Computing 21, 3 (June 1994), 301-315.

[14] P. T. Homer and R. D. Schlichting. Using Schooner to support distribution and heterogeneity in the Numerical Propulsion System Simulation project. *Concurrency—Practice and Experience 6*, 4 (June 1994) 271-287.

[15] A. A. Khokhar, V. K. Prasanna, M. E. Shaaban and C. Wang. Heterogeneous computing: Challenges and opportunities. *IEEE Computer 26*, 6 (June 1993), 18-27.

[16] Pratt & Whitney. The Pratt & Whitney PW4084 Engine. On-line html document. (http://www.utc.com/PW4000/fam112.html)

[17] J. A. Reed and A. A. Afjeh. Distributed and parallel programming in support of zooming in numerical propulsion system simulation, *OAI/OSC/NASA Symposium on Application of Parallel and Distributed Computing*, Columbus, Ohio. April 1994.

[18] J. A. Reed. Development of an Interactive Graphical Aircraft Propulsion System Simulator. Master of Science Thesis, University of Toledo, August 1993.

[19] T. J. Sullivan et al. Energy Efficient Engine: Fan Test Hardware Detailed Design Report. NASA CR-165148, 1983.

[20] Transportable Applications Environment Plus. Programmer's Manual, Version 5.2. Goddard Space Flight Center, National Aeronautics and Space Administration. December 1992.

Acknowledgments

The NPSS project is managed by the Interdisciplinary Technology Office (ITO) at NASA Lewis Research Center (LeRC). This work was performed in part on computing resources at the Advanced Computational Concepts Laboratory (ACCL) and the Computer Services Division at LeRC. Thanks are due to G. Follen, C. Putt and C. Miller of LeRC. This work has been supported in part by NSF grant ASC-9204021 and NASA grants NGT-50966, NAG3-1560, NCC-3-207, and NCC-3-452.

References

[1] Advanced Visual Systems Inc. *AVS Developer's Guide* (Release 4.0), Part number: 320-0013-02, Rev B, Advanced Visual Systems Inc., Waltham, MA, May 1992.

[2] Aviation Trade News. *The Avion Online Newspaper 83,* 4 (February 8, 1995). On-line html document (http://avion.db.erau.edu/avion/issues/spring95/issue4/aviationtrade.html).

[3] R. W. Bucy et al. Energy Efficient Engine: Component Development & Integration, Steady-State Performance Computer Program User's Manual for the IBM 370 and CDC 6600 Computers, GE Reference Computer Program - G0045I. R83AEB553, October, 1983.

[4] D. L. Burrus et al. Energy Efficient Engine: Combustor Test Hardware Detailed Design Report. NASA CR-168301, March 1984.

[5] R. W. Claus, A. L. Evans, G. J. Follen. Multidisciplinary propulsion simulation using NPSS. *4th AIAA/USAF/NASA/OAI Symposium on Multi-disciplinary Analysis and Optimization*, Cleveland, OH (September 1992).

[6] R. W. Claus, A. L. Evans, J. K. Lylte, and L. D. Nichols. Numerical propulsion system simulation. *Computing Systems in Engineering 2*, 4 (April 1991), 357-364.

[7] CLIPS Reference Manual, Basic Programming Guide. Software Technology Branch, Lyndon B. Johnson Space Center. CLIPS Version 5.1, September 10, 1991.

[8] D. Y. Davis and E. M. Stearns. Energy Efficient Engine—Flight Propulsion System Final Design and Analysis. NASA CR-168219, contract report prepared by General Electric Company, August 1985.

[9] GE 90, General Electric Aircraft Engines. On-line html document (http://www.ge.com/aircraftengines/ge90).

[10] E. J. Hall, R. A. Delaney, and J. L. Bettner. Investigation of Advanced Counterrotation Blade Configuration Concepts for High Speed Turboprop Systems, Task 5 — Unsteady Counterrotation Ducted Propfan Analysis Computer Program User's Manual, NASA CR-187125, Jan. 1993.

[11] R. Hayes. UTS: A Type System for Facilitating Data Communication. Ph.D. Dissertation, Department of Computer Science, University of Arizona, August 1989.

[12] P. R. Holloway et al. Energy Efficient Engine: High Pressure Compressor Detailed Design Report. NASA CR-165558, 1985.

[13] P. T. Homer and R. D. Schlichting. A software platform for constructing scientific applications from heterogeneous resources. *Journal of Parallel and Distributed*

	TESS	**Monitoring tool**	**ADPAC watch-dog**
Development and testing	SGI 4D/480, or Sun Sparc10 NASA Lewis	Sun Sparc10 NASA Lewis Research Center	Nodes on a cluster of IBM RS-6000 workstations, or nodes on an IBM SP-2 NASA Lewis Research Center, Cleveland, OH
	SGI 4D/440 University of Toledo	Sun Sparc10 University of Arizona	
SC'94 Washington, D.C.	SGI Onyx Conference floor	Sun Sparc10 Conference floor	
SC'95 San Diego, Ca.	SGI Indy Conference floor	Sun Sparc10 Conference floor	

Table 1: Configurations used to test prototype simulation system

areas. First, the expert system needs to detect and handle convergence problems that arise in the zooming strategy. For example, these include numerical inconsistencies between solvers used in the high- and low-fidelity simulations, and the ill-defined boundary condition problem described in Section 3.1. The second area is machine availability. This can range from dealing with short duration network outages to machine downtime schedules. For example, when a network outage occurs, the expert system may decide that enough ADPAC results are available to build the performance map, or that the simulation should wait until the network connection is restored, or that new ADPAC runs should be started on other accessible machines.

Another direction is to modify the source code of ADPAC to allow it to communicate directly with TESS and the monitoring and control system, rather than through its output files. A principal reason for not modifying the source initially is the desire to prove the feasibility of this approach and identify the specific changes needed. Another positive feature associated with using the output files is that it would be relatively straight-forward to substitute a different high-fidelity fan simulation and provide a similar level of monitoring through its output files. This technique allows for easy testing of different fan simulations without the initial need to involve the authors of the simulation.

Another zooming approach being studied is to use an intermediate fan simulation, specifically a two-dimensional, axisymmetric simulation. This has the advantage of not requiring as much execution time as the three-dimensional ADPAC simulation when less accuracy is needed. In addition, it will be possible in some cases to use the solution from the medium-fidelity simulation to jump-start the three-dimensional solution, thus shortening the execution time of the high-fidelity simulation.

The Schooner system is being extended to include better fault detection techniques and provide information about failures to the expert system and to the user. Schooner is also being extended to work with batch queueing systems which will provide greater access to machine resources. This will facilitate longer simulation runs by providing ADPAC platforms during more hours each day.

module. The configuration shown in Figure 4 is for the E^3 engine. Steady-state overall performance maps for the operation of the fan, high-pressure compressor, high-pressure turbine, and low-pressure turbine were constructed from engine design and test-stand data [3, 4, 12, 19]. Individual component design-point data, based on design and cycle-deck data, are used to define the operational parameters in each component. The TESS E^3 engine model can be executed in both zooming and non-zooming modes; the choice is made by selecting the appropriate fan component. Figure 4 shows the zooming mode with the *fan Multi-ADPAC* module in the engine.

5.2 High-fidelity fan

A grid geometry model representing the engine cowl, fan, bypass, and S-duct regions was developed from the E^3 fan geometry data. An axisymmetric view of the grid is shown in Figure 10b. Booster rotor and stator, core outlet guide vanes, and bypass outlet guide vane geometry were not included in order to reduce computing times. Experience modeling the E^3 fan with complete geometry has shown that computing time increases by approximately a factor of four when these components are included. For this work, overall simulation time is highly dependent on the ADPAC simulations; the increased time of computation outweighs the marginal improvement in predicted values when the additional geometry is included. The structured, multi-blocked, three-dimensional grid represents a single blade passage and consists of 16,380 grid points. The steady flow field in the fan is analyzed using the ADPAC solver and the given geometry to solve the inviscid Euler equations. To determine the single-valued flow parameters for the exit boundary condition, the mass-averaging process is applied at a location approximately one and one-half fan diameters downstream of the fan trailing-edge.

5.3 System Configuration

The prototype simulation system has been tested in several configurations. The following list shows the platforms that each part of the system can currently use:

- TESS executes within AVS version 5, and has been tested on SGI and Sun architectures,
- The Monitoring tool uses TAE+ version 5.2 and has been tested on Sun workstations, and
- The ADPAC binaries available for this project are compiled for the IBM RS-6000 and SP-2 architectures.

An early version of the system was demonstrated at Supercomputing ‘94 and the current version was demonstrated at Supercomputing ‘95. Table 1 summarizes these configurations and two development versions that are in use for testing zooming strategies and expert system techniques.

6. Future Work

The prototype simulation system described here provides a framework for developing and evaluating new zooming strategies, implementing monitoring and control system techniques, and addressing interconnection and configuration problems. The major need in the prototype is for expansion of the role of the expert system to allow long unsupervised simulation runs. The system as it currently exists has been tested and works well for simulation runs of up to several hours. To achieve longer, more realistic simulation runs requires expert system assistance in two broad

Figure 9: NASA's Energy Efficient Engine

engines such as the GE-90 [9]. Second, E^3 geometry and performance data is non-proprietary and readily available.

The E^3 fan consists of a number of individual hardware components. The major components are identified in the cross-sectional view of the E^3 fan geometry shown in Figure 10a. In addition to the fan blades that are attached to the rotating hub, a quarter-stage booster is used to provide additional super-charging for the core components. The quarter-stage island splits the total fan flow so that a portion of the total flow is supercharged by the quarter-stage rotor. Downstream of the booster rotor, the flow is further split with some booster flow re-entering the bypass stream and the remaining flow directed through the core outlet guide vanes (OGVs) and S-duct into the core.

This section describes how the prototype simulation system is used to model the E^3 engine with zooming on the fan component, and gives the machine configurations used in building and demonstrating the E^3 simulation.

5.1 Engine Model

The TESS E^3 engine model represents each major engine component with a comparable TESS

```
# called to get fan id to monitor and pathname to ADPAC output files
import mon_get_fan_number prog(
    "line_id"   val integer,   "path_name" res string[-])
    returns ("fan_id" integer)

# residual error reports, 10 results reported on each call
import mon_residual_report prog(
    "fan_number" val integer,  "residual_count" val integer,
    "cycle"      val array[10] of integer,
    "max_err"    val array[10] of float)

# warning reports
import mon_warning_report1 prog(
    "fan_number" val integer,  "message"  val string[-])
import mon_warning_report2 prog(
    "fan_number" val integer,  "message"  val string[-])

# reports at end of run showing ADPAC results
import mon_mass_in_report prog(
    "fan_number" val integer,  "mass_in"  val float)
import mon_mass_out_report prog(
    "fan_number" val integer,  "mass_out" val float)
import mon_pressure_report prog(
    "fan_number" val integer,  "pressure_ratio" val float)
import mon_pressure_plot_report prog(
    "fan_number" val integer,  "pressure" val array[52] of float)
```

Figure 8: ADPAC Watch-dog UTS Specification

and the corresponding list of output file names. In a manner similar to that used by *fan Multi-ADPAC*, the monitoring tool creates new lines, one for each watch-dog process. The watch-dog process uses the `get_fan_number` procedure to obtain the fan id it is monitoring and the pathname to ADPAC's output files. During execution, the watch-dog process makes repeated use of the `residual_report` and `warning_report` procedure calls to pass information back to the monitoring tool. At the end of the ADPAC run, the final results are read from files created by `mbave` and reported to the monitoring tool using the appropriate procedure calls.

5. Using the Prototype

A model of the NASA/General Electric Energy Efficient Engine (E^3) [8] has been implemented using the prototype simulation system. A cross-sectional view of the E^3 is shown in Figure 9. Air enters the engine, passes through the fan where the air flow is split into two streams. One stream passes into the core components of the engine and the other enters the bypass duct. The core flow passes through a high-pressure compressor, combustor, high-pressure turbine, and low-pressure turbine. After passing through the low-pressure turbine, the core flow is mixed with the bypass flow. The combined flow then passes through the nozzle and exits the engine. The E^3 was chosen for this project for two reasons. First, the E^3 design is a direct predecessor of modern turbofan

The `adpac_init` procedure is then used to obtain the initial values for *`makeinput`*. After *`mbave`* has returned, *`adpacslave`* calls `adpac_results` to give the results for this ADPAC instance to *fan Multi-ADPAC*.

A new performance curve is created by *fan Multi-ADPAC* each time fan performance data is needed by TESS. To reduce the overall simulation time, the space-averaged values are retained and used to create an overall fan performance map. Before running flow solutions, this data is checked to see if the current operating conditions are within the data range. If so, the data is interpolated and used in the system simulation. In this manner, the simulation time may be significantly reduced. This also has the added benefit of creating an overall fan performance map which can be used in subsequent, non-zooming TESS simulations.

4.3 Prototype Monitoring and Control System

Since the source code for ADPAC is not available for this project, the output data files are monitored instead. One of the data files is updated by ADPAC on each iteration during a run to report a number of quantities, including the desired residual and several types of warnings. One limitation of this approach is an inability to affect ADPAC once execution has started. Thus, the expert system is currently limited to displaying warnings and errors in the monitoring tool, rather than being able to actively steer ADPAC.

To simulate the type of monitoring desired given the constraints, a watch-dog process is created on each machine executing ADPAC. This process continuously checks ADPAC's output file for new data. As the file changes, the watch-dog examines it for values of interest, specifically the residual values from each iteration and warnings of interest to the expert system. It also reads the average results from the *`mbave`* program at the completion of the ADPAC run.

The specification file for the watch-dog process is shown in Figure 8. An analogous specification file is used with the monitoring tool. Execution of the watch-dog process is started after the monitoring tool receives from TESS the list of machines on which ADPAC is executing,

```
import tess_get_fan_number prog/(
   "line_id"  val integer) returns ("fan_id" integer)

import tess_adpac_init prog(
   "fan_id"   val integer,       "nproc"    var integer,
   "ndata"    var integer,       "xmn"      var float,
   "tin"      var float,         "pin"      var float,
   "xspool"   var float,         "exitpnd"  var float,
   "ilenc"    var integer,       "cname"    var string[80],
   "niter"    var integer,       "ilenssf"  var integer,
   "ssfname"  var string[80],    "niterssf" var integer,
   "iusessf"  var integer,       "initcall" var integer)

import tess_adpac_results prog(
   "fan_id"   val integer,       "ndata"    val integer,
   "ierrflag" val integer,       "results"  val array[10] of float)
```

Figure 7: ADPAC specification file

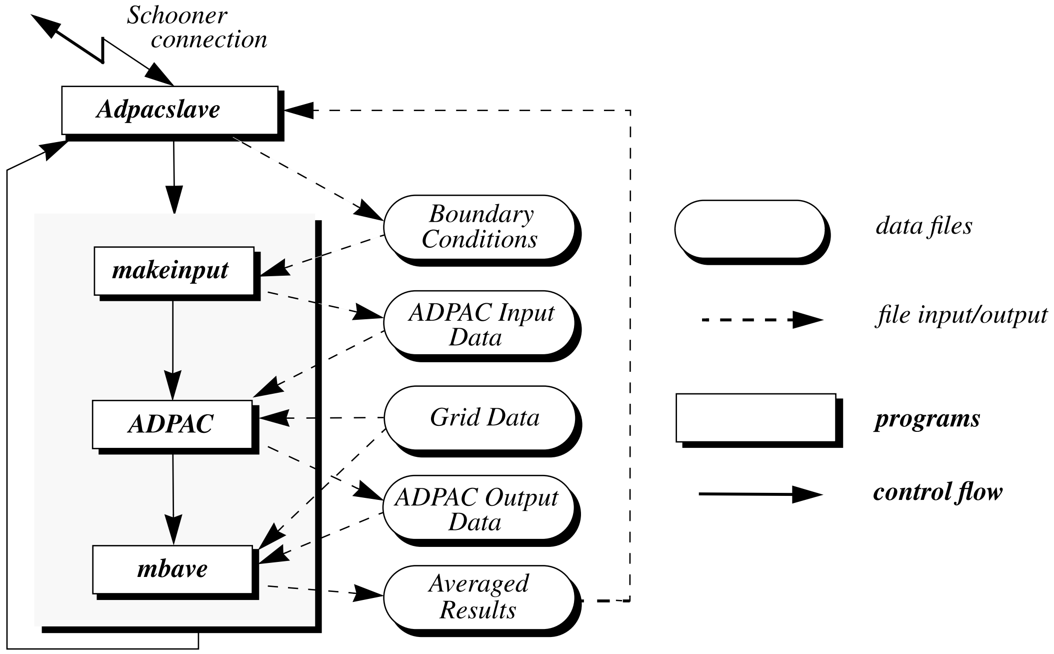
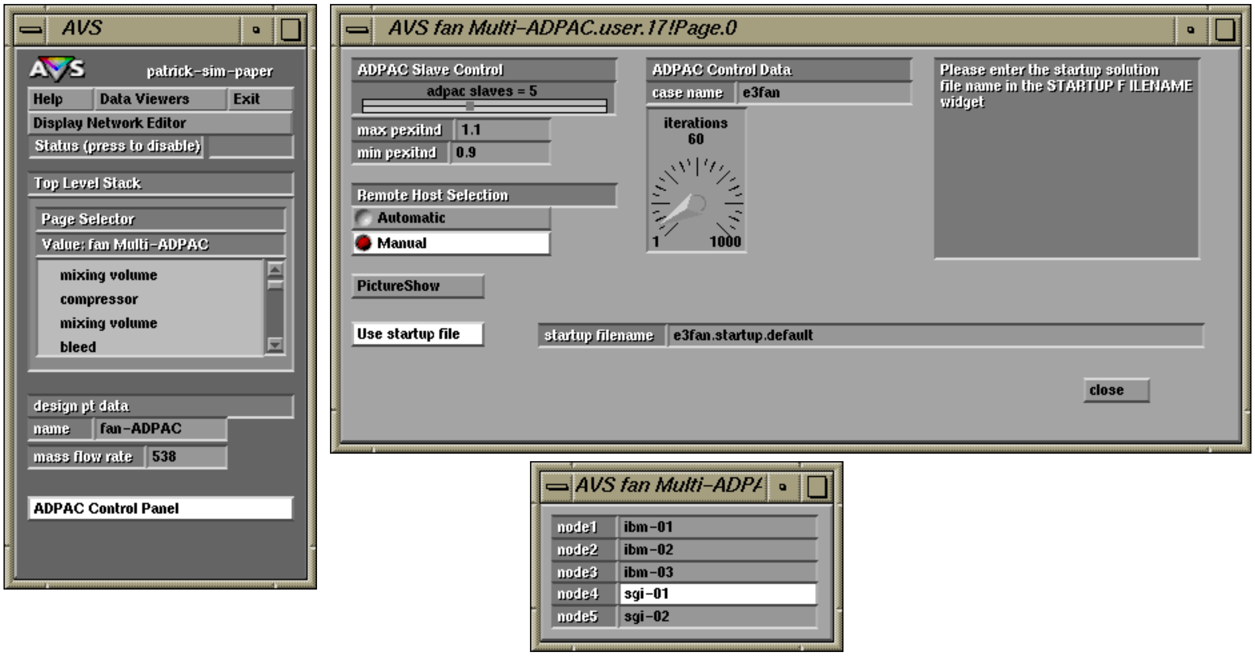


Figure 6: ADPAC code suite flow chart

Each time TESS needs fan performance data during a simulation, *fan Multi-ADPAC* creates the needed remote instances of ADPAC on a user-definable suite of remote machines. Boundary condition parameters are then supplied in response to queries from each ADPAC instance. As each ADPAC completes its run, it sends its results to *fan Multi-ADPAC*. Results are matched with boundary conditions, then used to create data points on the performance curve (see Figure 3). Once all the values have been received, the performance curve is interpolated to match the stagnation pressure ratio across the fan, provided by the TESS simulation, to determine the stagnation temperature ratio and mass flow rate. These values are then used by TESS to continue the complete propulsion system simulation.

A suite of codes has been developed to work with ADPAC and handle the beginning and ending boundary value computations. This allows experimentation with ADPAC as the high-fidelity component without the need to continually update ADPAC's source code during development. This approach also allows the substitution of different fan simulations without the initial need to involve the authors of each simulation. These codes are illustrated in the flow chart in Figure 6. The first program, `makeinput`, creates the ADPAC input data file from the boundary parameters. Then, `ADPAC` executes, reading its grid data file and the input data file. The output file produced by `ADPAC` is then read by the third program, `mbave`. This is a multi-block averaging program and integrates the three-dimensional flow solution to give the single (space-averaged) flow values which are needed by TESS. The `adpacslave` program coordinates the execution of the other three programs, and handles the Schooner communications with *fan Multi-ADPAC*. The UTS specification file for `adpacslave` is shown in Figure 7. The specification file for *fan Multi-ADPAC* is analogous. *fan Multi-ADPAC* starts the ADPAC instances by creating a line for each instance and starting `adpacslave` within each line, using the Schooner dynamic configuration library. Each `adpacslave` uses the `get_fan_number` procedure to determine its fan id for the run.



Value: fan Multi-ADPAC

name: fan-ADPAC

mass flow rate: 538

adpac slaves = 5

max pexitnd: 1.1

min pexitnd: 0.9

○ Automatic

● Manual

ADPAC Control Data

case name: e3fan

iterations 60 (1–1000)

Please enter the startup solution file name in the STARTUP F ILENAME widget

startup filename: e3fan.startup.default

node1: ibm-01

node2: ibm-02

node3: ibm-03

node4: sgi-01

node5: sgi-02

Figure 5: *fan Multi-ADPAC* module control panels

database is updated through a registration procedure each time a new component is started. When a component needs to call a computation not present locally, it first contacts the Manager to locate the needed resource. The Manager provides the location after matching and type-checking the UTS specifications.

Multiple threads of control are provided in Schooner through *lines*. Each line contains one or more components, with a single thread of control passing among those components through the procedure call chain. Dynamic configuration allows lines to be created and components to join lines as needed during execution. One type of cross-line synchronization is provided through the use of a distinguished shared line. This line contains components whose exported computations are available to components in any line.

4.2 Prototype Zooming System

The prototype zooming system is defined by two suites of codes. The first suite, residing on the user's workstation, runs AVS and TESS. The second suite consists of ADPAC and associated codes [17]. One instance of this second suite exists for each of the multiple fan simulations used in the zooming strategy.

A new TESS engine component module, *fan Multi-ADPAC*, was created to provide the user interface and functionality for the zooming system. The module

- Handles the basic AVS data transfer for the fan component within TESS,
- Spawns the remote ADPAC tasks through Schooner library calls, and
- Controls the data transfer between TESS and the ADPAC simulations.

To utilize the *fan Multi-ADPAC* module in a TESS engine simulation, the user defines the ADPAC control parameters and the remote machines on which to spawn the ADPAC simulations. Figure 5 shows the AVS pop-up windows used to accept this input from the user.

and allows component applications to join and leave the meta-computation as needed. This section describes an interconnection system that realizes this model of computing and then provides implementation details on the construction of the prototype simulation system.

4.1 Interconnection System

The Schooner interconnection system realizes the scientific meta-computation model, providing solutions to the interconnection and configuration problems. To solve the interconnection problem, each application becomes one component in the meta-computation and consists of a code block and an interface. The code block contains the user's code and implements one or more computations that accomplish a specific set of tasks. In developing the code block, the scientific programmer can use the combination of programming language, programming model, and architecture that is most suitable. The interface for each component is automatically generated and advertises that components available services to other components, as well as handling the job of locating and accessing the external services needed by the component. Figure 1 illustrates this idea. Schooner provides both static and dynamic configuration control for the user. Static control allows the user to select the components and hosts that will be needed for the execution, and to start and execute the meta-computation. Once execution has begun, dynamic control allows components to be added or removed as needed by the user or through Schooner library calls issued by the components themselves.

Schooner provides its interconnection services through four, mostly orthogonal, parts:

- a specification language,
- an intermediate data representation and accompanying data exchange library,
- a set of stub compilers, and
- a runtime support system.

The Universal Type System (UTS) provides both the specification language and the intermediate data representation [11]. The specification language is machine- and language-independent and is used to describe the interface for each component application. The language supports the standard base types integer, float, double, char, etc. It also supports array and record structure types and procedure parameters. Specific examples of the language are shown in Section 4.2 and Section 4.3 below. At runtime, the UTS specifications are also used to type-check arguments between the user of a computation and the provider of the computation. The UTS intermediate data representation provides a medium for exchanging data across machine architectures and handling data structure differences among languages.

The stub compilers, one for each supported language, read the UTS specifications for a component and generate the interface for the component. The interface contains the calls to the UTS data conversion library. It also provides the communication interface, allowing the component to exchange data and control with other components.

The runtime system implements application-level remote procedure call (RPC) control transfer between components, as well as configuration and control features. Each component in the meta-computation is implemented as a process on its respective host. In addition, there are two other types of processes present. One instance of the Schooner Manager is executed on one of the hosts and an instance of the Schooner Server is executed on each host. The Manager acts as the central coordinator. It is responsible for startup of the various components in the meta-computation, cooperating with the Server on each host to accomplish this. The Manager also maintains a database of the computational routines each component makes available. The reporting of continuous data during execution. A set of pre-defined objects are available that can be used to create vertical and horizontal scales, rotating dials, strip charts, etc. The user can also build objects specific to the application by creating custom objects using the supplied drawing tools in TAE+ and defining the type data — rotational, sliding, stretching, etc. — that will be supplied to the object. The data-driven object was a major reason for selecting TAE+ for this project, as it easily supports the type of monitoring needed for ADPAC.

To accomplish the immediate research goals, the monitoring tool allows the user to observe the progress of ADPAC runs and provides information to the expert system. The monitoring tool consists of windows for each instance of ADPAC. The right side of Figure 1 shows a snapshot of one such window taken at the end of an ADPAC run. The name of the machine executing this instance of ADPAC is shown at the top center of the window. The chart on the lower-right portion is a strip-chart and plots the residual on a log scale over the most recent 100 iterations. The residual provides a measure of how well ADPAC is approaching convergence. Convergence is generally achieved when the residual has dropped four orders of magnitude, while an oscillating residual is a symptom of a problem within ADPAC. Experience with TESS-ADPAC indicates that convergence is reached in most cases in 500 to 800 iterations. When a computation is finished, the strip-chart on the upper-right shows the pressure ratio plot at 52 points along a slice through the fan. The vertical scales on the upper left report the flow rate of air into and out of the fan. The vertical scale on the lower left shows the final pressure ratio (outlet/inlet) computed by ADPAC. Each of the vertical scales also shows the design point provided by the one-dimensional TESS model.

CLIPS. Building an expert system with CLIPS begins with a user-defined set of rules. The rules are written in a functional language and describe actions to take when specified conditions occur in the application. Typically, CLIPS supplies a status window that displays information to the user about the application and shows the progress of the knowledge engine as the package works through the rules. A C language interface is also available that bypasses the CLIPS status window and allows an application to be tied directly to the knowledge engine. This latter feature is being used in this current project. The C interface consists of function calls that pass data to CLIPS and return results from the rules. Callbacks through the C interface allow CLIPS to pass control commands to other components in the system.

The prototype expert system created for this project receives data from the monitoring tool via the C interface. The system then provides appropriate warning messages that are sent by callbacks to the monitoring tool where they are displayed to the user in the form of pop-up windows. When a type of warning can occur more than once in each ADPAC instance, the expert system will only report the first instance to the appropriate ADPAC monitoring window.

4. Assembling the Simulation System

The pieces of the prototype simulation system — the low-fidelity engine model TESS, the multiple instances of the high-fidelity fan component ADPAC, and the monitoring tool and expert system — are combined to create a distributed, heterogeneous meta-computation. This is a new model of scientific computing that requires support for interconnection and configuration. Interconnection support allows the transfer of data and control among the component applications. Configuration support allows the user to start and manage the various applications, degree of control over the simulation to manage the complexity. At the same time, the user should not be forced to constantly monitor simulations that, at a minimum, span hours of real time. This section describes an overall approach to providing monitoring and control of the simulation. It then gives descriptions of the monitoring tool, developed with the Transportable Application Environment (TAE+) GUI toolkit [20], and the control component, developed using the C-Language Integrated Production System (CLIPS) [7].

3.1 Control Strategy

A large number of variables can affect the outcome of a simulation and monitoring them can place a severe burden on the user. Two types of problems that can arise are physically unrealistic boundary conditions imposed on a component and numerical instabilities that arise within a component. Examples of the first type are the rise in air pressure that should occur at the outlet of the fan component, and the rise in temperature through the combustor that should result from the addition of fuel. Both examples would be relatively easy for an expert system to check, requiring only a test to determine if the desired quantity had increased. Correcting such problems would require appropriate rules to modify the boundary values and/or modify the parameters of the governing equations

More complicated examples arise from instabilities introduced by the numerical methods used to solve the problem. One example occurs when the flow equations fail to converge to a solution, falling instead into an oscillatory pattern. The fan component provides another example. Artificial, numerically-induced vortices in the air flow can form, reducing the effective area for flow through the fan and causing pre-mature choked flow. An expert system needs more complicated rules to detect such problems and implement the series of corrective steps

The immediate goal of this research is to build a monitoring and control system that gives the user a means of monitoring the high-fidelity components, and that can detect some types of problems and warn the user when they arise. The longer-range goals are to give the user direct control over more of the simulation parameters, and to develop more complex rules for the expert system to allow it to actively steer the simulation.

3.2 Control System Components

TAE+. This package supports the rapid prototyping and construction of X-windows graphical user interfaces. It provides a workbench that facilitates the design and layout of the application’s windows, allowing easy placement of the various objects within each window. A programming tools package allows the user to add code to the interface to provide program control over the various objects that make up the interface. Finally, a code generator automatically produces code, in a number of languages, that creates the interface and builds the main event loop for the application.

There are three basic building blocks available for use in designing windows for a TAE+ application. The first is a set of user-entry objects that allow the user to interact with the application through buttons, pull-down menus, and text fields. Second, there are data-driven objects that graphically display information from the application in real time through dials, strip-charts, thermometers, etc. The third category is information objects, such as text displays and help screens that provide the user with information or instructions about the application. The data-driven objects are particularly useful in a monitoring tool as they easily support receiving and

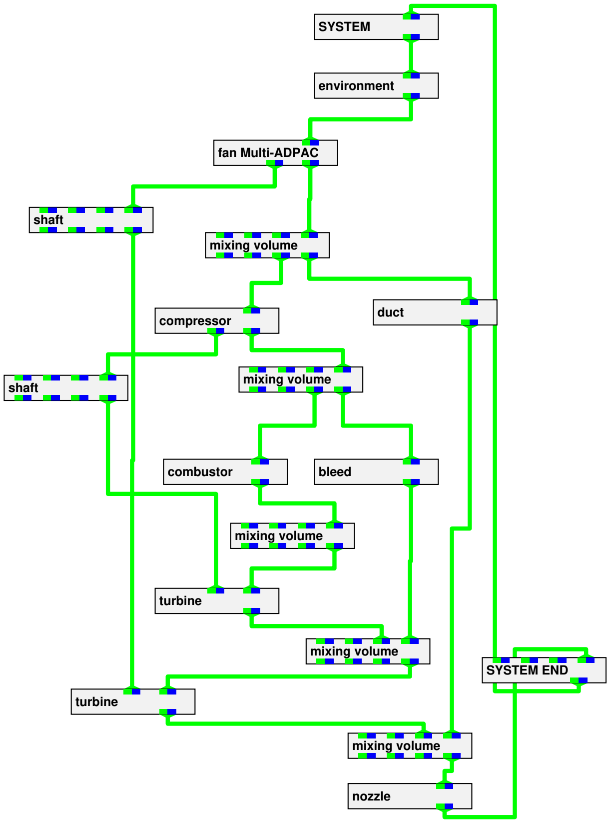


Figure 4: TESS engine model dataflow network

solved numerically for both the steady-state and transient portions of the simulation. Currently, for steady-state solutions, the user may choose either Newton-Raphson or Fourth-order Runge-Kutta methods. For transient solutions, the user may choose either Modified Euler, Fourth-order Runge-Kutta, Adams, or Gear methods. When simulation execution is begun, TESS first attempts to balance the engine at the initial operating point using the steady-state balancing method. Once the engine is balanced, the transient is begun and proceeds up to the number of simulation seconds defined by the user.

Advanced Ducted Propfan Analysis Code. ADPAC is a three-dimensional Euler/Navier-Stokes numerical analysis tool developed to study high-speed ducted propfan aircraft propulsion systems. The program utilizes a three-dimensional, time-marching numerical procedure along with a flexible, coupled 2-D/3-D multiple block geometric grid representation to predict the flow field in and around the fan. Multiple runs of ADPAC are needed to create the single-curve performance map used in the zooming strategy.

3. Intelligent Monitoring and Control

A complete engine simulation that includes high-fidelity components is characterized by a corresponding high degree of complexity. In addition, the presence of high-fidelity components can greatly increase the running time of simulations. The user of such a system needs a high

Typically, the averaged stagnation pressure ratio will not initially match the low-fidelity simulator value. The three-dimensional boundary condition representations can then be redefined and the above process repeated until the necessary match is found. In practice, this iterative approach to boundary value matching was found to be computationally unstable, requiring many iterations to achieve a balance. Worse, the iterative approach can lead to an oscillatory mode where convergence is never achieved.

A solution which eliminates the iterative nature of the process is the construction of a performance map from multiple runs of the three-dimensional component. A single-curve performance map, such as that shown in Figure 3, is constructed and the appropriate value can then be chosen from the map, interpolating as needed. To shorten the overall time for the simulation, the multiple runs can be performed in parallel when the necessary computational resources are available.

2.2 Simulation Tools

Turbofan Engine System Simulator. TESS is an object-based, one-dimensional, transient, thermodynamic aircraft engine simulator which runs under AVS [1]. This integrated system provides the graphical user interface and operating environment for construction of arbitrary engine configurations, selecting and controlling steady-state and transient engine operation, and graphical display of simulation results.

The Network Editor of AVS provides a visual interface for creating dataflow programs. For TESS, the dataflow is used to model the flow of air through the engine. Engine components (e.g., compressor, turbine, duct, etc.) are represented graphically as AVS module icons, or simply *modules*. Each module has a control panel where the operational characteristics of the engine component are defined by the user (e.g., the mass flow rate, design point performance data). An engine is created by selecting the modules needed and placing them in the work space of the Network Editor. The dataflow network is then created by connecting the modules to establish the physical connections of the engine. The left side of Figure 1 shows a portion of the TESS network used in testing the simulation system. Figure 4 shows a complete TESS engine network that models a two spool, two stream turbofan engine.

Once all of the components have been connected and their operational parameters entered, the user selects the length of time for the transient, and defines how the governing equations are to be

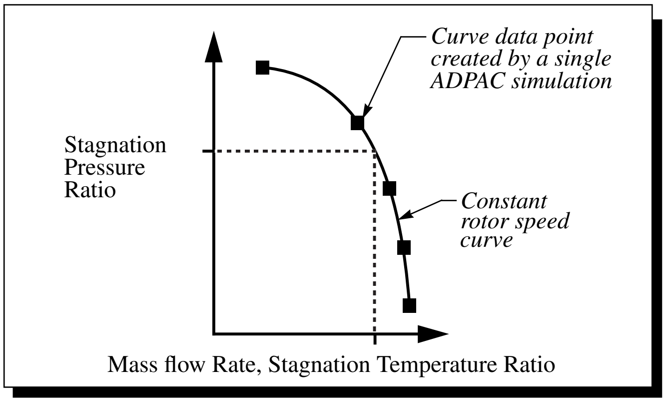


Figure 3: Single curve map for fan component

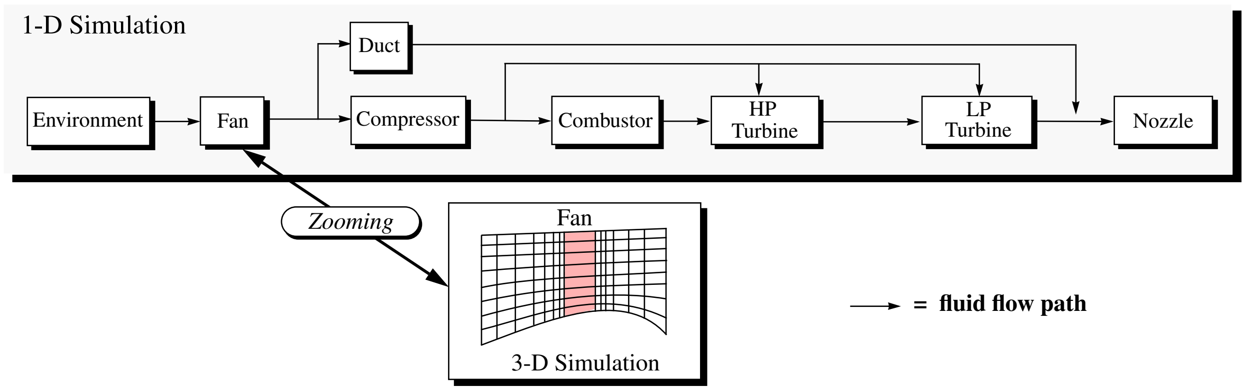


Figure 2: System schematic representing *zooming* on the fan component

2.1 Zooming

Understanding the physical operation of a jet engine requires simulation analysis at a sufficient level to fully reveal these processes. For example, a detailed analysis in the fan component of the engine can help resolve fan tip clearance problems and compressor stall problems due to fan tip rub. Because of the complex interactions between the various engine components, as well as between disciplines (heat, fluid, structural, etc.), this often requires a high-fidelity (more detailed) simulation. Three-dimensional (3-D), time-accurate computational fluid dynamics (CFD) solutions are capable of providing the necessary detail but are computationally intensive. The simulation of a complete engine at such a high level of fidelity requires computing resources that are currently unavailable. Zooming, the integration of simulation codes that model at differing levels of fidelity into a single simulation, provides the opportunity to reduce the overall computing effort needed while retaining the desired level of analysis in specific engine components. This idea is depicted in Figure 2. Here the fan component of a one-dimensional “baseline” engine model has been “zoomed” to a three-dimensional analysis.

Implementation of the zooming concept is difficult, due mainly to the inability to accurately resolve high-fidelity data fields from a single value as supplied from the low-fidelity system simulator. For the zooming to be accurate, the upstream and downstream boundary values (which are single-valued), must be extrapolated to define a suitable three-dimensional distribution of field variables such that when integrated over, the original single-valued boundary conditions are recovered.

For the current work in which the fan component is zoomed, this process begins with the single inlet boundary values for stagnation pressure, stagnation temperature, and Mach number, and the exit boundary value of static pressure from the fan component of the one-dimensional engine model. These are then extrapolated to appropriate three-dimensional field distributions and applied as boundary conditions to the fan simulation. The results of the fan 3-D simulation are then integrated to determine the mass-flow rate, and the mass-averaged values of outlet/inlet ratios for the stagnation pressure and stagnation temperature. The averaged stagnation pressure ratio is then compared with the stagnation pressure ratio computed across the engine model. If the values are identical, then the extrapolated field distributions are proved to be suitable representations and the averaged values of mass-flow rate and stagnation temperature ratio may be used in the one-dimensional simulation.

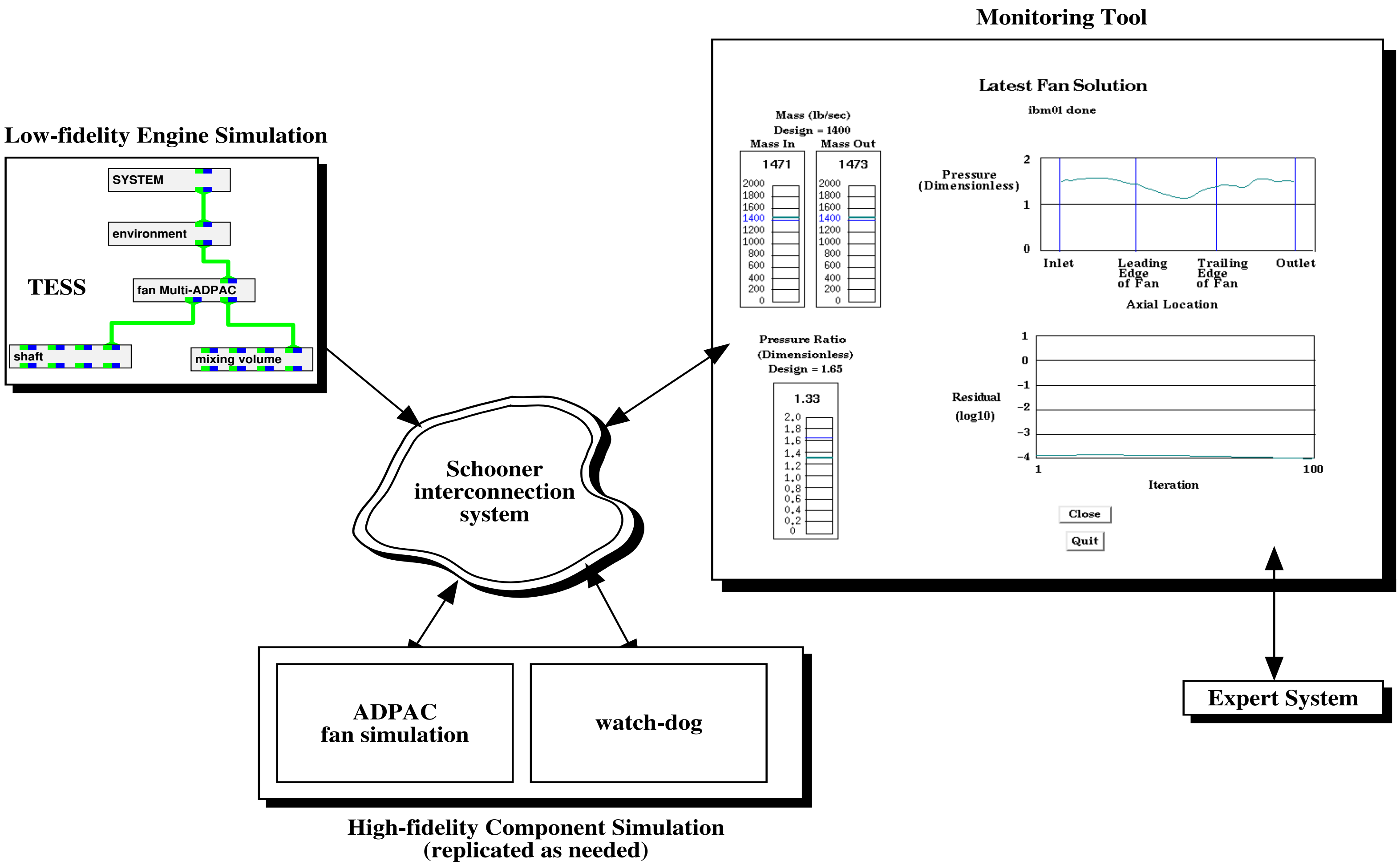


Figure 1: Prototype simulation system

currently provides warnings and errors to the user and will be able to actively steer the engine simulation. The Schooner interconnection system [13, 14] provides the software framework to connect the various tasks and solves the heterogeneity and configuration issues that arise. Figure 1 illustrates the software structure of the prototype simulation system implemented in this project.

This paper is organized as follows. Section two describes an engine model that demonstrates zooming on the fan component of a turbofan engine. Section three describes the design of the monitoring tool and expert system portions of the prototype. These assist the user in executing the simulation and will be used to explore techniques for intelligent control. Section four describes the Schooner interconnection system that integrates the components and gives details on the construction of the prototype simulation system. Section five describes how a specific engine has been implemented on the prototype system. Section six offers some future directions for this project.

2. Simulation Strategy

Modern commercial aircraft use large turbofan jet engines as their main propulsion systems. In these types of engines, air enters the engine, passes through a fan, and is accelerated to a velocity slightly higher than the surrounding air. A small percentage of this airflow is diverted into the core section of the engine which contains the compressor, combustor and turbine components, while the remaining air is bypassed around the core and exits the engine. The resulting increase in momentum of the bypass airstream accounts for a large percentage of the engines total thrust.

Over the last 25 years, turbofan engine size and performance have increased considerably. For example, in 1970, the Pratt & Whitney JT9D, the first high-bypass commercial turbofan engine, had a 92 inch diameter fan and provided 43,600 pounds of thrust [2]. By way of comparison, the 1995 Pratt & Whitney PW4084 engine employs a 112 inch diameter fan and produces 84,600 pounds of thrust [16]. Enlarging the fan diameter to increase thrust, however, can introduce new problems: as fan blades become longer, they are subject to more aerodynamic stresses which can result in blade bending and twisting. This affects the aerodynamics of the fan blade and can cause difficulties in fan-tip clearance with the surrounding cowling, possibly causing engine stall during flight. To investigate these problems, engine designers must either build scale prototypes and test these models, or perform numerical simulations of the fan. The first approach is becoming increasingly expensive, making numerical simulations of engine components a frequently used design tool. Often a component simulation of the fan is satisfactory in determining performance characteristics in the component. However, because the effects of engine components downstream of the fan (compressor, bypass duct, etc.) can affect the flow conditions in the fan, effects from these components must be considered.

For this reason, using an isolated simulation of the fan alone may not be sufficient. The remaining engine components must be included in the simulation to account for inter-component effects. In this section, we describe how an alternative simulation strategy known as *component zooming* provides a methodology for investigating these inter-component effects. First, we describe problems that arise in attempting to implement zooming, and outline the methods adopted to overcome these problems. Then, we describe TESS and ADPAC. TESS is the low-fidelity engine simulator, and the Advanced Ducted Propfan Analysis Code (ADPAC) [10] is the high-fidelity fan flow-analysis package used in the prototype zooming system.

Implementing Monitoring and Zooming in a Distributed Jet Engine Simulation

1. Introduction

Designing and implementing new propulsion technologies can be an expensive and time-consuming process. The Numerical Propulsion System Simulation project, sponsored by NASA Lewis Research Center, is applying new computer simulation techniques and parallel hardware to this problem [5, 6]. Specifically, it is fostering the development of parallel simulations to improve both the execution time and accuracy of the simulations. NPSS is also developing a simulation executive to support complete engine simulations constructed from the improved component simulations. Research on the simulation executive includes developing the monitoring and control techniques needed to manage the simulation, and exploring the use of expert system techniques to assist the user in controlling the simulation.

Several key issues must be addressed in the design of the propulsion system simulation. One is the integration of simulation codes at different levels of fidelity. Low-fidelity (less detailed) modeling requires empirical data that are not available at the preliminary design stage. On the other hand, high-fidelity (more detailed) modeling overcomes this limitation, but at a substantial computational cost. A concept being developed to overcome these limitations is *zooming*, which allows selected components to be modeled in detail and integrated into a low-fidelity engine simulation. Additionally, during a low-fidelity simulation, zooming provides a means of selectively examining in detail the physical processes within components of the engine.

A second issue is monitoring and control. A monitoring tool allows the user to observe the progress of the simulation through displays of its key parameters. An expert system can further improve the simulation by continuously monitoring and actively steering the simulation. This requires support in two areas. The first is the collection of knowledge and the formulation of rules that govern the design and operation of jet engines. The second is the integration of expert system software into the simulation executive to assist the user in executing the simulation.

A third issue in the overall design is heterogeneity. The engine component codes, monitoring tool, and expert system take advantage of a variety of workstation, vector and parallel platforms. Additionally, they employ a variety of programming models and languages. Combining these results in a *heterogeneous distributed program*, or *meta-computation* [15]. A software interconnection system addresses the heterogeneity issue and facilitates the construction of meta-computations by providing connections among the software components and implementing a configuration and control system.

This paper describes a prototype simulation system designed to address all three issues: zooming, monitoring and control, and interconnection support. The prototype employs the Turbofan Engine System Simulation (TESS) [18], which provides a low-fidelity model of a complete jet engine. In this model, the operational characteristics of the individual engine components are supplied in the form of performance maps that are constructed from experimental data. To provide descriptions of the physical processes occurring in an engine component beyond that supplied by a performance map, a high-fidelity component simulation is used. The simulation executive uses a monitoring tool that provides information about the high-fidelity component simulation to the user and the expert system. Based on this information, the expert system

Implementing Monitoring and Zooming in a Distributed Jet Engine Simulation

Abdollah A. Afjeh[*], *Patrick T. Homer*[†], *Henry Lewandowski*[‡],
John A. Reed[*], *and Richard D. Schlichting*[†]

Cleveland State University[‡], The University of Arizona[†], University of Toledo[*]

TR 96-13

Abstract

The NASA Numerical Propulsion System Simulation (NPSS) project is exploring the use of computer simulation to facilitate the design of new jet engines. Several key issues raised in this research are being examined in an NPSS-related research project: zooming, monitoring and control, and support for heterogeneity. The design and implementation of a distributed simulation executive that addresses each of these issues is described. In this work, the strategy of zooming, which allows codes that model at different levels of fidelity to be integrated within a single simulation, is applied to the fan component of a turbofan propulsion system. A prototype monitoring and control system supports experimentation with expert system techniques for active control of the simulation. An interconnection system provides a transparent means of connecting the heterogeneous systems that comprise the prototype.

August 7, 1996

Department of Computer Science

The University of Arizona

Tucson AZ 85721

Replaces TR 94-25 entitled “Development of an Intelligent Monitoring and Control System for a Heterogeneous Numerical Propulsion System Simulation”