A Distributed Computing Environment for Multidisciplinary Design

R. P. Weston* and J. C. Townsend*
NASA Langley Research Center, Hampton, VA 23681-0001

T. M. Eidson†
High Technology Corporation, Hampton, VA 23666

R. L. Gates‡
Computer Sciences Corporation, Hampton, VA 23666

Abstract

The Framework for Interdisciplinary Design Optimization (FIDO) project has the goal of developing a general distributed computing system for executing multidisciplinary computations on a networked heterogeneous cluster of workstations and vector and massively parallel computers. The concept being used for FIDO is course-grained parallelism, with instances of disciplinary codes (aerodynamics, structures, etc. for an airplane design problem) running on separate processors (including fine-grain parallel computers), under control of an executive on another processor, and exchanging data through a centralized data manager (on yet another processor). To allow the user to monitor the progress of the design iterations, the system includes a graphical user interface (which tracks the execution of codes performing the design iterations) and a separate process, called Spy, which allows a user to extract and plot data produced during current and previous design cycles, and then to steer the design process by changing appropriate control data. The software is written in modular form to ease migration to upgraded or completely new problems. In its current state of development, FIDO is being applied to a highly simplified case of a High-Speed Civil Transport design, involving a simplified problem with very few design variables. However, it has already demonstrated the ability to coordinate multidisciplinary computations and communications in a heterogeneous distributed computing system.

Introduction

Established as a part of NASA's contribution to the national High Performance Computation and Communication Program (HPCCP)[1], the Langley Framework for Interdisciplinary Design Optimization (FIDO) project has as its goal the development of a general programming environment for distributing a multicomponent computational problem across a networked system of heterogeneous computers. A multidisciplinary airplane design process was chosen for the development of the programming environment because of the interest in improving the efficiency of that process.

The FIDO system provides a means for automating the design process. It facilitates communication and control between components of the system, which include the diverse discipline computations involved in a design problem and the system services that facilitate the design. The computers used can include workstations, vector supercomputers, and parallel-processing computers, although only UNIX®-based workstations are used currently. Each computational task can be done by the computer type most appropriate for it. All of the computers involved are networked together, have access to centralized data, and work on their parts of the design, simultaneously whenever possible, under the coordination of a master code.

The following sections describe the model problem, conceptual environment, communications library, task control, user interface, data management, discipline segment functionality, and the data interrogation segment.

FIDO Model Problem

A simple model of a High-Speed Civil Transport (HSCT) design problem was chosen for the initial implementation of the FIDO system[2]. This problem illustrates the type of computational problem envisioned for the FIDO system; however, the scope is much reduced so that the focus can be specifically on computational system issues rather than on the design problem. Figure 1 is a view of the overall optimization loop for this simplified design problem. A particular design cycle (one counter-clockwise circuit of the loop) begins with the choice of the flight conditions, base aircraft geometry, and the values of the design constraints and the design variables, as shown at the top of the figure.

* Research Scientist, AIAA Member, Multidisciplinary Design Optimization Branch

† Research Scientist

‡ Computer Scientist

For the FIDO model problem, four disciplines are involved in the analysis of the aircraft: aerodynamics, structures, performance, and propulsion. In each cycle of the design, these disciplines are invoked to analyze the current definition of the aircraft as specified by the current values of the design variables. The results of the analyses provide a set of system responses that correspond to the current design variables. The optimizer program uses these responses plus the derivatives of the system responses with respect to the design variables. At the current stage of FIDO's development, these derivatives are obtained by finite differences using multiple analyses on a perturbed set of design variables. In the future, they may be obtained by other methods, such as automatic differentiation[3] within the discipline codes.

In order to derive new values for the design variables, the optimizer uses an objective function (which in this case is the minimization of the aircraft's gross weight for the specified range and payload) along with the current system responses and derivatives. These new values are fed back into the design loop until the process converges.

FIDO Environment

The conceptual environment in which the various discipline and service codes run is illustrated in figure 2. Each of the discipline codes (Aerodynamics, Structures, Performance, Propulsion) runs on a separate workstation, and they communicate with each other by sending and receiving information through a central data-manager code, which may run on a separate workstation. Because the role of the data manager is only to move information and manage its storage, a separate code called the interdisciplinary interface code (Interface) performs any computations necessary to convert data from one discipline into a form suitable for another discipline. (An example is the integration of aerodynamic pressures to provide structural forces.) Likewise, this code sends and receives its data from the data manager. A master program (Master) controls the order in which the various discipline codes run. An executive program initializes the FIDO communication network (COMM Network), which forms the backbone of the system.

The discipline, data-manager, executive, and master codes, along with the optimizer code (Optimization), comprise the primary computational segments(described in the next section) of the FIDO network. To provide a means of looking at the data as the design process proceeds, an auxiliary segment called 'Spy' is added. The Spy segment can be started while computations are underway, with multiple instances of Spy being allowed. With proper permissions, these can run on any workstation connected to the network, which allows remote consultants to view results as they are produced and to give timely advice.

An independent program (Setup) can be invoked by the problem designer to pick the flight conditions, the base aircraft geometry, the initial design constraint values, and the initial values of the design variables from a range of previously stored possibilities.

FIDO Communication Library

The FIDO communication library (in the middle of figure 2) is designed to be used as the communication backbone of a system of codes executed in a heterogenous, distributed network of computers. The modules in this library, as well as the overall concept for FIDO, were developed by the third author. The various computational tasks (and codes to do these tasks) must be sorted into groups (called segments) for COMMLIB. Each of these tasks is targeted for a different computer. The segments are further sorted into those that are primary and those that are auxiliary. The primary segments are those that perform functions essential to a run; each requires a driver or server to be started at the beginning of the run. The auxiliary segments are those that do not perform essential functions and so can connect with the COMM Network at any time via a user request. Even for the primary segments, not all the executable code must be loaded with the server; it can be executed later via a forked or remote process if that is more efficient.

The collective execution of the primary segments on a set of computers linked together by a physical network forms a distributed execution system. The part of that system that consists of the communication library, the selected computers, and the physical network is referred to as the COMM Network. The segments use the COMM Network for various communication chores by making calls to COMMLIB.

Although any two segments can directly transfer messages via COMMLIB, a more conservative approach for transferring data is to use a single data manager. The library includes a special set of message-passing functions that communicate with a data manager segment, which runs in parallel in a service mode while the other segments execute tasks. Thus, the system can be programmed so that nearly all global data (data used by multiple segments) are sent to and retrieved from the data manager. Both direct messages and data-manager messages can be used as the system designer determines best. Additional special calls are available to send profile information to the profile analyzer segment (currently the data manager) and to send event signals to the monitoring graphical user interface (GUI).

The communication library is merely a tailored interface to a low-level message-passing library (currently the PVM system from Oak Ridge National Labs*). It is tailored to provide the necessary functionality in a simple interface format that supports a distributed execution system like that described above. Each call in COMMLIB can actually send multiple messages to conduct a complex handshaking communication protocol between two segments, which allows the user-level calls to be simple. In addition, this approach provides improved portability and generality to the execution system because only COMMLIB itself needs to be modified if the low-level message-passing software is changed.

Task Control

The master segment (Master in figure 2) is responsible for the overall control flow of the primary segments in the execution system. A major exception is when multiple tasks (and, thus, segments) are started by the master in parallel; any synchronization needed between the parallel tasks must be done via traditional parallel-programming barrier calls.

The primary segments of the execution system operate in a host/slave mode. The executive code acts as the host and is initiated first. This code sets up the COMM Network, based on configuration files (ASCII data files that provide user-configurable system specifics). Specifically, it first runs a procedure to start a communication server on each computer. The executive code next distributes and starts all primary segments on the appropriate computers via the communication library (COMMLIB). The master code is then begun, which executes control tasks via COMMLIB within each of the primary segment servers as necessary to complete the appropriate sequencing of work for a run. Finally, the segment servers and the COMM Network are shut down.

Data Management

The purpose of the FIDO data manager (Data Manager in figure 2) is to provide a centralized access service for the storage and retrieval of data during a run of the FIDO system. For a given design problem, the definition of the data to be handled during a run is prepared by the problem designer during the setup phase, which results in several configuration files that define the data in a standardized format and contain initial values for appropriate data. On start-up, the data manager reads and internally stores the information in these files and then enters into a service mode; in this mode, data values are stored and retrieved upon request via COMMLIB calls during a run, and data files are moved among FIDO work and archival directories at the appropriate times. A detailed and a summary report file are generated to document the data-management activities during a run.

One of the files generated in the setup phase, which contains information about the atomic data elements, provides the basic description of the managed data. In this file, numeric codes are assigned to individual integer, floating point, and character string data elements or to arrays of these. An include file contains corresponding mnemonic macros for each of the numeric codes. These mnemonic macros are used in the discipline driver codes to reference the corresponding data values. Additional information provided for each atomic data element includes the name, type, units, a short description, usage rules, and array length, where appropriate.

Another file identifies the groups of atomic data elements that comprise the actual data packages sent and received by the data manager. This grouping of character strings and integer and floating point numbers results in more efficient use of the message-passing system. By using configuration files for all the data structures, the data manager code itself is kept generic and does not need any special coding for a new problem.

Discipline Segments

The discipline segments are represented by the blocks labeled Aerodynamics, Performance, Optimization, Interface, Propulsion, and Structures around the lower part of figure 2. Each functionally consists of two parts. One part is the set of codes that perform the discipline computations; the other is a driver that handles the discipline communications via COMMLIB and calls on the discipline codes as needed.

The discipline codes generally are established Fortran codes with well-known characteristics and proven reliability. These are the codes that do the actual analysis, which may be computationally intensive. A minimum of changes are made within these codes to prepare them for use with the FIDO system. These changes are those necessary to put these codes into what can be considered subroutine library form. That is, these codes are changed to make them callable as subroutines with computation and input/output control managed through subroutine arguments.

This control of output from the discipline codes is helpful in providing an orderly function of the FIDO

* PVM information is available by sending the e-mail message “send index from pvm3” to *netlib@ornl.gov*

system. By applying appropriate control, a summary log of the multiple analyses can be kept and the data that are needed can be correctly passed from one discipline to another. Data to be kept (including complete output files) are managed by the FIDO system, and the Spy segment is able to retrieve data interactively for display as requested during a run.

The discipline drivers are written in C; the discipline codes are called as subroutines. Each driver contains several blocks of code that are invoked by the master as needed to handle the start, analysis, gradient, and exit phases of the problem. Each block contains sub-blocks of code for computation, normal completion tasks, and error-handling tasks. The discipline drivers in the FIDO model problem can serve as templates or examples for more complex problems.

The design variable values, file names, and other data are passed through the discipline drivers as arguments to the discipline codes. According to the problem requirements, more than one discipline code may need to be called by a driver. An example is the aerodynamics discipline, which calls separate codes to obtain the induced, wave, and friction drag components in computing total drag for the model problem. The interfaces between the discipline codes and their drivers must be accurately specified in order to provide proper communications. In addition, the user needs to specify what the output files from the code are and how they are to be used by Spy.

For the FIDO model problem, the principal disciplines are aerodynamics, structures, propulsion, and performance. Functionally, the optimizer and interdisciplinary codes are also treated as discipline codes within the system. Because the emphasis in FIDO has been on developing the system, the discipline codes were chosen for speed rather than accuracy. They include linear aerodynamics codes and an equivalent plate structures code[4]; these are considered “low fidelity” codes but run in a few minutes on a workstation. In the future they will be replaced with medium- or high-fidelity codes (Euler or Navier-Stokes aerodynamics, finite-element structures), which will take minutes or hours to run on parallel or vector supercomputers.

Graphical User Interfaces

A graphical user interface has been developed to display the state of the FIDO system at all times from start-up to completion of a run. An example of this display is shown in figure 3, which illustrates the overall FIDO user interface concept. After initiating the FIDO system, the problem designer monitors the progress of the computations using the GUI windows shown on the leftmost screen. The upper left window displays current run parameters and contains pull-down menus for setting various options. The right window displays a simplified problem diagram and indicates which parts are starting up, active, inactive, or shutting down by changing the color of appropriate parts of the diagram. The color key is displayed in the lower left window. Additional detail of the system state can be obtained by selecting the boxes with a 3-D appearance. Doing so brings up an associated window (not shown) that displays sub-detail for the computations represented by that box. The level to which this nesting is continued is determined by the problem designer and specified in a GUI configuration file.

Spy Segments

The FIDO Spy segment (Spy in figure 2) is designed to provide secondary services to system codes that use COMMLIB. These services allow users of the FIDO system to retrieve results while the computations are in progress and display them as text or graphics. In addition, the principal user can alter the values of selected variables in order to provide guidance to the design process.

Four types of data are available: problem definition, cycle status, cycle history, and profile data. The problem definition data consist of the fixed problem parameters, initial values of some variables, and miscellaneous descriptive information. The cycle status data contain the current cycle number, phase, and task; a list of data that is available in Data Manager and Spy; and miscellaneous timing information. The cycle history data consist of selected scalar values and selected data arrays (on the computational grids) for each completed cycle. Finally, the profile data consist of the execution time histories of the various computational tasks. These data can be displayed on the screen as text or in graphical format. Current graphical formats include line plots (e.g., cycle history of the objective function, as shown in the middle screen of figure 3) and contour plots of distributed data (e.g., surface pressures, stresses, or deflections, as shown in the righthand screen of figure 3).

Multiple instances of the Spy code can connect to and communicate with the COMM Network that is used by a set of primary segments. Thus, the master/slave paradigm is augmented to include multiple (but limited) masters. The Spy connections can be initiated at any time while the FIDO system is running a problem. In the current implementation, new Spys become active only at the beginning of a new design cycle; future implementations will allow activity to begin within a cycle.

Actually, two types of Spy segment exist. One is designated as the ‘Master Spy’ and has special properties. It can be invoked only by the user who is running

the FIDO problem (the “designer”); only this user is allowed to change the current values of design variables, constraints, and initially-set parameters. This capability provides some measure of manual control over the design process. Because the problem designer may need advice during the design process (perhaps from a remotely located expert in one of the disciplines involved), the other type of Spy segment is provided.

This Spy type is designated as an ‘Agent Spy’; it is limited to displaying data, but can exist in multiple instances. It can be invoked with proper permission from any location connected to the network (e.g., anywhere on the Internet). Thus, the remote expert’s workstation can display any of the information available to the designer (shown schematically in figure 3). In fact, several users can examine the same or different FIDO output at different locations at the same time.

Sample Results

A simplified model HSCT wing design problem has been used in the development of the FIDO system. It uses three aerodynamic design variables (wing leading-edge sweep, root chord, and spanwise break location) and two structural design variables (inboard and outboard panel thicknesses for the equivalent plate structural model). Over a run of twenty cycles, the line-plots in the center Spy screen of figure 3 provide an example of how the design variables, system responses, and problem constraints evolved. The design objective of minimizing the total weight is shown as the top line-plot. These computations were made on a system consisting of one SGI and six Sun workstations

Concluding Remarks

The Framework for Interdisciplinary Design Optimization (FIDO) is being developed as a general programming environment for automating the distribution of complex computing tasks over a networked system of heterogeneous computers. The FIDO system facilitates communications between computational tasks distributed over a computer network and provides for automatic interactions in multidisciplinary problems, for example, how to reach a nearly optimal consensus in the aircraft design process.

In the FIDO system, the computers involved are networked together, have access to centralized data, and work on their parts of the design simultaneously in parallel whenever possible, under the direction of a master code. Each computational task can be assigned the computer type most appropriate for it. An auxiliary code, Spy, is provided for viewing results as they are produced and for steering the design process. In the viewing mode, it can be run as multiple instances and from remote locations. The FIDO software is written in a modular form in order to ease migration to upgraded or completely new problems: different codes can be substituted for each of the current code modules with little or no effect on the others.

The FIDO system has been designed to be adaptable to any distributed computing problem. It has been demonstrated for a simplified High-Speed Civil Transport (HSCT) aircraft design problem; currently, a more complex HSCT problem is being implemented.

References

[1] Holst, T. L., Salas, M. D., and Claus, R. W., “The NASA Computational Aerosciences Program - Toward TeraFLOPs Computing,” AIAA Paper No. 92-0558, Jan. 1992.

[2] Townsend, J. C., Weston, R. P., and Eidson, T. M., “A Programming Environment for Distributed Complex Computing - An Overview of the Framework for Interdisciplinary Design Optimization (FIDO) Project,” NASA TM 109058, Dec. 1993.

[3] Carle, A., Green, L., Bischof, C., and Newman, P., “Applications of Automatic Differentiation in CFD,” AIAA Paper No. 94-2197, June 1994.

[4] Barthelemy, J.-F., Wrenn, G.A., Dovi, A.R., Coen, P.G., Hall, L.E., “Supersonic Transport Wing Minimum Weight Design Integrating Aerodynamics and Structures,” J.Aircraft, Vol. 31, No. 2, pp. 330-338, Mar-Apr 1994..

Figure 1. FIDO model diagram for HSCT design problem

Figure 2. Schematic of FIDO system interactions.

Figure 3. GUI concept, including designer interaction with consultant through Spy.

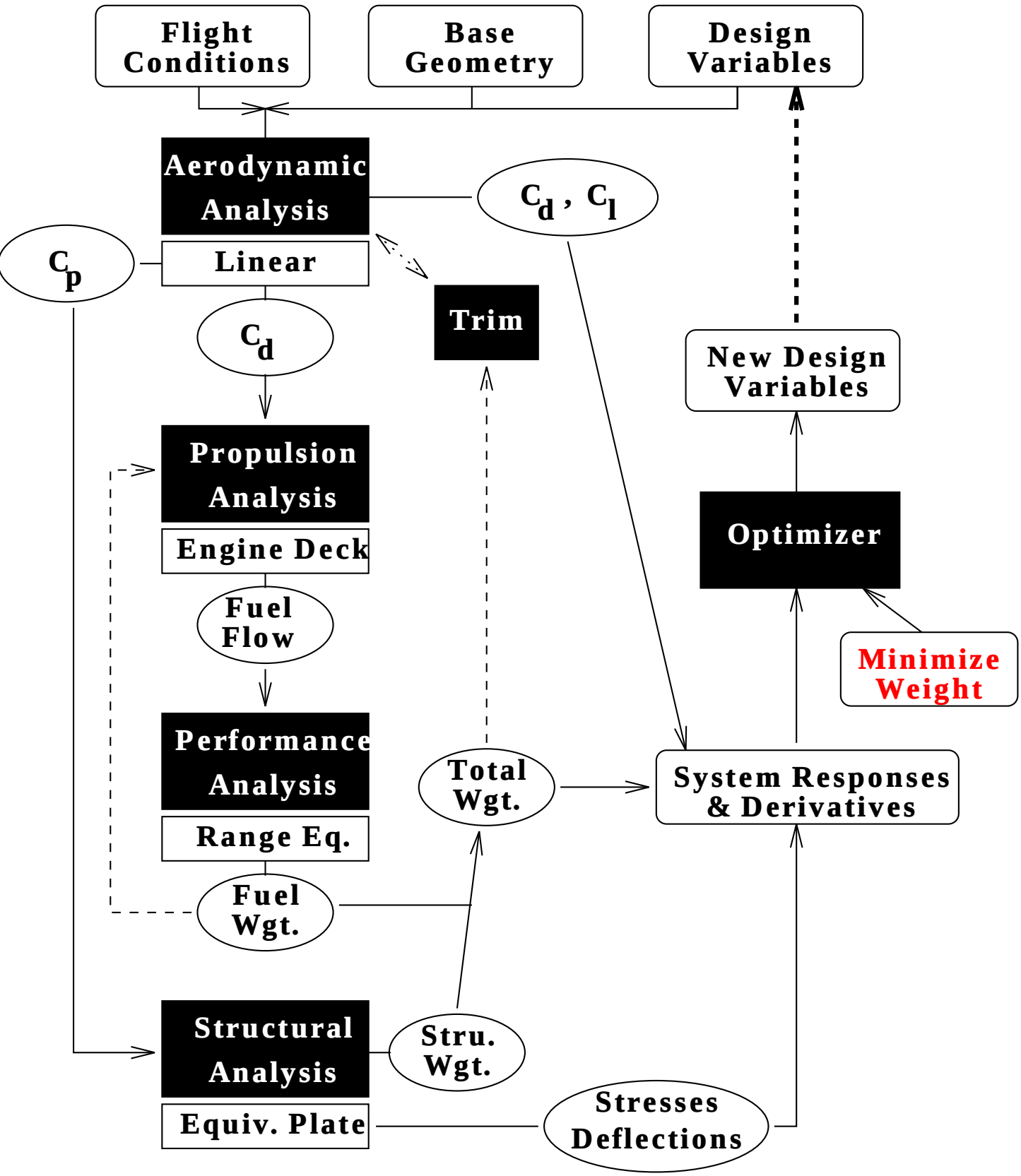

Figure 1. FIDO model diagram for HSCT design problem