Using Java To Develop Educational Engineering Software

John A. Reed, **Abdollah A. Afjeh**
The University of Toledo

Introduction

One of the most exciting recent developments in software technology is Java™, the programming system developed by Sun Microsystems Inc.[1] Since its introduction in early 1995, both the technical and mainstream press have been filled with articles about how Java will revolutionize the nature of the World Wide Web (WWW), client/server application development, and the economic model for software delivery. With all of the media hype, it is easy to lose sight as to what Java is, and why it is so interesting. For many people, Java is simply a way to add "flash" to otherwise static WWW pages. However, what is truly exciting about Java is that, for the first time, it is possible to write highly interactive, graphical applications which are platform-independent and can be transported across the WWW. These features, combined with the availability and pervasiveness of the Internet and WWW, make Java an attractive tool for developing and distributing educational software.

Background

To understand the potential benefits of Java for educational software development, it is necessary to understand some of Java's main features. First of all, Java is a general purpose object-oriented (OO) programming language offering OO capabilities such as data abstraction, encapsulation, polymorphism, and inheritance[2]. Object-oriented programming makes it possible to write robust, modular code which is easily modified and extended. Java was designed to be syntactically similar to C++, which is currently the most popular OO language. However, it was designed to be simpler to learn than C++, and thus removes many of C++'s shortcomings which make it complex and confusing. In addition, Java adds many of the better object-oriented features available in other object-oriented languages which are missing in C++. The most notable enhancements are the elimination of pointer-arithmetic, and the addition of automatic memory management (garbage collection).

The most significant feature of Java is the ability to create extremely portable applications. Unlike traditional programming languages which generate programs which are platform specific, Java code is compiled into an architecturally neutral file format (byte-codes) which permits a compiled Java program to be transported to platforms of differing architecture. The Java program can then be run on any computer which implements a Java interpreter and run-time system known as the Java Virtual Machine (JVM). The use of byte-coding and implementation of the JVM allows a Java program to achieve a high degree of portability: a Java code, compiled into byte-codes is portable to any machine which implements the JVM. This makes it possible to write a program once, without regard to a specific platform, and run it on any computer platform which implements the JVM.

The use of byte-codes and the virtual machine have made possible the development of Java *applets*: small Java programs which are transmitted across the WWW and run on the receiving computer. Applets bring the interactive capabilities of traditional desktop applications to the WWW. Furthermore, the platform independent nature of Java applets make it possible to develop, maintain and distribute a single version of a code from a WWW server, instead of multiple versions for each computer architecture and operating system combination.

Applets are typically embedded in a WWW page, and execute within a Java-enabled browser. When a page containing an applet is retrieved, the applet's byte codes are transferred across the network from the server to the student's computer. On the student's computer, the applet's byte-code is automatically loaded and run by the Java interpreter built into the web browser. The browser implements the Java runtime interpreter as well as graphical support for the applet. Because applets (generally) load from machines on the network, they are restricted from implementing certain functions for security reasons. For example, applets may not read, write or modify files on the local (client) machine. This non-secure state is indicated by the "Unsigned Java Applet Window" message displayed in the bottom of any applet graphical window (see any of the figures in this paper for an example).

Java and Education

The ability to construct interactive programs which are easily distributed across the WWW, make Java of considerable interest to educational software developers. Java-enabled browsers, such as Netscape Navigator™ or Microsoft Internet Explorer™, are now common in educational institutions and provide students with easy access to applets from any type of computer. With Java it is possible, for example, to enhance courseware through animations, present virtual lab exercise, or develop and distribute tutorials as easily as distributing standard WWW pages. This capability has not gone un-noticed in the educational community. In the short period since Java's release, over 600 educational applets have already been developed. At the time of this writing, nearly 100 engineering-specific educational applets have been listed at *Gamelan*, the official directory for Java applets[3]. These applets cover topics in all engineering disciplines, and although the majority of these are relatively simple and problem-specific, several sophisticated applets have been developed which demonstrate the effectiveness of using Java to develop educational software. Some notable example are listed as follows: The *Java Virtual Wind Tunnel* applet, developed by David Oh demonstrates two-dimensional implementation of computational fluid dynamics, including visualization of flow over solid objects[4]. The *Digital Simulator*, provides the ability to build and test digital circuits[5]. The *Ideal Flow Machine* and *Ideal Flow Mapper* applets, developed by William Devenport, are educational applets which allow the user to interactively create and visualize ideal flow[6]. The *Java Beam* allows students to investigate mechanical design of supported beams using an interactive and configurable interface[7].

In this paper, we describe our work in developing a Java applet which is to be used for instructing graduate and undergraduate students in the operation of gas turbine engine. The complex operating nature of the gas turbine system makes it a challenging subject to teach to students. The strongly-coupled nature of the engine's flow physics (and thus equations), combined with the large number of engine parameter variables, form an often large, non-linear system of equations which the student must solve. Consequently, computer programs capable of simulating gas turbine engines are usually provided to reduce the amount of hand-calculations needed. These

programs solve the problem at hand, but do not necessarily add to the student's knowledge of how the gas turbine operates. The gas turbine simulator described in the remainder of this paper, is aimed at providing an integrated environment for more effectively illustrating the operation of the gas turbine engine.

Java Gas Turbine Simulation Software

The Java Gas Turbine Simulator applet provides an interactive graphical environment which allows the rapid, efficient construction and analysis of arbitrary gas turbine systems. The simulation system couples a graphical user-interface, developed using the Java Abstract Window Toolkit, and a transient, space-averaged, aero-thermodynamic gas turbine analysis method entirely coded in the Java language. The combined package provides analytical, graphical and data management tools which allow the student to construct and control dynamic gas turbine simulations by manipulating graphical objects on the computer display screen. The simulator, running as a Java applet, can be easily accessed and run from a variety of heterogeneous computer platforms, including PC's, Macintosh™, and UNIX™ machines, through the use of Java-enabled WWW browsers.

The gas turbine analysis model used in the Java Gas Turbine Simulator is derived from the NASA DIGTEM code[8]. A complete description of the model can be found in ref. 9. In this model, the gas turbine system is decomposed into its individual components: inlet, compressor, combustor, turbine, nozzle, bleed duct connecting duct, and connecting shaft. Intercomponent mixing volumes are used to connect two successive components as well as define temperature and pressure at component boundaries. Operation of each of the components is described by the equations of aero-thermodynamics which are space-averaged to provide a lumped-parameter model for each component. For dynamic (transient) gas turbine operation, the model includes the unsteady equations for fluid momentum in connecting ducts, inertia in rotating shafts, and mass and energy storage in intercomponent mixing volumes. Overall performance maps are used to provide accurate steady-state representations of compressor and turbine component operation. Variable geometry effects in the compressor are accounted for using baseline maps that correspond to nominally scheduled geometry. Variable geometry maps are then used to bias the baseline map outputs by functions of the actual geometry.

In the Java Gas Turbine Simulator, the object-oriented programming features of the Java language are used to develop a digital representation of the gas turbine engine analysis model. Each of the components listed above are represented as objects in the simulator. Each component's characteristics, such as its data (e.g., performance maps) and methods (e.g., the mathematical equations used to describe its operation) are encapsulated within each object. The object-oriented nature of the Java language allows the program to model any gas turbine engine by combining specified types of component objects in almost any order. Currently, the following engine component types are available: AeroMixingVolume, BleedDuct, BleedCooledTurbine, Combustor, Environment, FuelSource, Nozzle, RotorShaft, StoredMassDuct, and VariableCompressor.

When the individual components in the gas turbine system are combined, each StoredMassDuct, RotorShaft, and AeroMixingVolume contributes its respective unsteady equation to form a system of unsteady ordinary differential equations. This system may then be solved using standard

numerically techniques such as Newton-Raphson, Runge-Kutta, etc. In the Java Gas Turbine Simulator, the process of combining components to form a gas turbine model is performed graphically by the student. Several numerical solvers are built into the system and can easily be selected by the student to solve the system of equations. This process will be illustrated in more detail in the next section.

Conducting a Simulation

Students begin a simulation by using a Java-enabled WWW browser to connect to the Java Gas Turbine Simulator webpage located on The University of Toledo Mechanical, Industrial & Manufacturing Engineering (M.I.M.E.) department web-server. This page contains the embedded Java Gas Turbine Simulator applet. In addition to the applet, the student may access additional information about the simulator using the hypertext links in the webpage. These links include access to a Java Gas Turbine Simulator tutorial, other publications on the simulator and related subjects, and information on the development of the Java Gas Turbine Simulator. The student displays the simulator by selecting the "Display Simulator" button, located within the web page. This action initiates the loading of the applet byte-codes to the browser's Java interpreter. When the loading is completed, the Java Gas Turbine Simulator's *Main* window (see Figure 1) is displayed on the computer screen. From the *Main* window, the student can access the various windows of the simulation system: *Engine Schematic Layout*, *System Control Dialog*, *Graphing*, *Transcript* and *Help* windows. Selecting the *Exit* button closes all opened windows except *Main*. The *Main* window is closed by selecting the "Dispose Simulator" button.

The graphics and windowing components which make up the Java Gas Turbine Simulator user-interface are built using the Java Abstract Window Toolkit (AWT)[10]. The AWT, which is part of the Java runtime system, provides a collection of platform-independent components for building graphical applications in Java. The AWT offers support for graphics operations, as well as supplying common graphical user-interface objects such as Buttons, Lists, TextFields, Choice boxes, etc. Platform independence is achieved through the use of *Peers* which are native GUI components manipulated by the AWT. Peers allow Java programs that use the AWT to retain the familiar look and feel of the host computer's native windowing system. This means that when a Java program is run on a Macintosh computer, the Buttons, Windows, List boxes, etc., all appear in the familiar Macintosh style. When the same program is run on a computer running Windows NT™, those components will appear as familiar Windows NT-style objects. To demonstrate, this, the Java Gas Turbine Simulator was run on several different computers using the Netscape Navigator browser and color images of the various windows were captured and are reproduced here. Figures 1, 2, and 7 are Motif™ graphics; figures 3 and 5 are Windows NT-style; and figures 4 and 6 are Macintosh-style.

Figure 1: *Main* window

Visual Construction of the Model. A gas turbine simulation model is developed by building a schematic representation of the gas turbine in the *Engine Schematic Layout* window (see Figure 2). Individual engine components are represented graphically as icons with each class of engine object (e.g., BleedDuct, VariableCompressor, etc.) having a uniquely shaped and colored graphical representation. The student

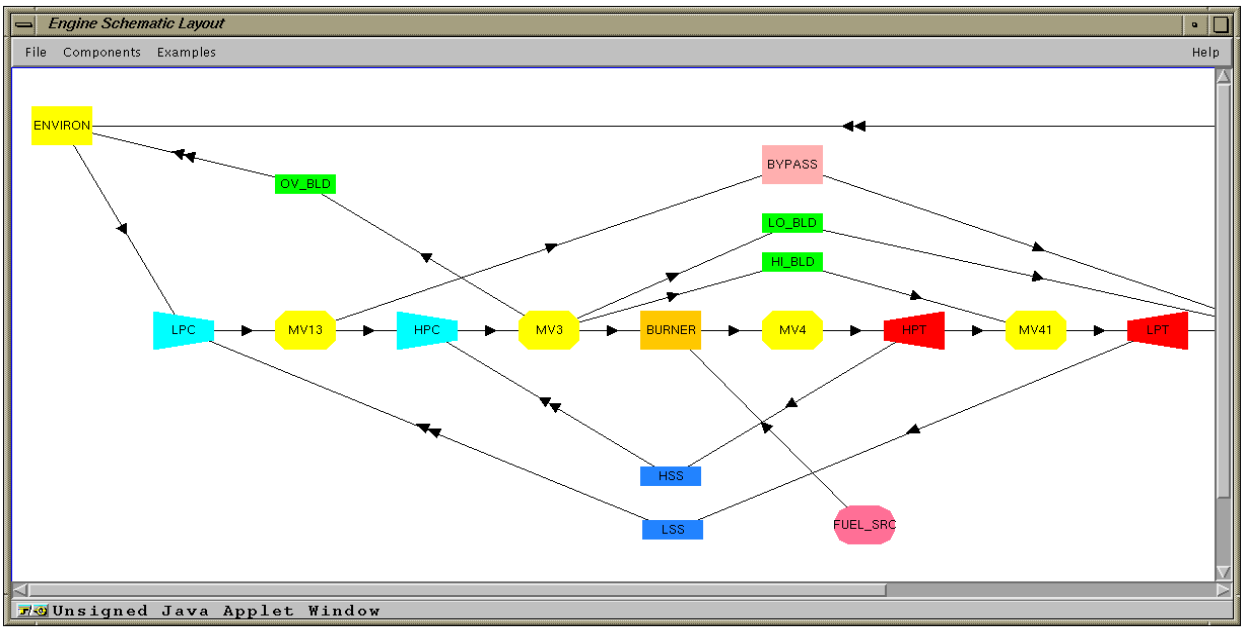

Figure 2: *Engine Schematic Layout* window

selects engine components to add to the model from the `Components` menu located in the *EngineSchematicLayout* menu bar. The component's icon is displayed in the work area and a Dialog window for that engine object is also displayed. The engine component's dialog window provides the means for the student to define the operational characteristics of the component (i.e., the component name, design- and initial-operating point performance data, etc.). For example, Figure 3 shows the dialog window for an `AeroMixingVolume` component. Similar dialogs, with appropriate design- and operating-point parameters, are used for each of the other engine components. Additional engine components are added as needed to complete the engine model. Once all of the icons have been placed on the schematic work area, they may be dragged into place and interconnected to create the engine schematic diagram. In the diagram, the arrow-headed connecting lines represent both the directional flow path for fluid through the engine, and the structural connections along which mechanical energy is transmitted. Figure 2 shows the *Engine Schematic Layout* window with a typical gas turbine engine model.

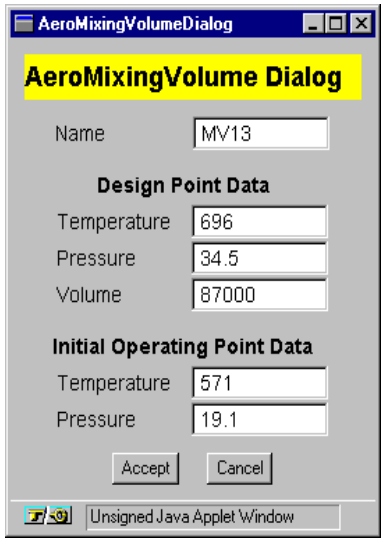

Figure 3: *AeroMixingVolumeDialog* window

Graphing. For information on the physical processes occurring in the gas turbine, the student may graph various component parameters during the transient. Pressing the `Graphing` selection button on the *Main* window displays the *Graph Control Dialog* (see Figure 4). As its name suggests, this dialog provides the student with control over parameters to be graphed during the transient portion of the simulation. From this dialog, the student may select to graph a number of specified parameters for any of the components currently displayed in the *Engine Schematic Layout* window.

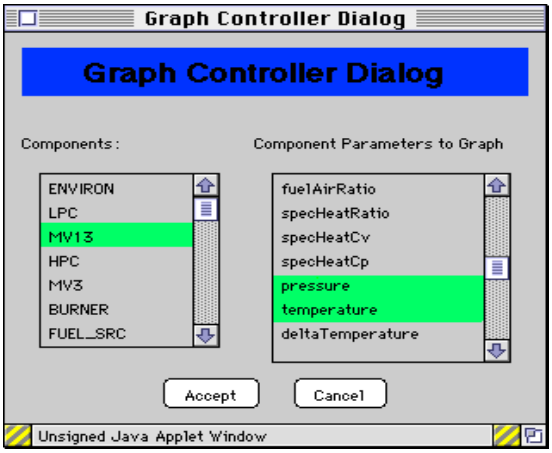

Figure 4: *Graph Control Dialog* window

The *Graph Control Dialog* is comprised of two List components. The left-hand List displays each of the engine components currently displayed in the *Engine Schematic Layout* window. The right-hand List displays parameters to be graphed for the selected engine component. This List allows multiple selections, allowing the student to select to graph any of the component's parameters. To illustrate its use, consider the following example as depicted in Figures 4 and 5. Here the student has selected the component named `MV13`. This component is an `AeroMixingVolume` object and as such has graph parameters of `Volume`, `Stored Mass`, `Temperature`, etc., which are shown in the right-hand List. From that List, the student has selected to graph the `Temperature` and `Pressure` parameters during the transient. Figure 5 shows the `Temperature` and `Pressure` graphing windows which are displayed during the transient. Note that the student has also selected to graph the `Temperature` and `Pressure` parameters of the `MV3`, `MV4`, and `MV41` components for comparison.

The *Graph Control Dialog* may also be used after a simulation to view other parameters not graphed during the simulation. The *Graph Control Dialog* object acts as a database for each of the parameters listed in the *Graph Control Dialog*, and as such acts as a post-processor, displaying each of the listed parameters after the simulation has completed.

Selecting Numerical Solvers. Once all of the gas turbine engine components have been connected and their input data entered, the student may define the system solvers and transient duration. In a transient analysis, the gas turbine system is settled (balanced) at or near the user defined initial operating point before beginning the transient. Currently, two steady-state

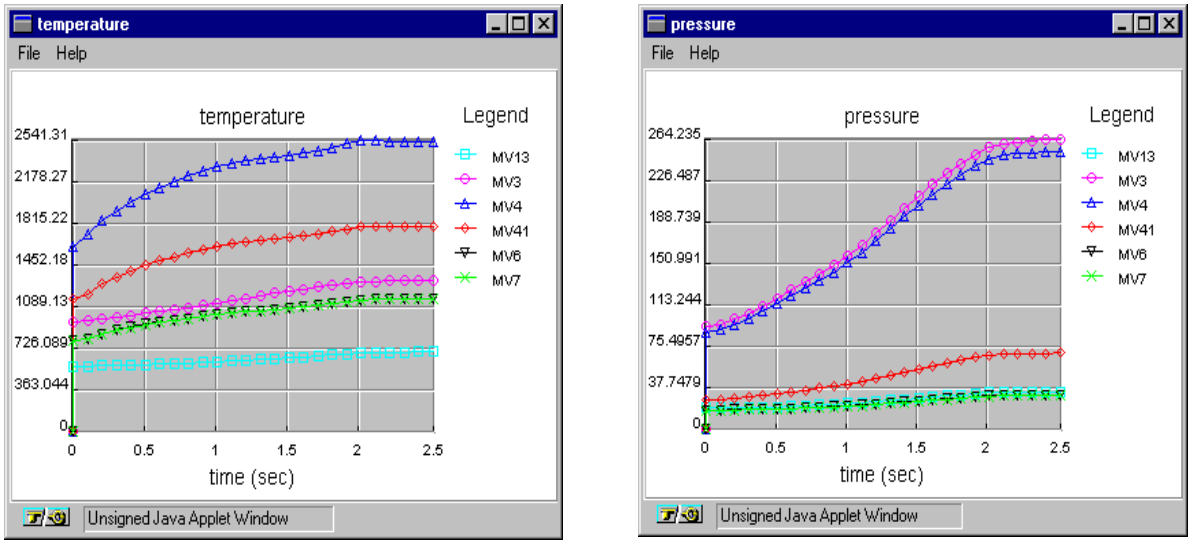

Figure 5: *Transient Graph* windows

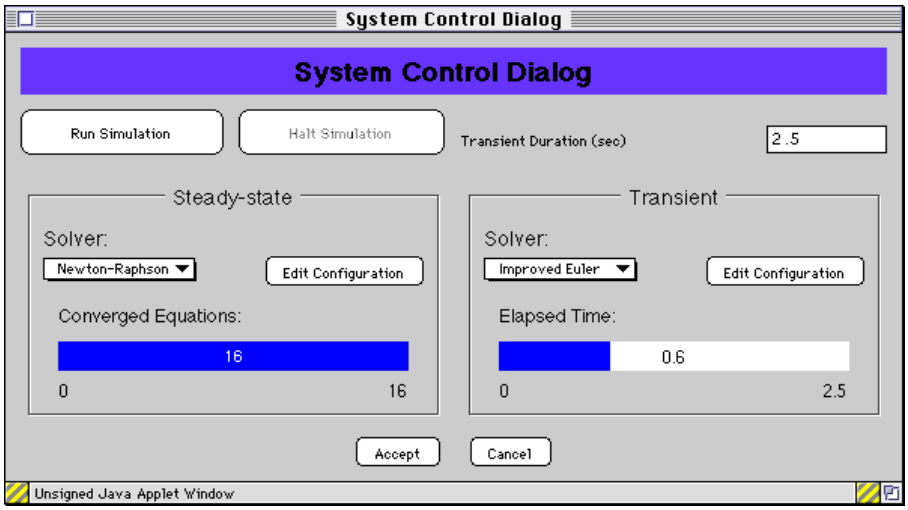

Figure 6: *System Control Dialog* window

balancing methods are implemented: `Newton-Raphson` and `Fourth-order Runge-Kutta`. Four transient integration methods are available: `Improved Euler`, `Fourth-order Runge-Kutta`, `Adams`, and `Gear`.

The *System Control Dialog* window (see Figure 6), which provides controls for the overall operation of the simulation, is accessed by depressing the `System Controllers` button in the *Main* window (see Figure 1). The steady-state numerical solver used to balance the gas turbine equations at the initial operating point is selected from a list of available solvers. This list is displayed using a Choice component which shows the current list selection. In Figure 6, the `Newton-Raphson` method has been selected as the steady-state solver. Similarly, the transient solver may be selected from the list of available transient solvers. In Figure 6, the `Improved Euler` method has been selected. The student may edit either of the selected solver's parameters by depressing the steady-state or transient `Edit Configuration` buttons. A *Transient Solver Dialog* (not pictured) for the selected method is displayed. Using this dialog, the student can define specific control values for the solver's operation. Similar dialogs are available for each of the available solvers, and selecting a different solver from the list will bring up different control parameters specific to the newly selected solver.

Below both the steady-state and transient solver selectors are the steady-state and transient *Progress Indicators*. These indicators act as gauges which provide visual feedback to the user during the simulation indicating the progress of the steady-state balancing or transient integration processes. The steady-state *Progress Indicator* displays the number of equations which have converged to steady-state. The transient *Progress Indicator* displays the elapsed time of the transient. For example, in Figure 6, the `Converged Equations` *Progress Indicator* shows that all 16 of the system equations have converged, and that, as shown by the `Elapsed Time` *Progress Indicator*, the transient simulation has progressed to 0.6 of a 2.5 seconds-long transient.

Running the simulation. After the engine configuration has been constructed, graph parameters selected, and the steady-state and transient solvers have been established, the student can begin the simulation by pressing the *Run Simulation* button located in the *System Control Dialog* window. The simulator first attempts to determine the steady-state (balanced) condition at the

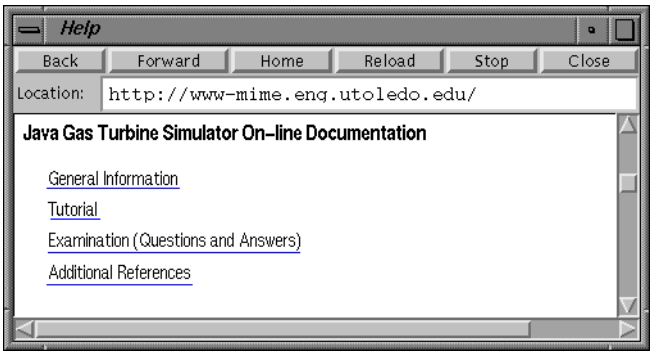

Figure 7: *Help Browser* window

initial operating point, as was defined by the student. Once the engine is balanced, the transient will begin. (If the student has defined a simulation ending time of zero in the *System Control Dialog*, the simulation will terminate giving only a steady-state solution.) The transient will run until the simulation time exceeds the `Simulation Duration` value entered in the *System Control Dialog*.

On-line help browser. The student may obtain additional information about the simulator by selecting the `Help` button from the *Main* window, or from the menu bars in other windows. The *Help* browser (see Figure 7) provides hypertext access to on-line documentation located on the web server using Java's built in support for WWW network access. From the browser, the student may access hypertext documents which describe the analysis theory used in the simulator, tutorials, and references which the student can use for further study. Another use of the *Help* browser is to provide student "examinations." Instructors can easily create hypertext files which contain questions (and answers) about gas turbine operation. Because the file is centrally located on the web server, there is only one file to edit, ensuring that all students have the same questions and simplifying the update processes. Although not implemented at this time, it is possible to create interactive forms which could be used to collect student responses for on-line grading.

Summary

The Java Gas Turbine Simulation software described in this paper provides an improved computer simulation environment for instructing students in gas turbine operation. As a Java applet, the simulator provides advantages over traditional simulation programs by allowing students access from any networked computer. This, combined with freely available Java-enabled WWW browsers capable of operating on a large number of differing computing platforms make the simulator widely accessible. The portability offered by the Java language and the platform independence of the AWT make Java an ideal language and runtime system for developing interactive educational software for use on networked, heterogeneous computer systems such as the World-wide Web.

The Java Gas Turbine Simulator provides advantages over traditional static gas turbine computer programs by allowing students to visually construct gas turbine models - a process which enhances the students appreciation of how a gas turbine is constructed. In addition, allowing the student to define the various engine component parameters aids in the identification of (1) information needed to carry out the simulation, and (2) variables that influence a component's performance. Moreover, it depicts how the governing equations are applied. Finally, simulation output in a visual format helps improve the student's understanding of how gas turbine components interact and how their parameters change during a transient.

This work also illustrates how programs written in Java address two common problem areas in educational software: 1) maintaining specialized software in a multi-platform environment, and 2)

software distribution. Universities, as well as students, often utilize a wide variety of computing platforms including PC's, Macintosh, and Unix computers. Until now, it has been difficult and costly to provide and maintain specialized versions of a given program for each platform. As hardware and operating systems are upgraded, the software must be updated and re-installed. As a result, in most cases, a program for a specific platform is presented and the student is forced to work on that type of computer. The platform independent nature of the Java program and wide availability of Java-enabled browsers reduce the need for specialized versions of software as a Java program will run on any of the major platforms. The process of software distribution is also simplified as the Java program may be delivered easily across the Internet directly to the student's computer. There is no need to install the software, since Java programs are loaded automatically upon being downloaded to the student's computer.

Acknowledgments

We would like to acknowledge the NASA Lewis Research Center Computing and Interdisciplinary Systems Office for partial support of this work (Grant No. NCC-3-207). In particular, we would like to thank Greg Follen for his continued support. J. A. Reed is partially supported by a University of Toledo Doctoral Fellowship. For directions on accessing the Java Gas Turbine Simulator applet, send e-mail to jreed@top.eng.utoledo.edu.

Bibliography

[1] Gosling, J., Joy, B., and Steele, G., "The Java Programming Language (version 1.0)," Addison Wesley, July 1996.

[2] Booch, G, "Object Oriented Design with Applications," The Benjamin/Cummings Publishing Company, Inc., New York, 1991.

[3] On-line document. URL: http://www.gamelan.com/

[4] On-line document. URL: http://raphael.mit.edu/Java/

[5] On-line document. URL: http://www.lookup.com/Homepages/96457/digsim/load.html

[6] On-line document. URL: http://www.aoe.vt.edu/aoe/faculty/davenfac.html

[7] On-line document. URL: http://ecsel.engr.washington.edu/JavaBeam/beams/simpleSupport.html

[8] Daniele, C. J., Krosel, S. M., Szuch, J. R., and Westerkamp, E. J., "Digital Computer Program for Generating Dynamic Engine Models (DIGTEM)," NASA TM-83446, 1983.

[9] Reed, J., "Development of an interactive graphical propulsion system simulator," Master of Science Thesis, The University of Toledo, Toledo, Ohio (August 1993).

[10] Gosling, J., and Yellin, F., "The Java Application Programming Interface: Window Toolkit and Applets," Addison Wesley, June 1996.

Biography

JOHN A. REED received his B.S and M.S. degrees in Mechanical Engineering at the University of Toledo in 1989 and 1993, respectively. He is currently pursuing his Ph.D. degree at the University of Toledo, where he is a University Doctoral Fellow. His research interests include developing computer simulation frameworks for multi-disciplinary systems, distributed heterogeneous computing, and developing engineering education software systems.

ABDOLLAH A. AFJEH is Professor and Director of Graduate Studies in the Mechanical, Industrial and Manufacturing Engineering department at the University of Toledo. He received his Ph.D. in Engineering Science form the University of Toledo in 1984. He is a member of ASME, AIAA as well as honorary societies Pi Tau Sigma, Phi Kappa Phi, and Sigma XI.