aerospace industry, knowledge is frequently encoded in various procedural programming languages. These programs typically perform computational functions such as simulation modeling; dynamic analyzing; and optimizing in support of the preliminary design, the detailed design, and the test. Design analysis of the product requires that these computer programs be integrated in a specific sequence in terms of their input and output data and order of execution. Because of the complexity of the interrelationships among the programs, numerous delays and errors occur during their integration. These delays and errors can increase costs, cause scheduling crises, and reduce design quality. However, the problem-solving knowledge required to perform the integration function can be formalized in an expert system that "understands" the objectives of the analyst and executes all programs necessary to produce the desired design analysis. This article describes the expert executive for preliminary design, which was developed at The Boeing Company to expedite the design analysis of aerospace vehicles.

## The Problem

Within the domain of preliminary design of aerospace vehicles, four codes form the basis of analysis: weight, aerodynamics, propulsion, and performance. Figure 1 depicts a typical configuration of these codes for one specific application. The bold rectangles represent technology codes, and the light ones represent input and output variables,

---

I wish to acknowledge the inspiration of Dr Janusz Kowalik of Boeing, who first suggested this research topic and provided continual encouragement in the face of technical difficulties. Dr Alan Mitchell of Boeing provided project management and domain expertise and encouragement. Dr Antonio Elias of the MIT Flight Transportation Lab introduced me to the paper airplane approach to symbolic manipulation of design equations. Steve White, now of Asymetrix, Phil Harrison and Tom Skillman, both of Boeing, and Kevin Layer of Franz Inc all provided invaluable technical assistance. Dr Douglas Dorrough and Caroline Fu of Boeing provided management commitment.

as indicated by the arrows. For example, *w1* is an output of the weight technology code and also an input to the aero technology code. Some input variables, such as *v*, are not output by any of the technology codes in the given configuration. These are known as free-design parameters.

From one application to the next, variations occur in the versions of the codes to be used and in the input-output relationships among them. A typical task might be to compute the range of an aerospace vehicle being designed for a tactical application. Because the performance code requires inputs from the propulsion and aero codes, *propulsion* and *aero*, as configured for tactical applications, must be run first. In turn, before the aero code can be executed, the weight code must be run. When the range is finally obtained, the cycle begins again, based on perturbations of free-design parameters. If a new application, such as a space vehicle, is considered, the technology codes are reconfigured, with possible additions or deletions of variables.

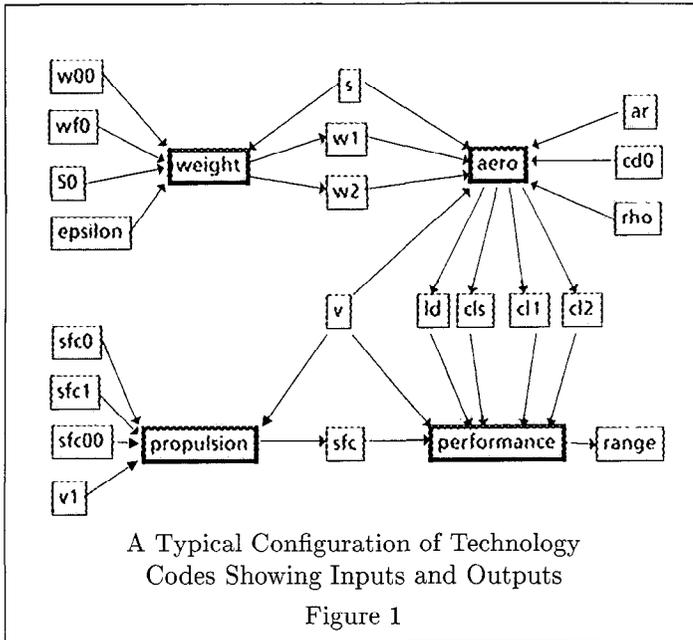
Each program is "owned" and validated by a *technology group*, such as weights technology or propulsion technology. In the absence of an automated integration program, technology groups interact to identify the appropriate technology codes to run, obtain input values

---

## Abstract

We have implemented a knowledge system that integrates the many computational programs (technology codes) Boeing aerospace vehicle designers use, thereby expediting design analysis. Because this system separates facts about attributes of the current set of technology codes from general knowledge about running the codes, those who maintain the system can keep it continuously up to date at low cost. In contrast, three conventional approaches failed because they could not be maintained easily.

---



from one another, run the codes, and analyze the results. The computed values are manually extracted from code outputs and inserted, possibly erroneously, as input values for other codes. Because the typical design cycle requires several weeks, and multiple design concepts can be generated in parallel, technology analysts might be uncertain about which version of which design they are addressing. Because of the time required for the paper output to be analyzed and the appropriate values to be passed to other groups and the scheduling problems for such a complex activity, often only one or two designs can be considered. It is then very difficult to justify the selection of one design over all other possible designs. Because of the high cost of the preliminary design process with respect to the quality and quantity of designs produced, the need for an integrated approach to preliminary design has long been recognized within the industry.

### Previous Approaches

Recognition of the problem did not lead directly to the development of an integrated preliminary design tool. On the contrary, a number of investigations into possible approaches to this problem led to the belief that a feasible solution might not be possible.

First, a procedural program that subsumed several technology codes was built. This program required changes to the technology codes. It was not maintainable because the individual groups continued to alter their technology code programs separately.

Second, a database management system was tried. The system featured an extensive library of software primitives to perform various functions such as unit conversion. It too required changes to the technology codes and was not maintainable. In addition, the procedural changes to

the software primitives could not keep up with the changes to the technology codes.

The third approach left the technology codes untouched and built a procedural program that initiated separate, independent processes consisting of the technology codes communicating through a common database. This was better because the technology organizations continued to maintain technical and managerial control over their codes. The rigid procedural integration program was still unacceptably costly to modify, requiring a flow time of approximately six weeks. However, it did provide a prototype and baseline for the knowledge system.

### The Knowledge System

Symbolic computing presented a new perspective to this problem. We hypothesized that if the general knowledge of how to run the technology codes and compute the values of design variables could be codified, then the specifics of the problem could be addressed relatively inexpensively, thereby providing a viable integration capability. Elias (1986) had already addressed the symbolic computation of design variables in the Paper Airplane project. In our approach, however, the numerical computations were required to be performed in external procedural programs rather than in Lisp functions. To investigate the feasibility of integrating the symbolic and numeric computation, we developed a preliminary design tool with an expert executive that contained symbolic knowledge of how to execute a set of computational programs. The expert executive assumes the housekeeping functions currently assigned to the technologists, thereby freeing them to perform analysis functions. By expediting the computation process, it provides more design alternatives. Zumsteg and Crossman (1986) are currently exploring a similar approach to the design of composite material structures.

The expert executive uses the multiple representation system (MRS) developed at Stanford University (Russell, 1985). Although the development environment was Franz Lisp on a Unix VAX 11/780, the entire system has been ported to an Apollo workstation network. The rule base consists of about 100 rules. In addition, a fact base contains information about the inputs and outputs for each of the programs to be configured. The size of the fact base depends mostly on the number of inputs and outputs because each is described by one fact.

### Knowledge Base

The rules provide knowledge about solving a specific, user-defined problem. For example, the expert executive has several rules for finding the value of a variable. Variable values are represented in this form:

(value-of \$var \$val)

This translates to "the value of \$var is \$val," where \$ indicates a variable. The variable sequence is arbitrary

but must be consistent. The internal knowledge base contains one statement, or proposition, in this format for each variable whose value is known through either user input or computation. Thus the knowledge base might contain the following with reference to Figure 1:

```
(value-of s 60 0)
(value-of wf0 4000 0)
(value-of w00 1000.0)
```

As additional knowledge is acquired through computation or interaction with the user, additional value-of propositions are asserted into the knowledge base.

## Fact Base

The fact base consists of input, output, script-name, and theory-name propositions. Input propositions are in the following format:

```
(input propulsion 1 v)
```

This translates to “the first input to the propulsion code is v (velocity).” The sequence number indicates the sequence in which it is read by the Fortran program. Output propositions are in the following format:

```
(output aero 4 cl2)
```

This translates to “the fourth output from the aero code is cl2 (coefficient-of-lift-2).” Script-name propositions are in the following format:

```
(script-name aero “/u1/assoc/kathryn/aeroscr”)
```

This provides a path to the script for executing the aero code. Theory-name propositions are in the following format:

```
(file-name default)
```

This provides a file name for storing a theory (set of propositions). In MRS, a theory provides symbolic partitioning of knowledge, so that a given logical subset of knowledge can be loaded from any specified file, or dynamically created by the knowledge system.

## Problem-Solving Paradigm

The expert executive uses the following problem-solving paradigm: “To run a program, if all the inputs to the program are present, execute the program, and return the result. Otherwise, if there is another program, the outputs of which provide the missing inputs, run that program first. Otherwise consult the analyst.” This paradigm is implemented as follows. First, there must be a rule for running the selected binary program:

```
(if (and (get-all-input-vals $code)
         (compute-result $code $invals $outvals
```

```
          $outvars $invars)
    (output-was-obtained $outvals)
    (pairlis $outvars $outvals $result))
(run $code $result))
```

This rule can be translated as follows: “If all the values for the program’s free-design parameters are present, the computation is performed, output is obtained, and the output variables are paired with their values, then it is true that the program has been run and the result returned.”

Then, for each of the conditions there must be at least one rule with that condition as a conclusion; for example:

```
(if (and (ready $code $invals $outvars $invars)
         (script-name $code $pathname)
         (execute-code $pathname $invals $outvals)
         (pairlis $outvars $outvals $pairs)
         (assert-computed-values $pairs))
    (compute-result $code $invals $outvals $outvars $invars))
```

This translates to the following: “If the program is ready (all its inputs are present in the required sequence), the script-name is known, it is executed, its output variables are paired with the computed values, and the computed values are asserted into the knowledge base, then it is true that results have been computed for the program.” Execute-code is the Lisp function that integrates symbolic and numeric computation. Execute-code is defined as follows:

```
(defun execute-code (script-name invals)
  (setq ports (*process script-name t))
  (setq pi (car ports))
  (setq po (cadr ports))
  (mapcar 'writeall in-vals)
  (setq return-val (readall))
  (close pi)
  (close po)
  (wait)
  return-val)
```

The \*process function creates a process executing whatever script is named in script-name and opens input and output ports to it. Standard input and output for the script are temporarily reassigned to the Lisp function, so that the object program reads from the function writeall, which writes program input values separated by carriage returns. Program output is read analogously by the readall function. When the ports are closed, the process dies.

This compute-result rule can fail because one of the inputs is not known. For this case, there is a second rule:

```
(if (and (unprovable (ready $code $invals $outvars
                      $invars))
         (setof $par (parent $par $code) $parents)
         (run-all $parents $ignore))
    (compute-result $code $invals $outvals $outvars
                    $invars))
```

This translates: "If the program is not ready to run, all the *parents* of the program (those programs whose output is input to the program) are found, and each of them is run, then it is true that results have been computed for the program."

Because the rule for running the program requires that the output values be present, the run rule will fail when this compute-result rule is invoked. However, the side effects of executing this compute-result rule cause the input variables to be bound. The first compute-result rule will now succeed. This rule is recursive because in order to run the parent programs, it might be necessary to run the "grandparent" programs.

The final element in this paradigm is consultation with the analyst. This is done by identifying all the free-design parameters whose values are unknown and requesting values for all of them at once through a Fortran menu-driven user interface program:

```
(if (and (setof $var-1 (and (input $code $* $var-1)
    (unknown
    (output $other-code-1 $* $var-1))
    (unknown
    (value-of $var-1 $val-1))) $temp-1)
    (setof $var-2 (and (antecedent $ant $code)
    (input $ant $* $var-2)
    (unknown
    (output $other-code-2 $* $var-2))
    (unknown
    (value-of $var-2 $val-2))) $temp-2)
    (union $temp-1 $temp-2 $var-list)
    (elements-in $var-list $vars)
    (length $vars $l)
    (> $l 0)
    (execute-monitor $vars $gvals)
    ,user interface program
    ,to obtain var bindings
    pairlis $vars $vals $pairs)
    ;makes variable-value pairs
    (assert-computed-values $pairs))
    ,forward chains to assert variable bindings
(get-all-input-vals $code))
```

Because this paradigm is executed recursively, it can address any level of depth. Because it executes the programs by starting up processes and passing inputs and outputs through ports, it can execute programs written in any language. The only application-dependent component of the system is the fact base, and the only requirement for the fact base is that it must name each input and output for each program and provide a sequence number in the argument list, so that the expert system can uniquely identify each variable, and the computational codes can receive the variables in the correct sequence. Because this information is explicitly given, there is no requirement that the facts themselves be in any special sequence.

## Example Capabilities

In the current example, the expert system is capable of performing the following tasks:

- It can run the weight program only, asking the analyst for any missing input values.
- It can run the aerodynamics code by first running the code and passing the computed output values from *weight* to *aero*, thereby relieving the analyst of the responsibility of asking the weight technologist to execute this code and then correctly entering the computed values into the input list for aerodynamics.
- It can run the propulsion code alone.
- It can run the performance code by first running *propulsion*; then *weight*; *aero*; and, finally, *performance*. (It could just as easily run *weight*, *propulsion*, and *aero*, or *weight*, *aero*, and *propulsion*. The first successful path is chosen based on the sequence of rules and facts.)
- It can compute any named output variable, such as l/d (lift/drag coefficient), or range, by running the code of which it is an output, thereby relieving the analyst of the responsibility of knowing anything at all about the program configuration.
- It can perform simple convergence, in which the analyst wishes to start with a given output, such as range, and compute an input, such as surface area, that, with all other variables held constant, will provide a close approximation of the given range.
- It performs all computations in real time, with the computed outputs being used immediately as inputs by the receiving programs.
- It takes full advantage of the networking capabilities of the operating system; in the case of the Apollo implementation, the weight, aero, propulsion, and performance codes are physically located in nodes in separate buildings.

## Performance

Formal benchmarks have not yet been established for the expert executive. However, it is known that once the fact base is present for a set of four technology codes having approximately 30 input and output facts, the design cycle requires approximately 1.5 minutes plus the sum of the run times of the technology codes. This is comparable to the run time for the experimental Fortran executive program initiating the technology codes and compares very favorably with the multiweek design cycle without automated integration. Because the primary disadvantage of the Fortran executive program is the cost of modifying it to reflect changes in the technology codes (requiring one person for approximately six weeks), the most significant benchmark for the expert executive will be its reconfiguration cost. A

single data point indicates that the fact base can be replaced in approximately one day. Because the fact base is entirely independent of the rule base (which is static) and because a simple and straightforward representation is used for the fact base, we are optimistic that the expert executive will prove clearly superior in its reconfigurability.

We believe that the expert executive for preliminary design will provide a framework for greatly expediting processes that are heavily computation bound with complex interfaces where iterations are very expensive. It can be generalized for any application by simply replacing the fact base. It can interact with any programs that can be compiled in the given hardware and software environment.

### Possible Future Extensions

We are considering several extensions of this system. One extension will be to incorporate other user interface technologies such as voice input and plot output. Another extension will be to incorporate an interface to the expert design analyzer, a prototype version of which has domain-specific knowledge of the design variables and their interrelationships and can, therefore, provide plausible free-design parameters and interpret the program outputs. Related to this enhancement will be incorporation of numeric optimization techniques that will probably employ a symbolic numeric coupling similar to that used in the expert executive. The Apollo-based version of this system has full window menu graphics. We plan to provide interfaces to other graphics functions such as plotting. Finally, we will extend the integration from the current homogeneous workstation network to a heterogeneous hardware and software environment through use of higher-level network protocols. This will enable some numerically computation-intensive programs that currently reside on the Cray and Cyber machines to be incorporated into this architecture.

### Conclusions

We have found that when the knowledge of how to execute a set of numeric programs is separated from the facts about these programs and represented in symbolic form, a highly generic and powerful tool can result. This tool can be used in any application that involves symbolic or numeric computation, with any number of programs written in any languages available in the given computing environment with minimal reconfiguration cost. The architecture will accommodate any program for which a path can be provided, thereby opening the possibility of loosely coupled integration of symbolic and numeric computation for design analysis.

### References

Elias, A. L. (1986). Knowledge engineering of the aircraft design process. In J. S. Kowalik (Ed.), *Knowledge-based problem solving*. Englewood Cliffs, NJ: Prentice-Hall

Russell, Stuart, Esq (1985) The compleat guide to MRS Tech Rep KSL-85-12, Knowledge Systems Laboratory, Stanford University

Zumsteg, J. R., & Crossman, F. W. (1986) *A computer-based design environment for composite material structures* ASTM Symposium on Composite Materials, Testing, and Design (In press)

### AAAI-86 CONFERENCE EXHIBITORS

*Exhibitors:* Exhibitors who have signed up by press time for AAAI-86 include: Ablex Publishing Corporation, Academic Press, Inc.; Addison Wesley Publishing Company, Inc.; Addison-Wesley Training Systems; Advanced Computer Tutoring, Inc.; Advanced Decision Systems; Aion Corporation; Apollo Computers Inc.; Arity Corporation; Artificial intelligence Corporation; Automata Design Associates; Automation News; Benjamin/Cummings Publishing; Cambridge University Press; Carnegie Group, Inc.; CL Publications; Computer Thought Corporation; Conference Book Service, Inc.; Data General Corporation; Delaware Valley AI Association; Digital Equipment Corporation; EIC/Intelligence, Inc.; Electronic Trend Publications; Elsevier Science Publishing Co., Inc.; Lawrence Erlbaum Associates, Inc; ExperTelligence, Inc.; Expert Systems International; Flavors Technology, Inc.; Franz Inc.; General Research Corporation; Georgia Tech Research institute; Gold Hill Computers, Inc; Harper & Row, Publishers, Inc.; Hewlett-Packard Company; Inference Corporation; Integrated Inference Machines, Inc.; IntelliCorp; Intelligence; Intermetrics, Inc.; IntelligenceWare, Inc.; IBM; Kemp-Carraway heart Institute; Kluwer Academic Publishers; Learned Information, Inc.; Lisp Machine, Inc.; Lithp Systems BV., Lockheed Missiles & Space Company, Inc.-Austin Division; Logicware, Inc.; Lucid, Inc.; McGraw-Hill Book Company; Micro Data Base Systems, Inc.; the MIT Press; Mitre Corporation, Morgan Kaufmann Publishers, Inc; Naval Research Laboratory, Information Technology Division; Neuron Data; Ohio State University Laboratory for AI Research; W. W. Norton & Company; Programming Logic Systems, Inc.; Programs In Motion, Inc.; Queen's University Computer Science Department; Quintus Computer Systems, Inc.; Radian Corporation; RCA Corporation Advanced Technology Laboratories; Richmond Publishing Corporation; SEAI Technical Publications; Silogic, Inc.; Scientific DataLink; Software Architecture & Engineering, Inc.; Software House, Inc.; Software Intelligence Laboratory, Inc.; The Sperry Corporation; Springer-Verlag New York, Inc.; Sun Microsystems, Inc.; Symbolics, Inc.; Systems Concepts; Systems Designers Software Inc.; Teknowledge, Inc.; Tektronix, Inc.; Texas Instruments Inc.; University of California Department of computer Science; John Wiley & Sons, Inc.; Xenologic Inc.; Xerox Artificial Intelligence Systems.