Structural Optimization 14, 203–218 © Springer-Verlag 1997

Distributed computing for multidisciplinary design optimization using Java

J.C. Becker, C.L. Bloebaum and K.F. Hulme
Multidisciplinary Optimization and Design Engineering Laboratory (MODEL), Department of Mechanical and Aerospace Engineering, State University of New York at Buffalo, Buffalo, NY 14260, USA

Abstract The programming language Java (recently referred to as the computer language of the Web) offers substantial possibilities for the type of complex engineering problems typically encountered in multidisciplinary design optimization (MDO) problems. In order to demonstrate the potential uses of Java for MDO problems, this paper presents the development of the Web Interface for complex engineering design (WICkED) software, which simulates the convergence of a decomposed complex system in a distributed computing environment and computes the sensitivity derivatives of the system with respect to the independent input variables using the GSE method or the finite difference method. In this application, one computer is designated as the server and sends out required inputs to a number of client subsystems over the Internet. A number of client computers can connect to the server and then receive the inputs necessary to calculate the solution to their model. As the code necessary to solve the model already exists at the client, only the inputs have to be sent over the network. When the client has solved the calculation, it returns the results to the server which processes the result to produce new inputs.

WICkED is written entirely in the Java programming language which allows server and clients to exist on completely different computer types and in heterogeneous, distributed networks. A number of parametric studies on the behaviour of complex systems in a distributed environment are performed and the results are reported in this paper. This research serves to identify potential problems as well as advantages in using Java for MDO applications.

1 Introduction

Many large engineering design problems are very well-suited for a decomposition into a number of smaller and coupled subproblems (Sobieszczanski-Sobieski 1982). Often, complex problems inherently exist in a decomposed format, especially when several different disciplines are involved in the problem. For example, in the case of aircraft design, a number of design groups, such as aerodynamics, structures, and controls, are all involved in the design process. Each of these groups has a model which describes their particular subsystem in their own computer environment and most likely the computers of all the design groups are linked together in a local area network (LAN). When the decomposition concepts of multidisciplinary design optimization (MDO) are applied to such a system, the use of the distributed computer platforms to perform virtual parallel processing is desirable. However, the different computer languages and computers that exist within the various groups create substantial roadblocks to such an implementation. This work presents an approach to solve MDO problems in a distributed computing environment using the "language of the Web", Java (Flanigan 1996).

2 WICkED

2.1 Motivation

The primary objective of this work is to demonstrate the effectiveness of using the Internet as a communication tool by building a computer application that simulates the analysis and sensitivity processes required in multidisciplinary design optimization (MDO) problems. Also, some of the problems, as well as advantages, of using this type of computational infrastructure for MDO applications are explored.

As an example, consider a large aerospace company with the functional groups of aerodynamics, structures and controls. Each group has its own computer system, different types of hardware, and uses software that fits their individual needs. Each group has a model for their disciplinary subsystem on their own computer platform. When one of the departments proposes a change in their disciplinary design, the other departments have to recalculate the solution to their models with the new input values on their own computer systems. Then these new outputs are given in return to the other groups. This iterative design process continues and is considerably hampered by the missing intercomputer communication.

Previous work has largely depended on the message passing tool PVM (Parallel Virtual Machine) and has resulted in such frameworks as FIDO (Framework for Interdisciplinary Design Optimization) (Weston 1994). This work introduces a new concept that allows analysts and designers to use a network of heterogeneous computers to solve an MDO problem by effectively using distributed computing via the language Java.

2.2 Introduction

The Web Interface for Complex Engineering Design (WICkED) simulates the design process using distributed computing. One computer is designated to be the server and sends out the inputs to the subsystems over the Internet. A number of client computers can connect to the server and then receive the inputs necessary to calculate the solution to their model. The code necessary to solve the model already exists at the client. Hence, only the inputs must be sent over the network. When the client has solved the calculation, it returns the results to the server which then processes the result to produce new inputs for the next iteration.

Considering the preliminary conditions given in the motivation, it should be possible that clients and possibly the server exist on different computer environments. Taking this into account, Java was chosen as the Internet computer language.

2.3 The Java programming language

The most common programming languages for a software applications in the field of engineering are currently C, C++ and FORTRAN. In the past few years, there has been a growth of multiple incompatible hardware architectures, each supporting multiple incompatible operating systems, with each platform operating with one or more incompatible graphical user interface. In addition, the growth of the Internet and the World-Wide Web have introduced even new dimensions of complexity into the development process of software. Java was designed to meet these challenges of application development in the context of heterogeneous, network-wide distributed environments. Java enables the development of secure, high-performance, and highly robust applications on multiple platforms in heterogeneous, distributed networks.

One of the primary characteristics of the language is that it is highly object-oriented, but still relatively simple. Object-oriented means that the focus lies on the data in the applications and on the methods that manipulate the data. In an object-oriented system, a class is a collection of data and methods that operate on that data. Taken together, the data and the methods describe the state and the behaviour of an object. Classes are arranged in a hierarchy, so that subclasses can inherit behaviour from a superclass. Inheritance is one of the main features of object-oriented languages.

Java is a distributed language because it is specifically designed to support applications on networks. It supports several levels of network connectivity through a package of certain networking classes. It is an interpreted language and its compiler generates byte-code rather than native machine code. To actually run a Java program, the Java interpreter is used to execute the compiled byte-code. Java byte codes provide an architecture-neutral object file format. The code is designed to transport programs efficiently to multiple platforms. A great advantage is that a Java program can be run on any system that implements the Java interpreter and the run-time system, thereby allowing for heterogeneous systems. Collectively, the interpreter and run-time system implement a virtual machine called the Java Virtual Machine. The language is designed to write highly reliable, robust software. Java is more strongly typed than other languages, which allows extensive compile-time checking for potential type-mismatch and other problems, which makes it relatively robust. Java is also a multithreaded language, in that it provides support for parallel processing of multiple threads of execution. This is useful, for example, to run a graphical user interface and perform several calculations at the same time.

The Java programming language was developed by Sun Microsystems and was first publicly released in January 1996. The compiler and interpreter currently exist in their first versions for UNIX on Sun workstations, for Macintosh computers and for PC's. New versions of Java, as well as enhanced, just-in-time (JIT) compilers and Java implementations from other companies will be released in the near future, making the language even more widely used.

2.4 Description of WICkED

The objective of this work is to design and implement a system to support virtual parallel processing for Multidisciplinary Design Optimization problems on a network of computers. The system uses a server/client model as can be seen in Fig. 1.

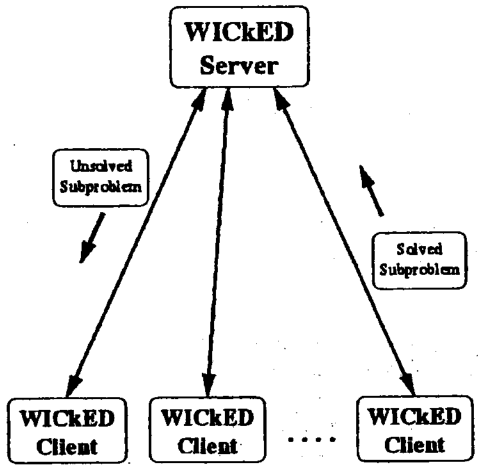


Fig. 1. WICkED system overview

The server creates subproblems which are part of a large decomposed problem. For this work, it is assumed that the problem is already decomposed into subproblems. The role of the clients is to solve the subproblems received from the server and to send back the subproblem solutions. The server also has to incorporate the returned subproblem solutions into the solution of the main problem. Only data (no code) will migrate on the system as it is running. Hence, when a node receives data (i.e. subproblem parameters or subproblem solutions), it already has the code to process that data. An overview over the construction of the server is given in Fig. 2.

The WICkED server includes a SystemManager and a ProblemManager. When a client connects to the server, the SystemManager allocates the connection to the ProblemManager, which then initiates a SessionManager to handle the connection. The SessionManager requests subproblems from its ProblemManager to send to the client and returns subproblems when they are received back from the client. The server is started by the ServerUserInterface which also has control over the settings for the server. The ProblemSolver can indicate the state of its operations through the display of the ServerUserInterface.

As seen in Fig. 3, the WICkED client has a SessionManager which receives subproblems from the server. Received subproblems are assigned to the ProblemSolver which creates a thread for the Subproblem to run in. Solved subproblems are retrieved by the SessionManager and sent back to the server. The ProblemSolver interacts with the ClientUserIn-

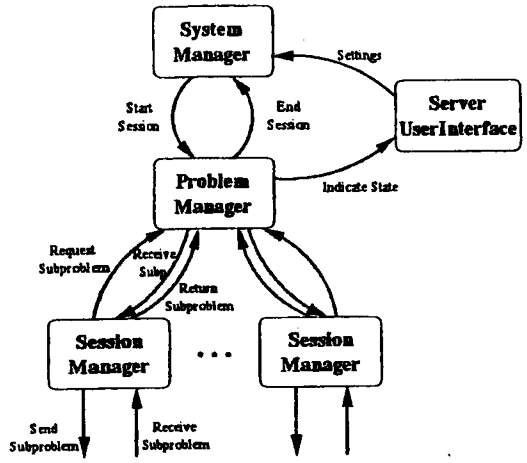


Fig. 2. WICkED server overview

terface which displays the progress as subproblems are solved. The ClientUserInterface can also order the ProblemSolver to stop processing subproblems and to close the network connection.

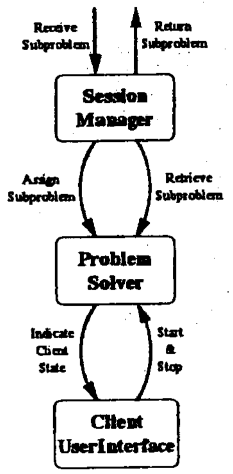


Fig. 3. WICkED client overview

WICkED uses the software Webcrunch by C. Daly which supplies the basic framework for solving problems using distributed computing. WICkED incorporates the Multidisciplinary Design Optimization problem into this framework and is designed to handle MDO problems which are already decomposed into subproblems. The graph in Fig. 4 illustrates how the general code is extended to handle a specific MDO problem.

Only three classes of the code (shaded in the graph) actually contain the specific application problem. The AbstractProblemManager of the server is extended by the WICkEDProblemManager which contains the starting values for the problem, a part which creates new subproblems when they are requested by the SessionManager, and a part which incorporates the returned solutions into the construction of the new problems. The WICkEDSubProblem contains information on how to send and receive subproblem inputs and results over the network and it also includes the algorithms

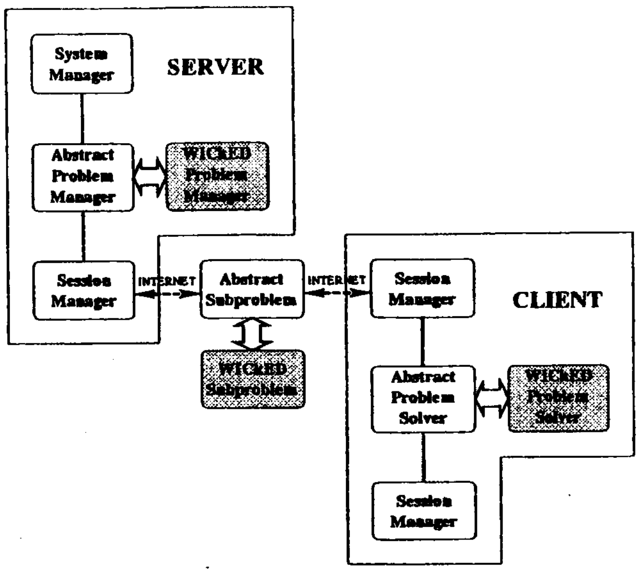


Fig. 4. WICkED system extension for MDO

which are necessary to actually solve the problems. The WICkEDSubProblem is known by both the server and the client. The server uses its methods to send input data to the client and the client uses the algorithm to solve the problem and then returns it to the server. The WICkEDProblemSolver uses the algorithms contained in WICkEDSubProblem as described above after it has identified the problem itself.

WICkED is used to solve a representative MDO problem in this work since we are particularly interested in the application of Java to this class of problems. An extremely important concern for MDO problems is the calculation of sensitivities, as this is typically the most computationally expensive parts of the MDO process. The next section gives an overview of the sensitivity analysis typically required for MDO problems.

When a new MDO problem is implemented in WICkED, the files WICkEdProblem.java, WICkEdProblemManager.java, and WICkEdProblemSolver.java must all be edited. These three files correspond to the shaded portions of Fig. 4. In WICkEdProblem.java, the constants numSS (the number of subsystems), numSSin (the number of subsystem inputs), and numSSout (the of subsystem outputs) must be changed in the class definition. The run method must contain the equations that solve the subproblems.

In WICkEdProblemManager.java, the contants described above must be changed. In addition, the initial values of the system output variables must be defined in the method *Initialize*. The method QuickSubProblemConstructor contains an algorithm that creates new subproblems and incorporates the solutions of solved subproblems in the convergence process. This should only be changed if the convergence algorithm is altered. If the user desires additional or different output data, it is in this file that such alterations must be made.

Changes to WICkEdProblemSolver.java must only be made if the problem name is changed. If changes are made to any of these three files, they must be recompiled to produce the updated class files.

3 Sensitivity analysis overview

The first step in the numerical analysis of a complex system typically includes a discretization of the equations into a model. The analysis generally requires the solution of algebraic equations, differential equations, or eigenvalue problems. Determination of the sensitivities required in the optimization process involves the mathematical problem of obtaining the derivatives of those equations with respect to their inputs.

Since the sensitivity analysis is typically the most expensive part of the optimization, it is therefore important that efficient approaches are used for this evaluation in the design process. There are a number of different techniques available, with one of the most popular being the finite difference approach due to its simplicity of execution. This approach is however extremely computationally expensive and can lead to errors when applied to complex systems. Other more traditional techniques that are commonly used are the analytical and semi-analytical approaches (Bloebaum 1989) . The global sensitivity equations (GSE) method (Sobieszczanski-Sobieski 1990) has significant advantages over these other methods for complex engineering problems.

As this work uses both the finite difference approach and GSE, both of these two methods are described in more detail in the following sections.

3.1 Sensitivity determination using the finite difference approach

The finite difference approach is one of the most popular techniques for the numerical determination of sensitivity information. It is easy to implement but it has the disadvantage that it is computationally very expensive and, further, accuracy problems may occur. The simplest finite difference approximation is the first-order forward difference approximation. Given a function $f(x)$ of the design variable x, the approximation $\frac{\Delta f}{\Delta x}$ of the total derivative $\frac{\mathrm{d}f}{\mathrm{d}x}$ is

$$\frac{\mathrm{d}f}{\mathrm{d}x} \sim \frac{\Delta f}{\Delta x} = \frac{f(x+\Delta x)-f(x)}{\Delta x}\,. \tag{1}$$

The flow chart for the finite difference approach can be seen in Fig. 5a. The finite difference approach requires that each design variable is perturbed separately by some prescribed amount. Then the whole system must be converged again and the function values associated with the perturbation are then calculated, after which the approximation of the total derivative can be calculated. Accuracy problems associated with this formulation are due to truncation and condition errors. Also, it is possible that the effect of a small perturbation may be lost when filtered through a large set of system analyses in MDO applications. If large perturbations are used, it is possible that nonlinearities of the system lead to imprecise sensitivity information. This approach is so expensive, as far as computation time is concerned, because the system has to be reconverged after each perturbation of a variable.

3.2 Sensitivity determination using the global sensitivity equations method

The global sensitivity equations (GSE) approach is also a methodology for obtaining the total sensitivity of the output response quantities of a coupled system with respect to the system design variables and was introduced by Sobieszczanski-Sobieski (1990). This approach defines the total derivatives of the output quantities in terms of the local sensitivities, which are partial derivatives of each subsystem's output with respect to its input. The global sensitivity equations for a two subsystem coupled problem can be written

$$\begin{bmatrix} \mathbf{I} & -\frac{\partial \mathbf{Y}_A}{\partial \mathbf{Y}_B} \\ -\frac{\partial \mathbf{Y}_B}{\partial \mathbf{Y}_A} & I \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}\mathbf{Y}_A}{\mathrm{d}\mathbf{X}_A} & \frac{\mathrm{d}\mathbf{Y}_A}{\mathrm{d}\mathbf{X}_B} \\ \frac{\mathrm{d}\mathbf{Y}_B}{\mathrm{d}\mathbf{X}_A} & \frac{\mathrm{d}\mathbf{Y}_B}{\mathrm{d}\mathbf{X}_B} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathbf{Y}_A}{\partial \mathbf{X}_A} & 0 \\ 0 & \frac{\partial \mathbf{Y}_B}{\partial \mathbf{X}_B} \end{bmatrix}. \tag{2}$$

The first matrix in the equation is the global sensitivity matrix (GSM), which is the matrix of the partial derivatives of all output equations (i.e. $\mathbf{Y}_A, \mathbf{Y}_B$) with respect to all other output equations. These sensitivities represent the couplings between interacting subsystems and can be computed within each subsystem, eliminating the need to perform computationally expensive interdisciplinary calculations. The dimension of this matrix is $n \times n$, where n is the total number of output equations in the system. The matrix on the right-hand side of the equation is the matrix of partial sensitivities of all system outputs with respect to all design variables (i.e. $\mathbf{X}_A, \mathbf{X}_B$). The dimension of this matrix is $n \times m$, where m is the number of system design variables. The middle matrix is the desired matrix of the total derivatives. These derivatives provide an indication of how a change in one or more design variables will affect all of the system outputs.

The GSE method is implemented as shown in the flowchart in Fig. 5b. The system is converged initially. Then each design variable and input quantity is perturbed sepa-

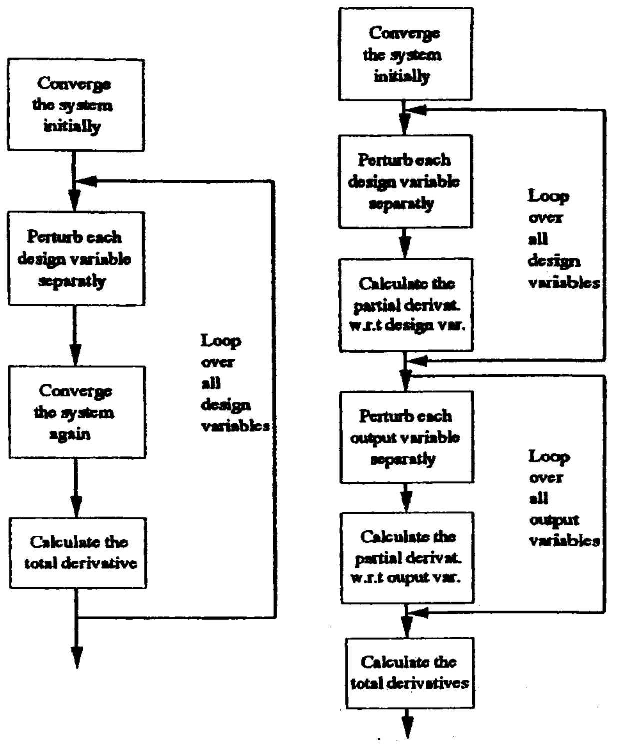

Converge the system initially
Perturb each design variable separatly
Converge the system again
Calculate the total derivative
Loop over all design variables
Converge the system initially
Perturb each design variable separatly
Calculate the partial derivat. w.r.t design var.
Loop over all design variables
Perturb each output variable separatly
Calculate the partial derivat. w.r.t ouput var.
Loop over all output variables
Calculate the total derivatives

Fig. 5. (a) and (b) Finite difference and GSE flowcharts

Table 1. Results for simulation 1

# SS	Exp	Sleep times [sec]	No. of calc.	$\frac{\#\text{calc}}{\#\text{clients}}$	total time	mean sleep time [sec]	$\frac{\text{total time}}{\text{mean sleep time}}$
3	1	3 × 10	137	46	496	10	4.96
3	2	3 × 50	137	46	2335	50	46.7
3	3 × 250	134	45	11282	250	45.1	
3	4	1 × 30 1 × 20 1 × 10	141	47	978	20	48.9
3	5	1 × 30 1 × 20 1 × 10	126	42	1012	23.33	43.4
3	61 × 90 1 × 30 1 × 10	150	50	2207	43.33	50.9	
3	7	1 × 250 1 × 50 1 × 10	150	50	5231	103.33	50.6
3	8	1 × 1000 1 × 100 1 × 10	157	51	18912	370	51.1
6	1	6 × 10	382	64	867	10	86.7
6	2	6 × 50	361	60	3244	50	64.9
6	3	6 × 250	340	57	14827	250	59.3
6	4	2 × 30 2 × 20 2 × 10	401	61	1574	20	78.7
6	5	2 × 40 2 × 20 2 × 10	383	64	1721	23.33	73.8
6	62 × 90 2 × 30 2 × 10	330	55	2597	43.33	59.9	
6	7	2 × 250 2 × 50 2 × 10	375	62	6655	103.33	64.4
6	8	2 × 1000 2 × 100 2 × 10	357	59	22193	370	60.0
9	1	9 × 10	1217	135	2630	10	263
9	2	9 × 50	1083	120	7181	50	143.6
9	3	9 × 250	996	111	28828	250	114.3
9	4	3 × 30 3 × 20 3 × 10	1298	144	4104	20	205.2
9	5	3 × 40 3 × 20 3 × 10	1333	148	4705	23.33	201.7
9	6	3 × 90 3 × 30 3 × 10	1328	147	7641	43.33	176.3
9	7	33 × 250 3 × 50 3 × 10	1905	212	23905	103.33	231.3
9	8	3 × 1000 3 × 100 3 × 10	1950	217	82133	370	222.0

rately for each subsystem and the partial derivatives for the subsystem output equations are calculated. After all the partial derivatives are calculated, the total derivatives can be computed by solving the equation for the desired matrix. The important difference between the finite difference and GSE approaches is that the system does not have to be reconverged again after each loop iteration in GSE since only the partial derivatives of the subsystem equations with respect to the design variables are calculated.

4 Server/client userinterfaces

The flowcharts in Figs. 5a and b are implemented in the WICkEDProblemManager. In the case of a GSE-solution, the program calculates the GSM-matrix and the partial-derivative matrix and then uses an LU-Decomposition (Press *et al.* 1988) to solve for the total-derivative matrix. The results, including the converged system and the total-derivatives, are saved to a file after the program has finished.

The equations of the subproblems are coded in the WICkEDSubProblem. The WICkEDProblemManager also initializes the starting values for the design variables and the system output equations.

The actual server and client are implemented using the ServerUserInterface and the ClientUserInterface. A picture of the Server and the ClientUserInterface (UI) can be seen in Figs. 6a and b.

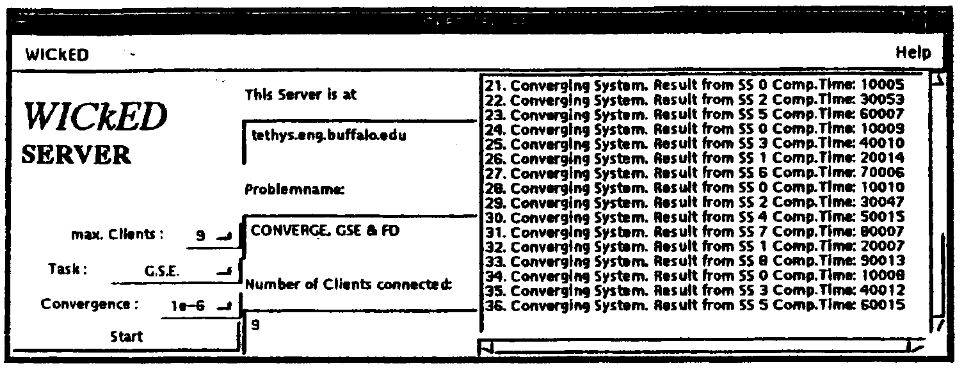
The Server UI includes a number of buttons on the left-hand side. The first button sets the maximum number of clients that can connect to the server. The task button choices include converging the system, converging the system and execute a finite difference calculation, or converging and then do a GSE computation of the sensitivities. The next button sets the convergence criteria for the system convergence. The last button starts the server after all settings are made. The displays show the name of the computer where the server is located, the name of the problem and the number of clients that are currently connected to the server. The right hand side display informs the user about incoming results and ongoing calculations. The Client UI has buttons for selecting the client number, the computer where the server is located and buttons for connecting and disconnecting to server. Displays include the server location, the client location, and the current state of the client as well as a display for information on current calculations.

5 Simulation of complex systems

5.1 *Introduction*

The motivation for doing parametric studies on MDO systems in a distributed computing environment is to gain knowledge about the behaviour of these systems in such an environment and to provide heuristics for future realistic MDO applications. When the solutions to the subsystems are computed in parallel rather than sequentially, the convergence behaviour changes. The convergence process is even more complicated when the computation times for the individual subsystems are not equal but different for each subsystem. Then, the order in which subsystem results are received is not the same as the order in which the inputs are sent out by the server.

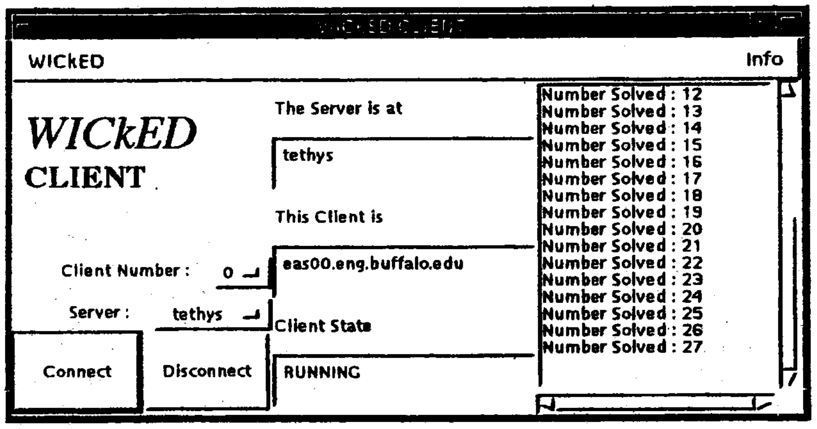
Three major sets of experiments and some preliminary studies were performed in this work. In the first case, any client is able to solve any subsystem. Hence, as soon as a



This Server is at: tethys.eng.buffalo.edu
Problemname: CONVERGE. GSE & FD
max. Clients: 9
Task: G.S.E.
Convergence: 1e-6
Number of Clients connected: 9

```
21. Converging System. Result from SS 0 Comp.Time: 10005
22. Converging System. Result from SS 2 Comp.Time: 30053
23. Converging System. Result from SS 5 Comp.Time: 60007
24. Converging System. Result from SS 0 Comp.Time: 10009
25. Converging System. Result from SS 3 Comp.Time: 40010
26. Converging System. Result from SS 1 Comp.Time: 20014
27. Converging System. Result from SS 6 Comp.Time: 70006
28. Converging System. Result from SS 0 Comp.Time: 10010
29. Converging System. Result from SS 2 Comp.Time: 30047
30. Converging System. Result from SS 4 Comp.Time: 50015
31. Converging System. Result from SS 7 Comp.Time: 80007
32. Converging System. Result from SS 1 Comp.Time: 20007
33. Converging System. Result from SS 8 Comp.Time: 90013
34. Converging System. Result from SS 0 Comp.Time: 10008
35. Converging System. Result from SS 3 Comp.Time: 40012
36. Converging System. Result from SS 5 Comp.Time: 60015
```

Fig. 6a. WICkED ServerUserInterfaces



The Server is at: tethys
This Client is: eas00.eng.buffalo.edu
Client Number: 0
Server: tethys
Client State: RUNNING

```
Number Solved : 12
Number Solved : 13
Number Solved : 14
Number Solved : 15
Number Solved : 16
Number Solved : 17
Number Solved : 18
Number Solved : 19
Number Solved : 20
Number Solved : 21
Number Solved : 22
Number Solved : 23
Number Solved : 24
Number Solved : 25
Number Solved : 26
Number Solved : 27
```

Fig. 6b. WICkED Client-UserInterfaces

client has solved a problem and the result is received by the server, the server sends out the next problem which should be solved by this client. In this case, the order on which subproblems are worked is determined by the server. These experiments are simpler but not as realistic as the next case.

For the second case, each client can work only on a specific problem. This corresponds to a real situation where each group has their own computer system with only their code(s). As soon as a client has solved a problem, the server will send out the next problem that can be solved by this client. The order in this case is determined by the clients, not by the server, as in the first case.

The last case investigated looks at the situation where more clients are available to those subsystem analyses which are more computationally expensive than others. The following section describes how the experiments are made as realistic as possible.

5.2 *Creating realistic problems*

The systems used for these parametric studies are complex systems created by the MDO simulation software CASCADE (Hulme and Bloebaum 1996). These systems are relatively simple compared to real systems since they consist only of a summation of several exponential functions with different coefficients. The computing time for such equations is actually very short - in the millisecond range on a Sun Sparcstation 4. In contrast, the time it takes to transport the data (which is the necessary input for these equations) over the network from the server to the client computer is much longer. Depending on the size of the system and the speed of the network connection, this time may be a few hundred milliseconds and is therefore considerably longer than the actual computing time.

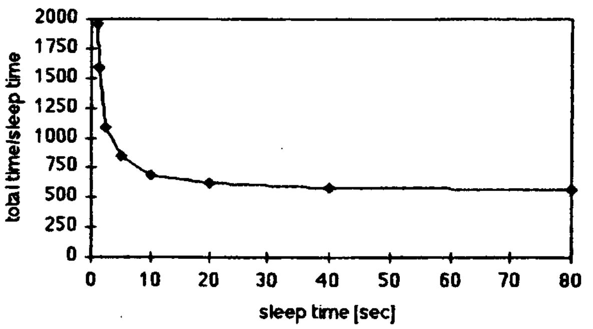


Fig. 7. Finding the minimum sleep time

The computing time for a real world optimization problem is considerably longer, probably ranging from several seconds to minutes and often days or even weeks. In such a case the computing time is much longer than the data transportation time and the data transportation time may therefore be neglected.

In order to simulate the behaviour of such a realistic system, an artificial sleep time is incorporated in the subsystems, which extends the actual computing time. This also makes it

Table 2. Results for simulation 2

# SS	Exp	Sleep times times [sec]	No. of calc.	# calc. per client time	Total time time [sec]
3	1	3×10	132	3×44	464
3	2	3×50	135	3×45	2330
3	3	3×250	141	3×47	11796
3	4	1×30 1×20 1×10	148	1×31 1×42 1×75	957
3	5	1×40 1×20 1×10	223	1×37 1×65 1×121	1505
3	6	1×90 1×30 1×10	329	1×32 1×80 1×217	2906
3	7	1×250 1×26 1×10	578	1×26 1×100 1×452	6519
3	8	1×1000 1×100 1×10	2200	1×28 1×215 1×1927	28024
6	1	6×10	360	6×60	830
6	2	6×50	360	6×60	3322
6	3	6×250	354	6×59	14959
6	4	2×30 2×20 2×10	380	2×43 2×57 2×90	1461
6	5	2×40 2×20 2×10	480	2×46 2×74 2×120	2021
6	6	2×90 2×30 2×10	578	2×35 2×78 2×175	3384
6	7	2×250 2×50 2×10	1192	2×35 2×121 2×440	8878
6	8	2×1000 2×100 2×10	3926	2×35 2×334 2×1695	35133
9	1	9×10	1151	9×128	2437
9	2	9×50	1206	9×135	7943
9	3	9×250	1134	9×126	38992
9	4	3×30 3×20 3×10	1429	3×112 3×146 3×218	4292
9	5	3×40 3×20 3×10	1629	3×108 3×173 3×262	5251
9	6	3×90 3×30 3×10	2602	3×108 3×253 3×507	11017
9	7	33×250 3×50 3×10	6078	3×116 3×466 3×1444	30263
9	8	3×1000 3×100 3×10			

possible to investigate a number of combinations of different subsystems by simulating different computing times for the coupled subsystems within a system.

First, it is essential to find the minimal computing time necessary for the simulation of a real system. In this case, a real system is defined as a system where the minimum time it takes to compute the output of the system is sufficiently larger than the longest time it takes the input and output data to travel over the network. The longest network traveling time occurs when the amount of data sent over the network is the largest. This is the case for the biggest system simulated, which is a system consisting of ten subsystems with ten output equations and five design variables each. Several different sleep times for this system have been investigated, with the sleep times being equal for all subsystems within one system. Those systems were run with one client only. The result is shown in Fig. 7, where the sleep time is plotted on the X-axis and the total time it took to solve the system divided by the individual sleep time is shown on the Y-axis. For a system where the computing time is sufficiently larger than the network traveling time, the total time it takes to solve the system is linearly dependent on the sleep time. In this case, the coefficient of total time over sleep time is essentially constant. It can be seen that this occurs somewhere between approximately 10 to 20 seconds.

5.3 Simulation 1

From Fig. 7, it is seen that the coefficient approaches a constant value as the sleep time increases. From this graph, it is assumed that a minimum computing time of 10 seconds is sufficient to simulate a real system as described above and, hence, this time is used in all following calculations.

Table 4. Selected sensitivity results for the structural design variable of the CSI problem

Sensitivity	$A = 1$ in^2	$A = 0.1$ in^2
$\frac{dY_{s1}}{dX_s}$	61.4558	61.4558
$\frac{dY_{s2}}{dX_s}$	61.1866	61.4558
$\frac{dY_{s3}}{dX_s}$	61.4558	61.4558
$\frac{dY_{s4}}{dX_s}$	60.3661	55.4331
$\frac{dY_{s5}}{dX_s}$	104.912	104.912
$\frac{dY_{s6}}{dX_s}$	104.698	103.688
$\frac{dY_{s7}}{dX_s}$	61.4558	61.4558
$\frac{dY_{s8}}{dX_s}$	60.2712	54.7283
$\frac{dY_{c1}}{dX_s}$	-0.112987	-1.4686
$\frac{dY_{c2}}{dX_s}$	-0.455284	-6.03167
$\frac{dY_{c3}}{dX_s}$	-0.089751	-1.2254
$\frac{dY_{c4}}{dX_s}$	-0.49505	-6.73744

In this set of experiments, any client is able to perform any subsystem calculation. As already noted, while this case is not very realistic, it serves to provide a foundation for comparison of other experiments. These simulations include three different systems. The first system consists of three subsystems with three input and three output variables, respectively. The second system has 6 subsystems with five input and six output variables and the third system has 5 input and nine output variables. In these simulations, the systems are converged and then the total sensitivities for the systems are computed using the GSE Method (Sobieszczanski-Sobieski 1990).

Table 3. Results for simulation 3

#	Exp	Sleep times	No. of calc.	# calc. per client	Total time
SS		times [sec]		time	time [sec]
6	9	2 × 90 2 × 30 2 × 10	380	2 × 43 2 × 47 2 × 90	1461
6	10	4 × 90 2 × 30 2 × 10	472	4 × 23 2 × 61 2 × 129	2274
6	11	6 × 90 2 × 30 2 × 10	582	6 × 25 2 × 69 2 × 146	2485
6	12	8 × 90 2 × 30 2 × 10	604	8 × 24 2 × 68 2 × 138	2443
6	13	10 × 90 2 × 30 2 × 10	648	10 × 24 2 × 687 2 × 136	2513
6	9	2 × 250 2 × 50 2 × 10	1192	2 × 35 2 × 121 2 × 440	8878
6	10	4 × 250 2 × 50 2 × 10	1054	4 × 26 2 × 106 2 × 369	6869
6	11	6 × 250 2 × 50 2 × 10	778	6 × 19 2 × 78 2 × 254	4857
6	12	8 × 250 2 × 50 2 × 10	872	8 × 20 2 × 85 2 × 271	5534
6	13	10 × 250 2 × 50 2 × 10	822	10 × 18 2 × 80 2 × 240	4900
6	9	2 × 1000 2 × 100 2 × 10	3926	2 × 35 2 × 234 2 × 1695	35133
6	10	4 × 1000 2 × 100 2 × 10	1724	4 × 16 2 × 108 2 × 722	17078
6	11	6 × 1000 2 × 100 2 × 10	1666	6 × 14 2 × 107 2 × 684	15079
6	12	8 × 1000 2 × 100 2 × 10	1332	8 × 12 2 × 89 2 × 529	12086
6	13	10 × 1000 2 × 100 2 × 10	1724	10 × 14 2 × 116 2 × 675	15109

The distribution of the sleep times for the different experiments and the results of all simulations are shown in Tables 1 to 4. Table 1 shows the setups and results for all simulations performed in this section. The upper part of the table includes all data for the 3 subsystem problem. The third column contains the distribution of the sleep times for the particular experiment. For example, for 3 subsystems and experiment 4, there is one subsystems with a sleep time of 10 seconds, one subsystems with 20 seconds, and one subsystems with 30 seconds, where the sleep time corresponds to the calculation time for the solution of one subproblem. For these experiments, the number of clients connected to the system is always equal to the number of subsystems. The total number of calculations for all subsystems is shown in the second column, followed by the number of calculations divided by the number of clients (which is the number of calculations that each client performed). The last three columns show the total time required to solve the whole system, the mean of all sleep times of the system, and the total time divided by the mean sleep time.

These results are also visualized in the following figures. Fig. 8 shows the number of calculations per client for the systems consisting of 3, 6 and 9 subsystems.

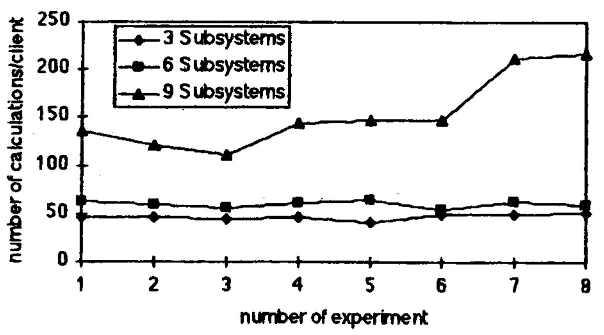


Fig. 8. Number of calculations per client for simulation 1

This figure visualizes that the number of calculations per client is about the same for all experiments for one subsytem. Therefore, it can be assumed the number of calculations is independent of the calculation times. Figure 9 proves, as expected, that the total time necessary to solve the whole system increases when the system size increases.

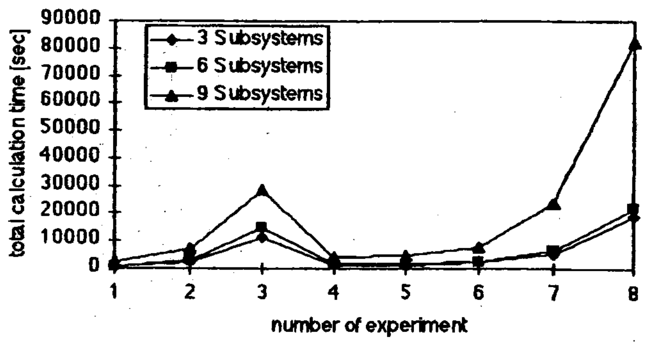


Fig. 9. Total calculation time for simulation 1

Figure 10 shows that the total calculation time divided by the mean of all sleep times of the system always varies, for the most part, around a constant value.

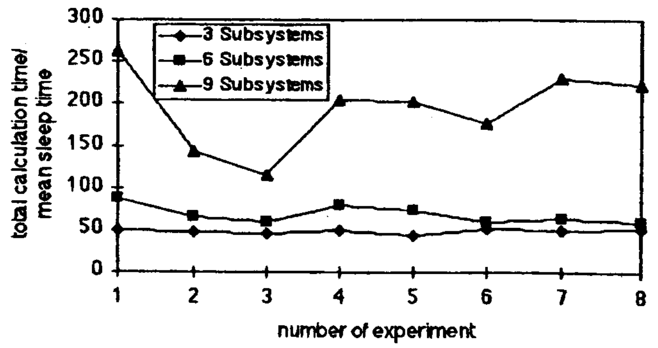


Fig. 10. Total calculation time over mean sleep time for simulation 1

This constant value is specific for each problem. This can also be seen by looking at the following equation:

$$\text{time}_{\text{total}} = \frac{\#\text{clients}}{\#\text{subsystems}} \times \text{mean sleep time} \times \frac{\#\text{calc}}{\#\text{clients}} + \text{time}_{\text{server}} + \text{time}_{\text{network}} \approx$$

$$\frac{\#\text{clients}}{\#\text{subsystems}} \times \text{mean sleep time} \times \frac{\#\text{calc}}{\#\text{clients}} . \qquad (3)$$

This equation describes how the total calculation time can be computed. The number of clients connected to the server and the number of subsystems in the system are know. The sleep times are equal to the calculation times for a single subsystem solution and therefore the mean sleep time is known after a few calculations. In this case the sleep times are given. The number of calculations per client is approximately constant and independent of the sleep times. The computation time of the server and the network travel time are in general much smaller than the calculation times for the subsystem solutions and may be neglected without a significant loss of accuracy.

It should be noted that there is some randomness in the way the results are obtained because the convergence history for the systems may be different for every solution. The inputs to the subsystems are always sent out by the server in the same order. For example, for the system with three subsystems the order is always 1 - 2 - 3 and will repeat as necessary until the system is converged. In contrast, the order in which the results are received is not necessarily constant. For example, in one iteration, the result of a certain subsystem may arrive at the server before the result of another subsystem, and this order may be reversed the next iteration. This may be due to a number of reasons. For example, the network traveling time is not always constant but also dependent on the load on the network. This also means that the subsystem inputs are updated in a different order, which therefore changes the subsystem results the next time and so on. The final results for a certain system are always the same, but the way these results are obtained may be different every time because they are obtained by virtual parallel computing (distributed computing).

5.4 Simulation 2

For this second set of simulations, each client can only work on a specific subproblem. This is a more realistic setup compared to the previous simulation case. In a company where each discipline has their codes only on their computers, then only a particular computer can be used to solve particular subproblems. The server can assign subproblems only to those clients which have the code to solve them. This setup requires a much greater effort in terms of coordination on the server side because subproblem inputs are not sent out in a predetermined order. Immediately after a client has finished a problem and the problem results have been returned to the server, the server calculates the next set of inputs for this client. It is also possible that more than one set of results arrives at the server at almost the same time. Therefore, the part of the server code which performs the incorporation of incoming results into the solution of the global problem must be synchronized. This means that this part of the code cannot be executed simultaneously. Since Java is a multi-threaded language, it usually allows multiple threads to run which could modify objects simultaneously. This is prevented by synchronizing this part of the code, which is the QuickSubProblemConstructor. The numerical results can be seen in Table 2 in the Appendix.

For example in Table 2, 3 Subsystems and Experiment No. 2 there is a sleep time of 50 sec for all three subsystems. The total number of calculations it takes to solve the system and to calculate the derivatives is 135 and each subsystem calculation is performed 45 times. The total computing time for this system is 2330 sec.

Figure 11 compares the results of these simulations to the previous experiments. As expected, the comparison confirms that there is no change in the results for those problems where the calculation time is equal for all subproblems. This is the expected result because in both cases the number of clients is equal to the number of subproblems and all subproblems have the same sleep time.

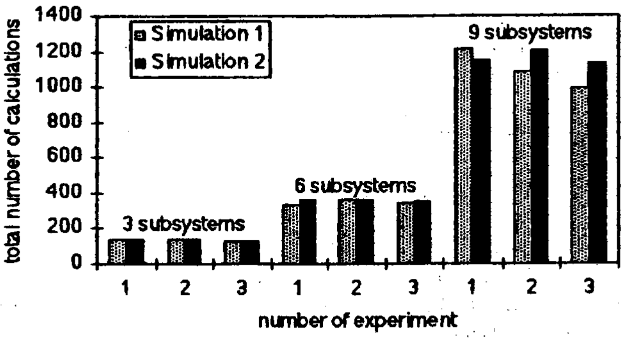


Fig. 11. Comparison of simulation 2 to 1

Figures 12 to 14 display the results for experiments 4 to 8, where the sleep times are different for all subproblems. In these figures, a client set refers to all those clients that work on problems with an equal sleep time.

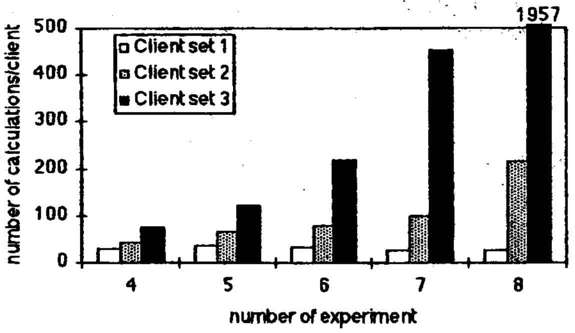


Fig. 12. Results for 3 subsystems - simulation 2

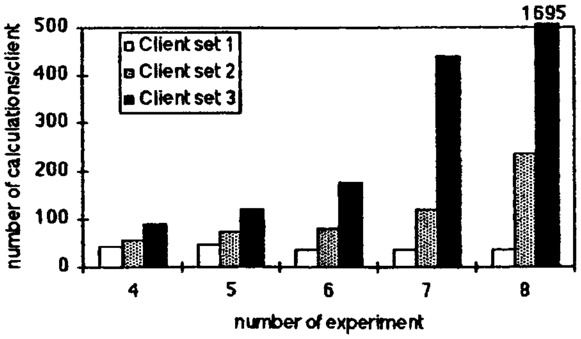


Fig. 13. Results for 6 subsystems - simulation 2

The most important result of these figures (Figs. 12-14) is that the number of calculations for the clients with the

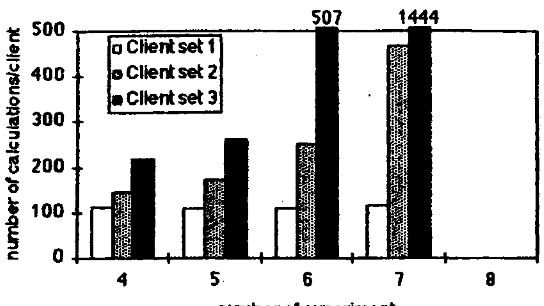


Fig. 14. Results for 9 subsystems - simulation 2

longest calculation time is constant for a specific problem. The subproblems which have the longest calculation time are the determining factor for the convergence process. Each subproblems has to be solved a certain number of times (which is specific for a particular problem).

For example, in Fig. 14, the Client Set 1 always requires approximately 110 calculations. This is specific for this problem. Calculations of the subsystems with the shorter sleep times are executed constantly. Due to this, the number of calculations is extremely high for these subsystems. The system with 6 subsystems for example, executed the subsystem with the longest calculation time only around 40 times, while the number of calculations for the fastest subsystem in experiment 8 was almost 1700.

The longest computing time determines the total time as can be seen in the following equation:

$$\text{time}_{\text{total}} = \#\text{calc}_{\min} \times \text{time}_{\text{longest comp}} + \text{time}_{\text{server}} + \text{time}_{\text{network}} \approx \#\text{calc}_{\min} \times \text{time}_{\text{longest comp}}. \quad (4)$$

This result suggests that if possible, the most computing power should be given to those subproblems which have the longest computing time. This leads to the next case of simulations, where more clients work on subsystems with the longest calculation times than on the other subsystems.

5.5 Simulation 3

This final set of experiments uses the system with six subsystems to investigate the use of a larger number of clients for those subsystems which require the most computational effort. Three different cases with a sleep time distribution of (90,30,10) seconds, (250,50,10) seconds and (1000,100,10) seconds, respectively, are examined.

Figures 15 and 16 display the total number of calculations and the total calculation time. For the system with the largest variation in the sleep time distribution (1000,100,10), there is a significant decrease in the number of calculations as well as in the calculation time when two clients instead of one client are used for each 1000 second subsystem. There is also a gain for the system with the (250,50,10) second distribution, but for the system where the calculation times are closest together (90,30,10), the system cannot be solved any faster than before. It can be seen from these figures that there is a point for each system where an increase in the number of clients does not result in a faster solution of the task. This means that it does not always make sense to contribute as much computing power as possible to the client with the longest sleep time. The reason for this can be seen in Fig. 17.

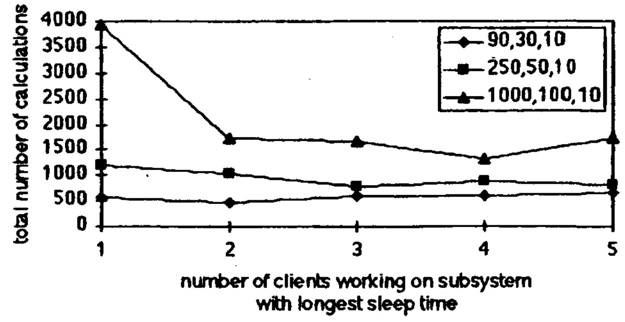


Fig. 15. Total number of calculations - simulation 3

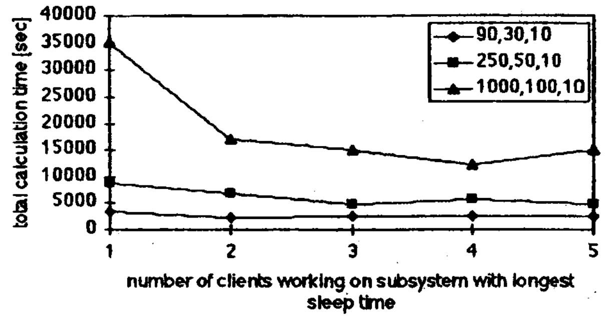


Fig. 16. Total calculation time - simulation 3

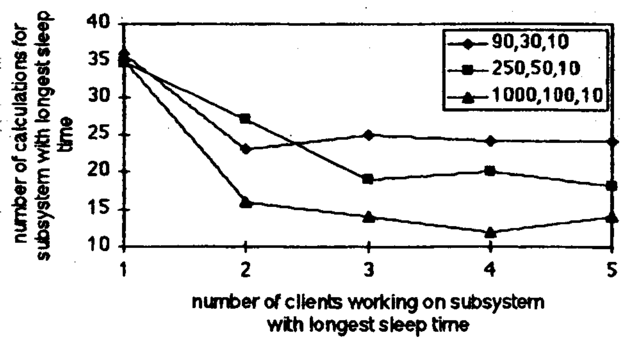


Fig. 17. Number of calculations for subsystem with longest sleep time

An increase in the number of clients that work on the same problem means that situations may occur where several clients work on the same problem at the same time. This is not desired because the results are updated when they are received at the client, which means that the latest result received at the client is the only one that counts. Therefore, when two clients work on the same problem at exactly the same time, only one of those two results can actually be used. The situation may occur frequently but is especially undesirable when the client works on the largest problem.

Figure 17 shows that there is a minimum number of calculations necessary to converge and solve a subsystem and this number cannot be reduced even by contributing more computing power to this subproblem. It can also be concluded that it makes sense for systems of this size to double the computing power for the subsystems with the longest computing time. This is a general problem when doing parallel computing for distributed systems and should be investigated further. In order to save computing power, subsystem calculations should only be executed when the there is actually new input for the system, which means there will be a change in the output. This is especially critical for system with large differences in the computation times. There, the subsystems with short computation times are executed unnecessarily for a large number of times.

6 Application problem: structural and control optimization of a truss structure

6.1 Introduction

In this section, a more realistic multidisciplinary problem is used to demonstrate the potential of distributed computing using Java. Large structures in space are suspect to environmental and onboard disturbances that can produce structural vibrations. These disturbances may be caused by the crew, the docking of other spacecraft, or other influences. Space structures tend to have little inherent damping. The low mass to surface ratio results in extremely high flexibility, thereby requiring active controls. The integration of disciplines in the optimization approach in order to achieve an optimal design is desirable especially in aerospace design. The interactions between the involved disciplines are represented more realistically. In this application, a ten-bar truss structure is used as an example.

6.2 Optimization model

The ten-bar truss structure seen in Fig. 18 is the model used to demonstrate the effectiveness of using Java to perform the GSE method in the controls/structures interaction (CSI) problem.

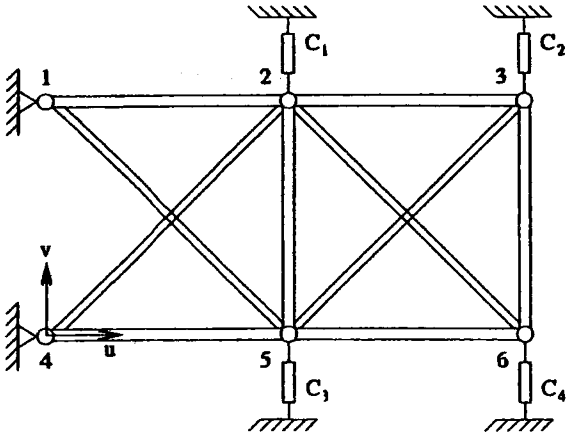


Fig. 18. Truss structure with four controllers

The truss is equipped with active controllers at the free nodes to limit the dynamic displacements to preassigned levels. Two degrees of freedom (u and v) are permitted at each of the six nodes, thus yielding a twelve degree-of-freedom system for which four of these are constrained at the wall. The truss is made of aluminum and is subject to static and dynamic loadings as seen in Fig. 19. The structural lateral dynamic displacements are controlled by hydraulic controllers placed at nodes 2, 3, 5 and 6. The forcing function $F_d(t)$ is a ramp input that acts in the y-direction at node 3. There is also a static force F_S acting at node 6.

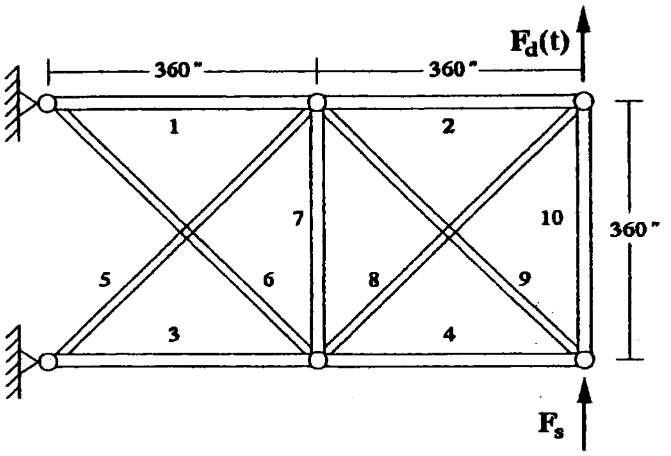


Fig. 19. Truss structure with static and dynamic loading

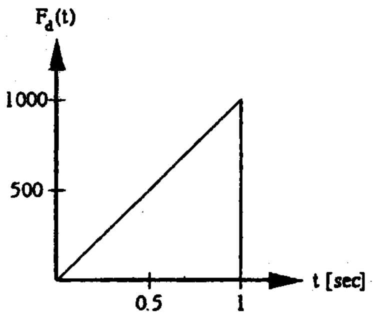


Fig. 20. Dynamic force function

6.3 Design problem statement

The design objective of the CSI problem is to find the minimum weight structure subject to constraints. The constraints may include static stresses, natural frequencies, static and dynamic displacements. Here, only the dynamic displacement constraints are included in this problem. The objective function of the design synthesis problem is the total weight contributed from the structures and the control system. This function can be written as

$$F = W_S + W_C \, . \tag{5}$$

Traditionally, the weight of the control system (i.e. transducers, sensors, actuators, hydraulics, etc.) is either considered negligible with respect to the structural weight or is arbitrarily assigned as some constant. However, when extremely light, flexible, actively-controlled structures are considered as in the case of space structures or ultra-light aircraft, the weight of the control system can become significant. In this work, the weight of the control system is considered to be a function of the control input u. An empirical relation is used to represent this weight relationship and is as follows:

$$Wc = \sum_{i=1}^{m} Kc\sqrt{|u_i|} \, , \tag{6}$$

where m is the total number of actuators and K_C is a prescribed constant, which is 0.5 in this study.

The total weight of all structural components is calculated as

$$Ws = \sum_{j=1}^{n} \rho_i A_i L_i \,, \tag{7}$$

where n is the total number of truss members, ρ is the density of the material used, A is the cross-sectional area and L the length of each truss member.

The design variables are divided into structural design variables and control design variables. The structural design variables X_S are the cross-sectional areas of the truss elements

$$X_S = (A_1, A_2, \ldots, A_{10})^T \,. \tag{8}$$

The control design variables X_C are the constants α and β which determine the damping matrix of the structure

$$X_C = (\alpha, \beta)^T \,. \tag{9}$$

The control design variables influence the controls analysis through the **A** matrix in the first-order state-space equation, thus contributing to the design of the control law.

The constraints for this optimization problem can be divided into structural and control system constraints. Here, the structural constraints are taken as the static displacements and are in the form of

$$g_i = \frac{G_i}{G_i\mathrm{all}} - 1 \le 0 \,, \tag{10}$$

where $G_i$all is the prescribed allowable limit for each constraint. Similarly, the control system constraints dictate limitations on the dynamic displacements. There can also be upper and lower bounds specified for the design variables.

6.4 *Determination of the sensitivity information*

The behaviour sensitivity derivatives are required for the piece-wise linear approximation to the nonlinear optimization process. The sensitivities are found by using the GSE method. The interactions between the structural and controls systems can be seen in Fig. 21. Each system has inputs in form of design variables and outputs from the other system. The analyses are written as

$$S[(X_S, Y_C)Y_S] = 0 \,, \quad S[(X_C, Y_S)Y_C] = 0 \,. \tag{11}$$

The outputs can be rewritten in the explicit form.

$$Y_S = f_S(X_S, Y_C) \,, \quad Y_C = f_C(X_C, Y_S) \,, \tag{12}$$

and the following global sensitivity equations are obtained:

$$\begin{bmatrix} \mathbf{I} & -\dfrac{\partial \mathbf{Y}_S}{\partial \mathbf{Y}_C} \\ -\dfrac{\partial \mathbf{Y}_C}{\partial \mathbf{Y}_S} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \dfrac{\mathrm{d}\mathbf{Y}_S}{\mathrm{d}\mathbf{X}_S} & \dfrac{\mathrm{d}\mathbf{Y}_S}{\mathrm{d}\mathbf{X}_C} \\ \dfrac{\mathrm{d}\mathbf{Y}_C}{\mathrm{d}\mathbf{X}_S} & \dfrac{\mathrm{d}\mathbf{Y}_C}{\mathrm{d}\mathbf{X}_C} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \mathbf{Y}_S}{\partial \mathbf{X}_S} & 0 \\ 0 & \dfrac{\partial \mathbf{Y}_C}{\partial \mathbf{X}_C} \end{bmatrix} . \tag{13}$$

6.5 *Subsystem analysis*

6.5.1 *Structures subsystem*

The equation for the free vibration eigenvalue problem associated with the structures subsystem is

$$(\mathrm{K} - \omega_i^2 \mathrm{M})\phi_i = 0 \,, \tag{14}$$

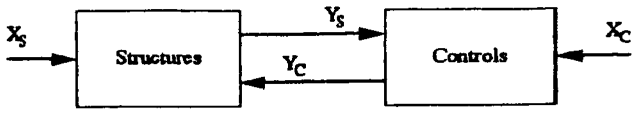


Fig. 21. Structural/control interactions

where ϕ and ω are the eigenvector and the eigenvalue for the i-th mode, respectively; **M** and **K** are the mass and stiffness matrices for the structure, which are computed in the structural analysis. The static structural analysis can be written as

$$\mathbf{Kd} = \mathbf{f} \,, \tag{15}$$

where **d** is the displacement vector and **f** is the applied load vector.

6.5.2 *Controls subsystem*

The equation of motion for a structure with active controls and undergoing forced vibration is

$$\mathbf{M}\ddot{\mathbf{x}} + \mathbf{C}\dot{\mathbf{x}} + \mathbf{Kx} = \mathbf{Du} \,, \tag{16}$$

where **M** and **K** are again mass and stiffness matrices, **C** is the damping matrix and **D** is the control influence matrix. The damping matrix is traditionally represented by the proportional relationship

$$\mathbf{C} = \alpha \mathbf{M} + \beta \mathbf{K} \,, \tag{17}$$

where α and β are proportional constants which are the design variables of the controls subsystems.

The second-order differential equation is a finite element representation of a structure in which the mass and stiffness matrices are symmetric and positive definite or at least positive semi-definite. This equation can also be written as a first-order state space equation

$$\dot{\mathbf{z}} = \mathbf{Az} + \mathbf{Bu} \,, \tag{18}$$

where the state vector is defined as

$$\mathbf{z} = \begin{pmatrix} \mathbf{x} \\ \dot{\mathbf{x}} \end{pmatrix} , \tag{19}$$

Further, the plant matrix is in the form

$$\mathbf{A} = \begin{bmatrix} 0 & \mathbf{I} \\ -\mathbf{M}^{-1}\mathbf{K} & -\mathbf{M}^{-1}\mathbf{C} \end{bmatrix} , \tag{20}$$

where **I** is the identity matrix, and the control vector is

$$\mathbf{B} = \begin{pmatrix} 0 \\ -\mathbf{M}^{-1}\mathbf{D} \end{pmatrix} . \tag{21}$$

The optimal control feedback control law (Junkins and Kim 1993) is obtained by minimizing a quadratic performance index J, which is a function of the state vector and the control vector. The optimal control problem is therefore

$$J = \frac{1}{2} \int_0^{\infty} [\mathbf{z}^T \mathbf{Q} \mathbf{z} + \mathbf{u}^T \mathbf{R} \mathbf{u}] \, \mathrm{d}t \,, \tag{22}$$

subject to

$$\dot{\mathbf{z}} = \mathbf{AZ} + \mathbf{Bu} \,, \tag{23}$$

where the matrices **Q** and **R** are arbitrary weighting matrices with the restriction that **Q** must be at least positive semi-definite and **R** must be positive definite. The solution of this optimal control problem results in the nonlinear algebraic equation called the algebraic Ricatti equation (ARE) which has the following form:

$$\mathbf{A}^T\mathbf{P} - \mathbf{PBR}^{-1}\mathbf{B}^T\mathbf{P} + \mathbf{PA} + \mathbf{Q} = 0 \,, \tag{24}$$

where $\mathbf{P}$ is a symmetric positive definite matrix referred to as the Ricatti matrix.

Once the Ricatti matrix is found from the ARE, the control gain matrix $\mathbf{G}$ can be determined as

$$\mathbf{G} = \mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}. \tag{25}$$

The optimal state feedback control law then results in

$$\mathbf{u} = -\mathbf{G}\mathbf{x}. \tag{26}$$

This relationship is substituted into the state equation to obtain

$$\dot{\mathbf{z}} = [\mathbf{A} - \mathbf{B}\mathbf{G}]\mathbf{z}. \tag{27}$$

This differential equation can be solved for the state vector which is then used to calculate the optimal control vector $\mathbf{u}$.

6.6 Numerical simulation

The optimization process for the coupled problem is shown in Fig. 21. In this study, only the convergence of the system and the sensitivity analysis is simulated using WICkED. Once the sensitivity derivatives of the system are known, the approximation and the optimization can be performed easily.

Table 5. Selected sensitivity results for the controls design variables of the CSI problem

Sensitivity	$A = 1$ in^2	$A = 0.1$ in^2
$\frac{dY_{s1}}{dX_{c1}}$	0	0
$\frac{dY_{s2}}{dX_{c1}}$	-0.250897	-0.612268
$\frac{dY_{s3}}{dX_{c1}}$	0	0
$\frac{dY_{s4}}{dX_{c1}}$	-0.592336	-1.20878
$\frac{dY_{s5}}{dX_{c1}}$	0	0
$\frac{dY_{s6}}{dX_{c1}}$	-0.196842	-0.322637
$\frac{dY_{s7}}{dX_{c1}}$	0	0
$\frac{dY_{s8}}{dX_{c1}}$	-0.636311	-1.28849
$\frac{dY_{s1}}{dX_{c1}}$	0	0
$\frac{dY_{s2}}{dX_{c1}}$	-0.033273	-0.012989
$\frac{dY_{s3}}{dX_{c1}}$	0	0
$\frac{dY_{s4}}{dX_{c1}}$	-0.135329	0.0308731
$\frac{dY_{s5}}{dX_{c1}}$	0	0
$\frac{dY_{s6}}{dX_{c1}}$	-0.026832	0.000061
$\frac{dY_{s7}}{dX_{c1}}$	0	0
$\frac{dY_{s8}}{dX_{c1}}$	-0.149817	-0.023838
$\frac{dY_{c1}}{dX_{c1}}$	-0.22672	-0.612329
$\frac{dY_{c2}}{dX_{c1}}$	-0.49417	-1.20905
$\frac{dY_{c3}}{dX_{c1}}$	-0.177599	-0.323686
$\frac{dY_{c4}}{dX_{c1}}$	-0.529616	-2.32838
$\frac{dY_{c1}}{dX_{c1}}$	0.000659	0.0129161
$\frac{dY_{c2}}{dX_{c1}}$	0.002367	0.0305255
$\frac{dY_{c3}}{dX_{c1}}$	0.000184	-0.000045
$\frac{dY_{c4}}{dX_{c1}}$	-0.000167	-0.024223

For this simulation, the given model is simplified further by assuming that all truss members have the same cross-sectional area, which is the structural design variable. The lumped mass matrix and the stiffness matrix for the truss structure are as follows:

$$\mathbf{M} = \rho\mathbf{A}\mathbf{L}\cdot \begin{bmatrix} 1+\frac{\sqrt{2}}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1+\frac{\sqrt{2}}{2}+m_{c1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1+\frac{\sqrt{2}}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1+\frac{\sqrt{2}}{2}+m_{c2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{3}{2}+\sqrt{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{3}{2}+\sqrt{2}+m_{c3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1+\frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1+\frac{\sqrt{2}}{2}+m_{c4} \end{bmatrix}, \tag{28}$$

$$\mathbf{K} = \frac{\mathbf{EA}}{\mathbf{L}} \begin{bmatrix} 3 & 1 & -1 & 0 & 0 & 0 & 1/2 & 1/2 \\ 1 & 2 & 0 & 0 & 0 & -1 & 1/2 & 1/2 \\ -1 & 0 & 3/2 & 1/2 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 3/2 & 1/2 & 1/2 & 0 & -1 \\ 0 & 0 & 1/2 & 1/2 & 3 & 1 & -1 & 0 \\ 0 & -1 & 1/2 & 1/2 & 1 & 2 & 0 & 0 \\ 0 & -1 & 1/2 & 1/2 & 1 & 2 & 0 & 0 \\ 1/2 & 1/2 & 0 & -1 & 0 & 0 & 1/2 & 3/2 \end{bmatrix}, \tag{29}$$

$$\mathbf{d} = (u_2\ v_2\ u_3\ v_3\ u_5\ v_5\ u_6\ v_6)^T. \tag{30}$$

The following material constants are used: $\rho = 0.1$ lbs/in^3 $= 2.768\ 10^4$ kg/m^3, $L = 360$ in $= 9.144$ m, $E = 10^7$ psi $= 7.0307\ 10^9$ kg/m^2. The starting value for the structural design variable is

$A = 1$ in^2 $= 6.4516\ 10^{-4}$ m^2.

Using these matrices, the equations for the two subsystems are implemented in WICkED. The design variables and the output equations for the structures system and the controls system are,

$$X_S = A, \tag{31}$$

$$Y_S = \mathbf{M}, \mathbf{K}, \tag{32}$$

$$X_C = \alpha, \beta, \tag{33}$$

$$Y_C = m_{cl}, m_{c2}, m_{c3}, m_{c4}, \tag{34}$$

where A is the cross-sectional area for all truss members, and $\mathbf{M}$ and $\mathbf{K}$ are the mass and stiffness matrices associated with the truss structure. These two structural output matrices are the necessary input for the plant matrix $\mathbf{A}$ in the controls problem. The controls design variables α and β are explained in (9) and the output variables are the masses of each of the control systems which are the input for the structures subsystem. As pointed out previously, the mass of a control system is a function of the control input u and in order to calculate the control input it is necessary to solve the first-order state space equation (27).

The solution to this equation is

$$x(t) = e^{\overline{A}t}x_0 + \int_0^t e^{\overline{A}t}\mathbf{B}\mathbf{U}(t)\,dt\,, \tag{35}$$

where $\overline{\mathbf{A}} = \mathbf{A} - \mathbf{BG}$ is the plant matrix of the closed loop system. This equation is implemented in the controls subsystem and is the computationally most expensive part of the analysis.

In this numerical analysis, the gain matrix **G** is considered to be constant. The gain matrix is calculated once with the initial values for all design variables and this matrix is used during the whole calculation. This does not eliminate the coupling of the subsystems because a coupling exists through the **A** matrix of the controls analysis.

WICkED was used to calculate the sensitivity derivatives for the truss structure. The results can then be used to perform an approximation and an optimization of the truss structure as shown in Fig. 22.

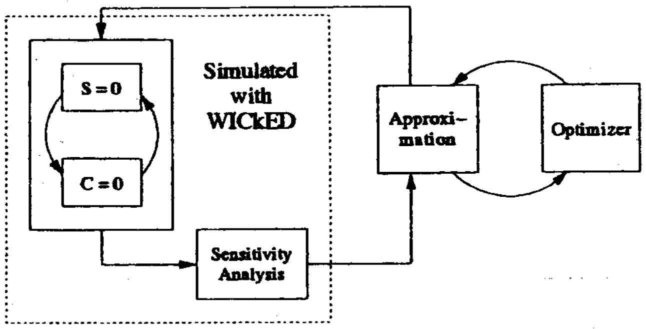


Fig. 22. Coupled structural/controls design process

The Tables 4 and 5 show selected sensitivity results for two different values of the cross-sectional area, which is the structural design variable. Tables 6 and 7 show a comparison of results obtained with the GSE method to those obtained with the finite difference method. In these tables, X_S is the cross-sectional area A and Y_{S1} to Y_{S8} are the elements on the diagonal of the lumped mass matrix; X_{C1} and X_{C2} are the constants α and β which are used to form the damping matrix **C** and Y_{C1} to Y_{C4} are the masses of the four controllers.

The results show that WICkED is capable of converging a realistic multidisciplinary design optimization problem. As expected, the network traveling time does not have any impact on the solution of a realistic problem. In this problem, the computing time for the controls subsystem was around 20 sec on a Sun Ultra Sparc. This proves that the assumptions in the Simulations Chapter are correct, considering that this is a relatively fast computer.

6.7 Simulation using the CSI problem

In this section, the Controls/Structures Interaction Problem is used for one set of simulations in order to substantiate the results reported in the previous chapter. These simulations correspond to Simulation #3 of the previous section, where a larger number of clients is used for the subsystem which requires the most computational effort.

In the CSI problem, the controls subsystem is much more computational expensive than the structures. Therefore this

Table 6. Comparison of GSE and finite difference results for the structural design variable

Sensitivity	GSE	F.D.
$\frac{dY_{s1}}{dX_s}$	61.455	61.455
$\frac{dY_{s2}}{dX_s}$	61.186	61.330
$\frac{dY_{s3}}{dX_s}$	61.455	61.455
$\frac{dY_{s4}}{dX_s}$	60.366	60.949
$\frac{dY_{s5}}{dX_s}$	104.91	104.91
$\frac{dY_{s6}}{dX_s}$	104.69	104.81
$\frac{dY_{s7}}{dX_s}$	61.455	61.455
$\frac{dY_{s8}}{dX_s}$	60.271	60.905
$\frac{dY_{c1}}{dX_s}$	-0.11298	-0.12517
$\frac{dY_{c2}}{dX_s}$	-0.45528	-0.50594
$\frac{dY_{c3}}{dX_s}$	-0.08975	-0.09989
$\frac{dY_{c4}}{dX_s}$	-0.49505	-0.55001

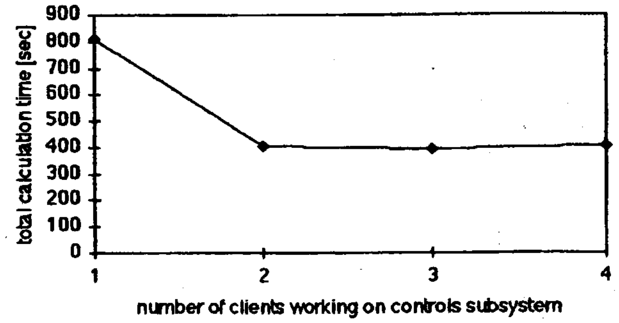


Fig. 23. Total calculation time - CSI problem

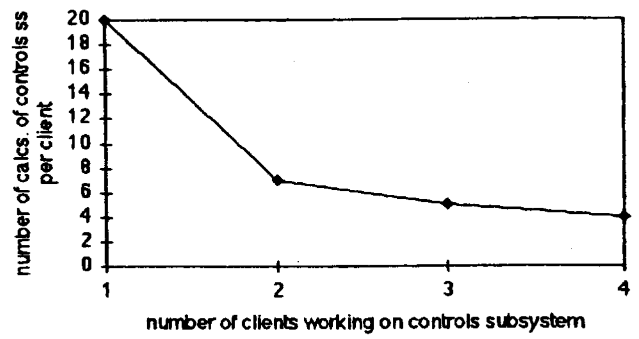


Fig. 24. Number of calculations for controls subsystem - CSI problem

subsystem is solved by more than one client. The results are reported in Figs. 23 and 24.

It was found that the results obtained by the simulation of the CSI problem correspond to those that were obtained previously in Simulation 3. As can be seen in Figure 23, the total computation time is reduced efficiently when the number of clients working on the controls problem is raised from one client to two clients. Raising the number of clients further does not improve the computing time. The reason for this is that there are too many clients doing the same work in parallel. Therefore some of the computing power is wasted.

Figure 24 shows that the number of computations of the

Table 7. Comparison of GSE and Finite Difference for the controls design variables

Sensitivity	GSE	F:D.
$\frac{dY_{s1}}{dX_{c1}}$	0	0
$\frac{dY_{s2}}{dX_{c1}}$	-0.25089	-0.33416
$\frac{dY_{s3}}{dX_{c1}}$	0	0
$\frac{dY_{s4}}{dX_{c1}}$	-0.59233	-0.94207
$\frac{dY_{s5}}{dX_{c1}}$	0	0
$\frac{dY_{s6}}{dX_{c1}}$	-0.19684	-0.26755
$\frac{dY_{s7}}{dX_{c1}}$	0	0
$\frac{dY_{s8}}{dX_{c1}}$	-0.63631	-1.01553
$\frac{dY_{s1}}{dX_{c1}}$	0	0
$\frac{dY_{s2}}{dX_{c1}}$	-0.03327	-0.10692
$\frac{dY_{s3}}{dX_{c1}}$	0	0
$\frac{dY_{s4}}{dX_{c1}}$	-0.13532	-0.44616
$\frac{dY_{s5}}{dX_{c1}}$	0	0
$\frac{dY_{s6}}{dX_{c1}}$	-0.02683	0.08987
$\frac{dY_{s7}}{dX_{c1}}$	0	0
$\frac{dY_{s8}}{dX_{c1}}$	-0.14981	-0.48675
$\frac{dY_{c1}}{dX_{c1}}$	-0.22672	-0.33416
$\frac{dY_{c2}}{dX_{c1}}$	-0.49417	-0.94207
$\frac{dY_{c3}}{dX_{c1}}$	-0.17759	-0.26755
$\frac{dY_{c4}}{dX_{c1}}$	-0.52961	-1.01533
$\frac{dY_{c1}}{dX_{c1}}$	0.00065	-0.10692
$\frac{dY_{c2}}{dX_{c1}}$	0.00236	-0.44616
$\frac{dY_{c3}}{dX_{c1}}$	0.00018	-0.08987
$\frac{dY_{c4}}{dX_{c1}}$	-0.00016	-0.48675

clients working on the controls problem only changes significantly when the number of clients for the controls subsystem is increased from one to two. Again, this corresponds to the previously obtained results.

In this chapter, it has been demonstrated using the CSI example, that Java has substantial potential for enabling virtual parallel computing for MDO applications.

7 Concluding remarks

This paper presented the Web Interface for Complex Engineering Design (WICkED) tool, which simulates the convergence of complex engineering systems in a distributed computing environment and also computes the sensitivities of those systems using the GSE or the finite difference method. Both of the above processes are well-suited for decomposition, and subsequent distribution to a parallel computing environment; hence their incorporation into this research work. As an extension of this work, it may be beneficial to attempt to extend the Java-based distributed computing methodologies to include a formal optimization procedure as well. Such an extension would finalize an "outer-loop" for a complete MDO design synthesis. However, it would likely be difficult to port established optimization codes (many of which are written in Fortran) that were designed to run in a sequential environment, to a Java-based parallel architecture.

A number of simulations solving different complex systems were performed and the convergence behaviour of those systems was analysed. The results obtained demonstrated that the subsystems with the longest computational times dominate the convergence process and that the most computational power should be assigned to those systems. It was exhibited that more than one computer can be used to work on a subsystem and that this is especially useful for the subsystem with the longest time. It was shown on the other hand, that too many computers working on the same problem at the same time produce results that cannot be used because they do not contain new information. The derivative calculation for a ten-bar truss problem was performed as an application problem in order to show the ability of WICkED to solve real problems.

It was further demonstrated that the programming language Java holds great promise for industrial MDO applications. Numerous areas for future work exist. Two of these include improving the speed of the calculations and the message passing over the network and the application of WICkED to more complex and more realistic problems. Also, subsystem calculations should only be performed when necessary in order to save computing power.

Acknowledgements

The authors wish to gratefully acknowledge partial support of this work under NSF PFF grant DMI 9553210 and NSF REG grant DMI 9622314.

References

Bloebaum, C.L. 1989: Global sensitivity analysis in control-augmented structural synthesis. *AIAA-Paper No. 89-0844*

Bloebaum, C.L. 1991: *Formal and heuristic system decomposition methods in multidisciplinary synthesis.* Ph.D. Thesis, University of Florida, Gainsville, FL

Bloebaum, C.L. 1995: Coupling strength-based system reduction for complex engineering design. *Struc. Optim.* **10**, 113121

Bloebaum, C.L.; Hajela, P.; Sobieszczanski-Sobieski, J. 1993: Decomposition methods for multidisciplinary synthesis. *Control & Dynamic Systems* **57**

Flanigan, D. 1996: *Java in a nutshell.* Sebastopol, CA: O'Reilly & Associates

Hulme, K.; Bloebaum, C.L. Development of CASCADE - A multidisciplinary design test simulator. *AIAA-Paper No. 96-4029*

Junkins, J.L.; Kim Y. 1993: *Dynamics and controls of flexible structures.* Washington, DC: AIAA

Press, W.H.; Teukolsky, S.A.; Vetterling, W.T.; Flannery, B.P. 1988: *Numerical recipes in C: The art of scientific computing.* Cambridge, UK: Cambridge University Press

Sobieszczanski-Sobieski, J. 1982: A linear decomposition method for large optimization problems - Blueprint for development. *NASA Technical Memorandum 83248*

Sobieszczanski-Sobieski, J. 1988: Optimization by decomposition: A step from hierarchic to non-hierarchic systems. 2nd NASA/Air Force Symp. on Recent Advances in Multidisciplinary Analysis and Optimization (held in Hampton, VA)

Sobieszczanski-Sobieski, J. 1990: Sensitivity of complex, internally coupled systems. *AIAA J.* **28**, 153–160

Sobieszczanski-Sobieski, J.; Bloebaum, C.L.; Hajela, P. 1991: Sensitivity of control-augmented structure obtained by a system decomposition method. *AIAA J.* **29**, 264-270

Sun Microsystems 1995a: *The JAVA language environment, a white paper.* Mountain View, CA

Sun Microsystems 1995b: *The JAVA language specification.* Mountain View, CA

Weston, R.P.; Townsend, J.C.; Eidson, T.M.; Gates, R.L. 1994: A distributed computing environment for multidisciplinary design. *AIAA 94-4372-CP*

Received March 14, 1997
Revised manuscript received June 11, 1997