AIAA-99-1348

AIRCRAFT DESIGN PROBLEM IMPLEMENTATION UNDER THE COMMON OBJECT REQUEST BROKER ARCHITECTURE

R. Sistla†, A. R. Dovi†, P. Su†† & R. Shanmugasundaram‡
Computer Sciences Corporation
3217 N. Armistead Avenue, Hampton, Virginia 23666

Abstract

The paper describes a component-based computational environment for implementing aircraft design problems. The environment is conducive to taking advantage of the parallelisms inherent in the problem and distribute the individual disciplines on machines most appropriate to their needs while insulating the developer and the user from the complexity of the underlying communications constructs. Common Object Request Broker Architecture and Java programming language are used to encapsulate discipline codes as "objects". An interface file identifies all the information needed by a user of the object. Legacy codes are "wrapped" using Java's native interface methodology and each such code is called from a module which implements the services of the object. A server program ties the implementation to the interface. Data and file management are accomplished using Java's database connectivity to access a commercial relational database management system. Java's Beans Development Kit is used to implement the disciplines and sub-tasks as reusable components that provide a graphical interface for user input as well as facilitate interactive and visual object connectivity and problem execution progress monitoring. This approach has been used to implement a simple aircraft design optimization problem, the analysis part of a large scale high speed civil transport design optimization problem, and a stand alone aerodynamic optimizer.

Introduction

Conventional aircraft design requires diverse engineering disciplines to execute independent of each other, in sequence, and often times iteratively and interactively. This process is further complicated by the fact that the focus, emphasis, and approach of each discipline can be quite distinct, and multiple invocations of the discipline programs are often required to arrive at a feasible design. The end result is a set of thumbprint, carpet, or, correlation plots from which a "best" design may be chosen. The whole process is vastly time-consuming and does not, in general, include all engineering disciplines early in the design process [1,2].

Preliminary design traditionally deals with disciplinary sizing and shaping, with reliance on previous designs of a vehicle type. The vehicle must sustain several critical flight (loading) conditions throughout the operational envelope during which the loads are typically redistributed due to aeroelastic effects. Analytical and test verification of designs may be performed throughout the design process [1]. Other conditions such as flutter, divergence, control reversal, and gust loading must also be addressed. However, these conditions may not be included in more detail until static strength design is completed [3,4]. Information from each discipline is analyzed and the design is modified in a sequential, iterative process.

The traditional approach results in a design procedure that is largely inflexible and computationally taxing. New techniques in multidisciplinary design optimization are aimed at improving design efficiency, design cycle time reduction, and introducing decision critical information early in the design process [5]. An earlier effort within the Framework for Interdisciplinary Design and Optimization (FIDO) project [6] used Parallel Virtual Machine (PVM) to handle communications among various discipline codes executing in a "host/slave" mode. This framework was sensitive to the host operating system and changing the analytical connectivity or switching discipline codes required major programming intervention.

The goal of the current framework is to provide a programming environment for automating the distribution of a complex computing task over a networked, heterogeneous system of computers. These computers may include engineering workstations, vector supercomputers, and parallel processing computers. The present paper describes a computational environment for multidisciplinary analysis and optimization of a High Speed Civil Transport (HSCT) aircraft, capable of concurrent analyses in a distributed computing environment and significantly reducing the design cycle time while introducing more detailed analysis early in the design process.

† Senior Computer Scientist, Senior Member, AIAA
†† Computer Scientist
‡ Member of Technical Staff, Member, AIAA

The present approach incorporates advanced aircraft design techniques within a new design framework utilizing the Common Object Request Broker Architecture (CORBA) and the **Java** programming language. The primary benefit of this system is the flexibility to demonstrate advanced technology concurrent multidisciplinary design integration techniques and the capability to introduce detailed analyses early in the design process. Such a multidisciplinary analysis and optimization (MDO) system capable of concurrent analyses using several disciplines such as aerodynamics, structures, mission/performance and optimization is under development for the High Performance Computation and Communication and Computational Aerosciences (HPCC P/CAS) program. This system uses CORBA in an Object Framework [7] for the integrated design of a HSCT aircraft configuration across a networked system of heterogeneous computers using the client-server paradigm. A central relational database is used for information interchange and file management. The goal of the design is to minimize the vehicle gross takeoff weight (GTOW) for given flight conditions and mission requirements.

The paper will describe two implementations in the aircraft design domain based on the object-oriented approach using CORBA and Java. The first is a simple representative aircraft design problem based on fast limited-fidelity discipline codes including an equivalent plate structural analysis, linear aerodynamic analysis, table lookup for propulsion and a simple range equation for performance fuel weight estimation. The second implementation is of a standalone aerodynamic optimizer based on high-fidelity discipline codes including a nonlinear aerodynamics code, parametrized geometry codes and an optimization code. The components of the technologies used, CORBA and Java, will be described and their use in the implementation will be detailed. Components developed for the second implementation were reused without change in the Analysis part of a large scale design of a HSCT aircraft.

The Framework

A multidisciplinary analysis and optimization system has been developed that is capable of concurrent analyses using several disciplines such as aerodynamics, structures, performance, propulsion, and optimization. This system, based on CORBA and the Java programming language and its Applications Programming Interface's (API)'s, uses the client-server paradigm in an Object Framework for implementing problems such as the integrated design of a high speed civil transport aircraft over a networked system of heterogeneous computers. The Beans Development Kit (BDK) is used to provide a Java Beans-based graphical interface for user input, interactive and visual object connectivity, and for monitoring the progress of problem execution. Java's Database Connectivity (JDBC) is used by the client and server objects to communicate with a central relational database. Java's Remote Method Invocation (RMI) is used on platforms where a cost-effective commercial implementation of CORBA is not available. In the three applications described in this paper, optimization plays a common role. In the case of the representative high speed civil transport design problem, the objective is to optimize the airplane weight for given cruise conditions, range, and payload requirements, subject to aerodynamic, structural, and performance constraints. The design variables include both structural thickness and geometric parameters defining the airplane shape. The framework provides the capability to switch between low-, medium-, and high-fidelity codes with ease.

Common Object Request Broker Architecture (CORBA):

The Common Object Request Broker Architecture is a specification adopted by a consortium of industry representatives known as the Object Management Group (OMG) to define a framework for developing distributed applications. CORBA allows client objects to invoke server objects across the network without having to deal with the underlying complexities of object implementation and invocation. In this model, an object is an encapsulated entity with a unique identity whose services can be accessed only through a well defined *interface*. The implementation of the object (language, operating system, other system specific aspects) as well as the location of the invoked object are transparent to the requesting client. The details of the architecture are discussed below.

CORBA can be thought of as a "software bus" [Fig. 1] connecting various objects, both application and service, on a network of computers. Objects on the bus can be used by any other objects on the bus, with the Object Request Broker (ORB) mediating the transfer of messages between them. In this configuration, there is peer-to-peer communication where servers can be clients for the services of other objects on the bus. CORBA also defines a wide range of services and facilities [8] to extend the core capabilities of the ORB.

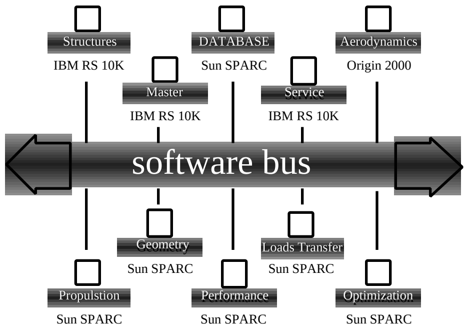


Figure 1. CORBA as a 'software bus'.

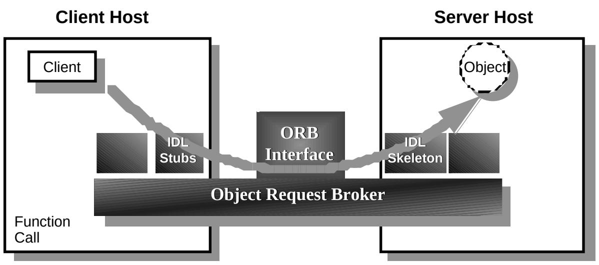


Figure 2. The Object Request Broker, ORB.

Object Request Broker (ORB):

A software implementation of the CORBA specification is called the Object Request Broker, or, ORB [Fig. 2]. The ORB mediates the transfer of messages from a program to an object on a remote networked host. The ORB delivers requests to objects and returns any responses. The key feature of the ORB is the transparency of how it facilitates the client/object communication. The client is not required to know where the target object resides, how and in what programming language it was implemented, or the operating system on the host computer. When a client makes a request, the client is not concerned whether that object is currently active and ready to accept requests. The ORB transparently activates the object, if required, before delivering the request. The client does not need to know what underlying communication mechanism the ORB uses to mediate the message passing between the client and the server. All these enable the user to generate "thin clients" i.e., all the number crunching is done on the server-side on computers most appropriate for the task. The ORB frees the application developer to focus more on the application domain issues and less about the low-level distributed system programming issues.

An ORB is one component of the OMG's Object Management Architecture (OMA). The others include the application objects, CORBA services, and CORBA facilities. Services include a) Naming Service - which allows the clients to find objects based on names, b) Trading Service - which allows clients to find objects based on their properties. CORBA facilities define a set of high-level services that applications frequently require when manipulating distributed objects.

Different commercial implementations of the ORB must all be able to talk to each other using a standard network protocol called the Internet Inter-ORB Protocol (IIOP).

Within an object framework, each object communicates with others on a peer-to-peer basis. Each object is a client of other services and a server for the services it provides. Very often, a client for one request is a server for another. This architecture facilitates network programming by allowing the creation of distributed applications as sets of cooperating reusable objects that interact as though they were implemented in a single programming language on one computer.

Interface Definition Language (IDL):

Although CORBA objects are implemented using standard programming languages, each CORBA object has a clearly defined interface, specified in the CORBA Interface Definition Language. Before a client can make a request to an object, the client must know the types of operations supported by the object. An object's interface specifies the operations and types that the object supports and thus defines the requests that can be made to that object. These object interfaces, written in IDL, are similar to classes in C++ and to interfaces in Java. IDL is a declarative language, not a programming language. It forces interfaces to be separate from object implementations. To use a discipline code, the user needs to know only what interfaces are implemented by the code.

The IDL interface is compiled by an IDL Compiler. IDL compilers are available for several programming languages. For Java, the IDL compiler produces several Java constructs which correspond to the IDL definition. The mapped constructors may be divided into those that allow a client to access an object through the object interface, and another set of constructs which allow the object to be implemented in a server.

Java and its APIs:

Java is a general-purpose concurrent class-based object-oriented programming language that fits naturally into the CORBA object orientation architecture. Java is specifically designed to have as few implementation dependencies as possible and allows application developers to write a program once and then be able to run it everywhere on the Internet. Java is robust, architecture neutral, portable and with the "just-in-time" compilers, approaches speeds comparable to those of languages such as C or FORTRAN. Java's multithreading capability provides the ability to execute multiple activities in parallel.

Java's many APIs such as RMI, JNI, JDBC and Java Beans (discussed below), make Java the programming language of choice for distributed applications, particularly those dependent on legacy codes, relational databases and requiring a graphical user interface.

Remote Method Invocation (RMI): Remote Method Invocation (RMI) is Java's alternative to CORBA's remote invocation capabilities and is designed to simplify the communication between two objects residing on different hosts. RMI is useful only for communication between Java objects, and it does not currently use a standard transmission protocol such as IIOP. This API is a handy alternative when a suitable commercial ORB is not available (or too expensive) for a specific platform. Of course, the platform should have the Java Virtual Machine (JVM) on it.

Java Native Interface (JNI): While a pure Java solution is nice in principle, realistically, for an application such as airplane design, there are several situations where it becomes necessary to use codes written in another language. Java's Native Interface methodology permits calling such legacy codes from a Java program. To make calling native methods possible, Java comes with hooks for working with system libraries and a few tools to relieve some of the associated tedium. However, the usage of native methods precludes portability. In the present framework, the portability issue would not be a problem since these methods would be used on the server side of the application.

Java Database Connectivity (JDBC): The JDBC API is a set of specifications that define how a program written in Java can communicate and interact with a database. JDBC defines how the communication is to be carried out and how the application and database interact with each other. More specifically, the JDBC API defines how an application opens a connection, communicates with a database, executes SQL statements, and retrieves query results. JDBC provides a vehicle for the exchange of SQL between Java applications and databases. JDBC classes are available for several popular commercial databases which makes it possible to switch databases on an application without being required to make any significant changes to the Java code.

JavaBeans: A JavaBean is a reusable software component that can be manipulated visually in a builder tool. The builder tools may include web page builders, visual application builders, GUI layout builders, or even server application builders. The JavaBeans API provides an environment in which a programmer can "wrap" an object as a component that can be used by other developers. JavaBeans provide the capability and a set of standards for a design or builder tool to be able to query the component package and access the object's properties through a process known as introspection.

JavaBeans have three distinct elements: Properties, Events, and Methods. Properties are the internal variables associated with a component. Events are a way for components and applications to communicate with each other. Common events include mouse movements and clicks, keys being pressed, objects receiving or losing focus, etc. In addition, the user can add custom bean events to handle special requirements. Methods are functions that the component can perform by invocation from the outside world.

JavaBeans can expose selected properties for a user to set at design time or get at run time. Properties can be as simple as a file name or as sophisticated as a color editor or arrays of data. Simple properties are displayed in the JavaBean's Property Sheet, while more sophisticated properties require custom built property editors accessible from the bean's Property Sheet. Customizers can also be written to permit users to edit multiple properties at the same time and make this graphical user interaction more user friendly.

Object Creation:

In the current implementation, Iona Technologies' implementation of the CORBA standard for the Java programming language, OrbixWeb, was chosen as the ORB. For each discipline, to be wrapped as an object, an interface file is written in IDL (Java for RMI implementation) identifying the services offered by that object, the required input parameters, the outputs, and the types of errors the object can "throw". Figure 3 shows a listing of an IDL interface for the implementation of the airplane drag estimation discipline code as "aeroDrag" object. The interface for aeroDrag has only one method, indicating that the object provides only one service - *getTotalDrag* that takes as input two floating point values: cruise angle of attack and the pressure drag coefficient, and returns a floating point value, the total drag coefficient - *cd_total*..

The object and its services, as identified in the interface file, are then implemented in Java in an *implementation* file. The services are often obtained by the execution of a discipline code. Discipline codes are in general legacy codes, written either in FORTRAN or C, and are accessed through an intermediate function which is created following the guidelines set in JNI for calls to

```
module Aerodynamics
{
    // Exception checks if error code is returned
    exception gotNegativeFlag
    {
        long errorNo;
    };

    interface aeroDrag
    {
        //Operations
        float getTotalDrag( in float alfa_cruise,
                            in float cd_pres )
                    raises (gotNegativeFlag);
    };
};
```

Figure 3. An IDL interface for the aeroDrag object.

native functions. Within the implementation file, a central relational database is queried for needed data and file information. Any required file management is done based on this file information. A third item of software associated with the creation of a discipline object is the "Server" class code. This component ties the implementation class to its IDL interface. Servers provide objects for use by clients and other servers.

In a client code (the fourth of four files associated with an object), the service *getTotalDrag* from object *aeroDrag* is obtained by first instantiating the object as

```
aeroDrag my_aeroDrag =
aeroDragHelper.bind(":aeroDragSrv","cmb");
```

where *my_aeroDrag* is an instance of *aeroDrag* and the implementation is linked to the interface by the object reference *aeroDragSrv*. The service is being requested of a server on the computer "cmb". The actual request for service is done by:

```
float cd_total =
my_aeroDrag.getTotalDrag (alfaCruise,
                          cd_pres);
```

To create a JavaBean, the client code is "wrapped" following the guidelines set by Java's Beans Development Kit. This wrapper is associated with a BeansInfo file which identifies the discipline and other properties exposed to the user through a Property Sheet and associated custom property editors. Customizers, if any, would be implemented in separate software components.

A distributed application can then be assembled in one of two ways. In the first approach, a 'master' client program can be written to implement the design analysis algorithm by making service calls to the distributed application objects. In the second approach, the custom discipline JavaBeans can be imported into either BDK's Beanbox, or into one of several commercially available application builders such as Java Studio, Visual Cafe', Jbuilder, etc. Once imported, these beans become available as icons or menu options. A user would select appropriate beans from the menu and place them in the work area. A "connection wizard" is usually used to connect the beans together to form the required analytical network. The connections are made by tying events to methods. One type of connection handles sequencing the executions of the various discipline beans. Another type of event called the Property Change Event manages the communication of changes in exposed properties between discipline beans dependent on these properties. In this environment, graphical user interface component beans can be integrated into the application including buttons, labels, text windows and chart beans to monitor execution progress. The application execution can be started by a simple clicking on a button.

In either case, when the master program is executed or when the button is pressed, the client calls are transferred to the ORB which then passes the function calls through the server code to the target object. Components of the ORB are implemented by the OrbixWeb daemons running on the server hosts or by the RMIregistries in the case of Remote Method Invocation.

Simple Airplane Design Optimization Problem

This section describes the implementation of a simplified airplane design optimization benchmark problem in the CORBA/Java based Object Framework. This problem is considered an excellent multidisciplinary optimization test case since the interplay of multiple disciplines is attempted while carrying along only a small number of design variables, constraints, and a single objective function. Figure 4 shows the example HSCT model, without engine masses and control surfaces, that was studied within this framework. Figure 5 presents the discipline segments and the information flow necessary for the design and analysis of an optimal HSCT configuration.

The principal disciplines for the design problem are: aerodynamics, structures, propulsion, and performance. The design objective is to minimize the aircraft gross take-off weight for a given cruise conditions, range, and payload requirements. The weight is minimized subject to aerodynamic, structural, and performance constraints such as the limiting values of lift and drag, maximum stresses at critical points on the wing inboard and outboard panels, and range. Design vari

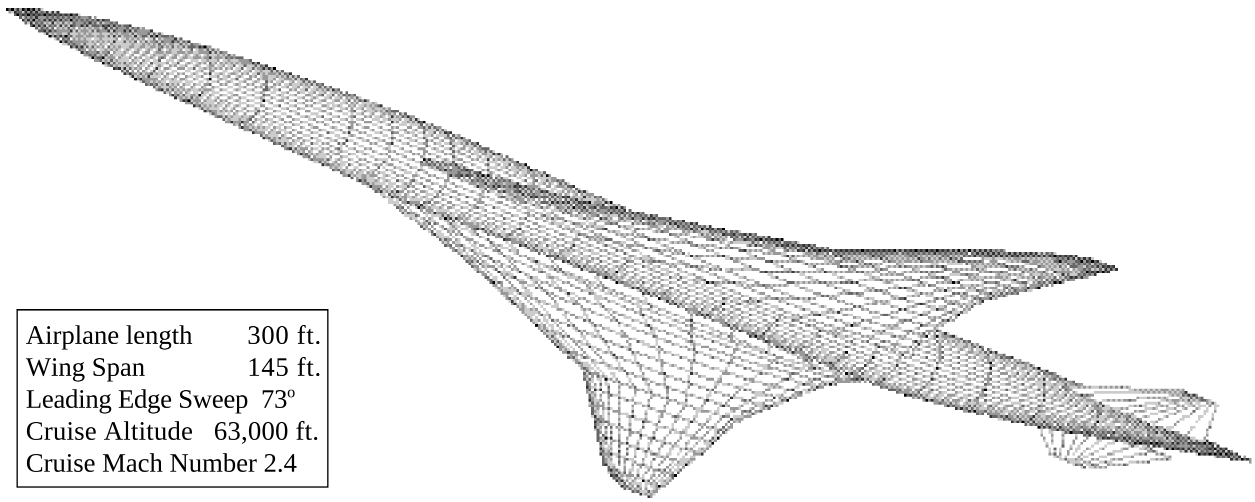


Figure 4. HSCT model problem and design point data

ables include wing sweep, wing root chord, distance to wing sweep angle break, and the inboard and outboard skin thickness.

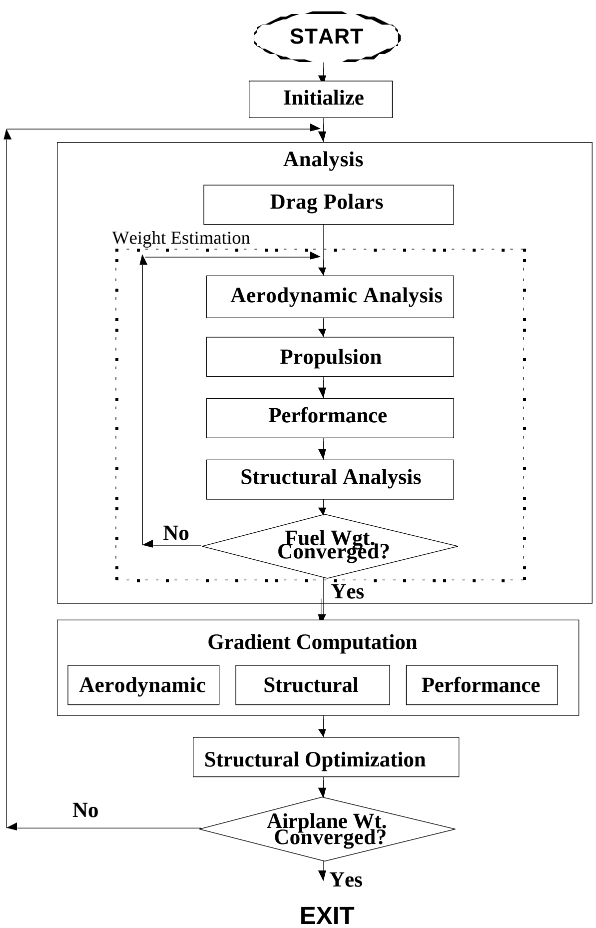


Figure 5. Flow-chart of Design Optimization Problem

After initialization, the design optimization proceeds in three phases, a) analysis, b) gradient computation, and c) optimization. The analysis phase begins with a calculation of drag polars using the medium-fidelity code Wingdes [9]. Lift and drag values for a range of angles of attack are used in generating parametric representations of aerodynamic responses. All subsequent aerodynamic analyses for that design cycle will utilize these drag polars to compute lift and drag. The next step in the analysis phase is the iteration for the airplane weight convergence. The weight iteration loop begins with a static trim analysis where force balance is computed for two different load factors: a) load factor = 1.0 for drag calculation used in performance analysis, and b) load factor = 2.5 for loads calculation in structural design. The propulsion segment computes the current fuel flow rate. The performance segment uses this flow rate to produce an estimate for fuel weight.

A structural analysis is done once during the first iteration to determine the structural weight. A loads transfer program converts the aerodynamic pressure distribution over the airplane to vertical forces on the structure at a trimmed angle-of-attack and a load factor of 2.5. The structural analysis program used here is the Equivalent Laminated Plates Solution, ELAPS [10]. The total weight is then computed as the sum of the fixed weights, structural weight, and the fuel weight. The process is repeated until the total weight converges within a predefined tolerance.

In the next phase, all the system response derivatives required by the optimizer are computed. The gradients of aerodynamic and structural constraints are computed using finite-differences. Gradients of the fuel weight with respect to design variables are obtained using a closed-form expression.

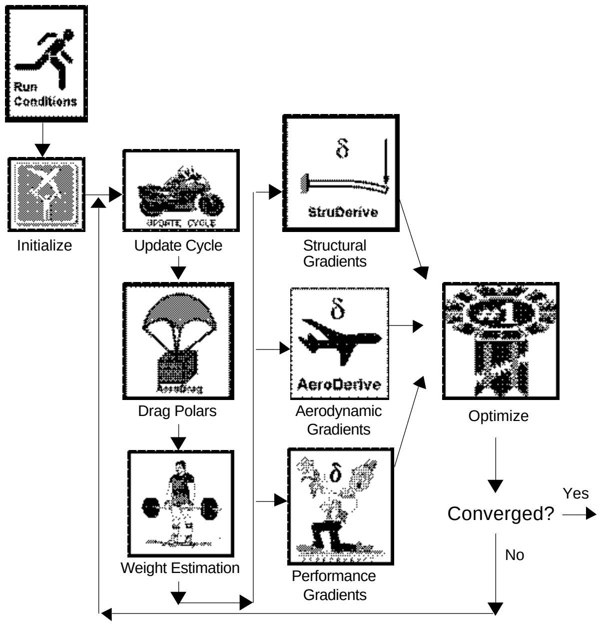


Figure 6. JavaBeans Connectivity.

In the third phase, Conmin [11] with linear approximations is used as the optimization program. Conmin uses the method of useable-feasible directions to minimize the objective function, the airplane gross weight, subject to the aforementioned design constraints and computes an updated set of values for the design variables.

The problem disciplines were wrapped as objects in the CORBA/Java environment. The client side codes were wrapped as JavaBeans and imported into BDK's Beanbox. Figure 6 shows the analytical connectivity of the discipline JavaBeans in the Beanbox. The Design Optimization Problem is presented in Fig. 5.

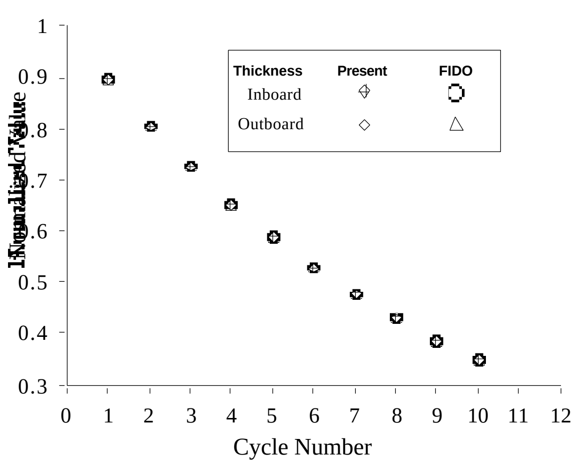


Figure 7. Variation of Structural Design Variables.

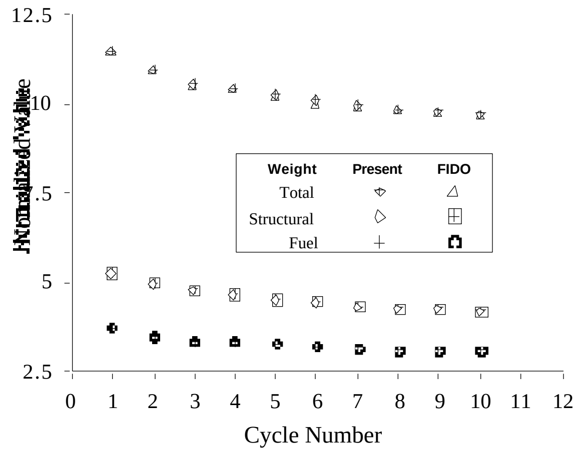


Figure 8. Variation of Airplane Weights.

A typical HSCT configuration flying at Mach 2.4 at an altitude of 63,000 feet was analyzed using the analytical connectivity shown in Fig. 5. Figure 7 shows the variation of the structural design variables with cycle number while Fig. 8 shows the variation of the airplane weight components with cycle number. The results show excellent agreement with results obtained from an implementation of the current problem in an earlier PVM based framework[12].

Stand-alone Aerodynamic Optmizer

A second application implemented in the CORBA/Java distributed computing framework is a stand-alone aerodynamic optimizer. The application uses the parallel computation of aerodynamic derivatives via automatic differentiation of the Euler/Navier-Stokes solver CFL3D [13] coupled with an optimizer and surface/volume grid deformation tools to perform an optimization to reduce the drag of a HSCT airplane configuration.

Central to any gradient-based problem is the evaluation of solution derivatives with respect to the chosen design variables. Differentiation of the CFD source code used to obtain the solution gives exact derivatives of the discrete equations, without the step size problems of finite differences[14]. A parameterized surface definition is used that relates the shape to geometric design variables. The method is a free-form deformation approach very similar to morphing techniques used in computer animation.

Figure 9 shows the analytical connectivity for the application. The design cycle begins with grid generation. Grid generation includes the following steps: a) use the updated design variables along with the parame-

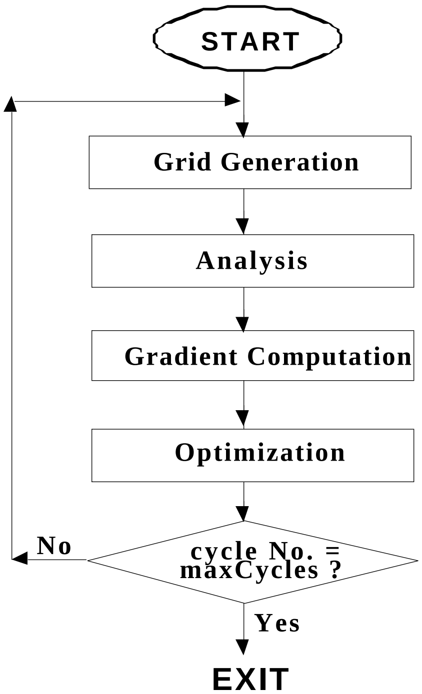


Figure 9. Stand-alone aerodynamic Optimizer.

terized grid to form an updated surface grid and its sensitivity derivatives with respect to the design variables, b) use the updated surface grid to generate a volume grid and corresponding sensitivity derivatives, and c) convert the volume grid and sensitivity data to binary format. The former may be divided from a single block to multiple blocks so that CFL3D may be executed in a parallel mode

Grid generation is followed by the analysis phase in which an analysis is conducted using CFL3D. This can be followed by a gradient computation phase. During this phase a flow solver analysis is followed by sensitivity analysis. Under assumed HSCT flight conditions the flow solution converges rapidly and hence the analysis phase is skipped in favor of a combined analysis and gradient calculation phase.

In the next phase, the optimizer computes new values of the design variables. The updated design variables are the input to the next cycle in the optimization process. The overall analysis is run for a preset number of design cycles. The unconstrained optimization problem uses 27 shape design variables to maximize the value of the lift coefficient.

The analysis codes which include the geometry analysis codes, CFL3D, and the optimizer were wrapped as objects in the CORBA/Java framework. The client-side was wrapped using JavaBeans technology to provide a graphical user interface through which the user is

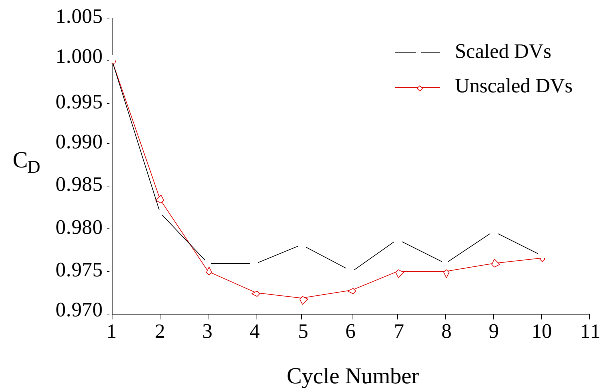


Figure 10. Variation of C_D with cycle number.

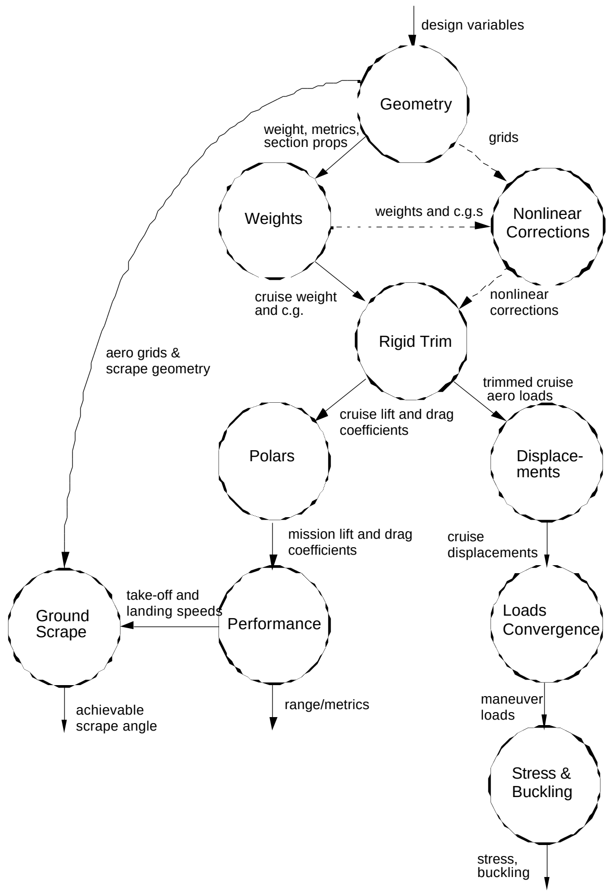
able to modify selected input conditions including options to start the analysis from scratch or continue from a selected cycle in a previous design analysis. Normalized drag results for an optimization run of 10 cycles is presented in Fig. 10. Both scaled and unscaled design variables are shown for the current implementation. These results compare identically with those obtained by the conventional process.

A High Fidelity Aircraft Design Problem

The experience gained from the earlier implementations and a number of the objects developed therein are being used in the CORBA/Java implementation of a large scale aircraft design optimization problem titled HSCT4.0. The objective is to demonstrate the application of high performance computing techniques to the problem of multidisciplinary optimization of a subsonic transport configuration using high fidelity analysis simulations early in the design process. The HSCT4.0 problem considers 27 shape design variables and 244 structural design variables (the 0/45/90/core layers in 61 optimization zones). Figure 4 shows the example model and Fig. 11 shows a high level flow chart of the multidisciplinary analysis portion of the design optimization problem.

During the Analysis phase, the values of the objective function and the constraints are evaluated. The process begins by deriving an updated geometry and corresponding linear, nonlinear aerodynamic meshes and a structural finite element mesh by applying the updated shape design variables to a baseline parameterized geometry grid.

Next, in the Weights process, structural weight from a finite element analysis, fuel weight from a performance analysis, and empirical weights representing all other components such as passengers, seats, actuators, etc. are all evaluated and assembled to provide the



Figur e 11. Flowc hart of analy sis in MDO probl em.

"as built" weight in terms of nodal weights and a total weight and center of gravity location for cruise and take off conditions.

A Nonlinear Corrections module periodically computes "corrections" for use with subsequent linear aerodynamics computations. The process uses updated linear and nonlinear aerodynamics grids, aircraft total weight for each load condition considered, along with multiple runs of the linear aerodynamics code covering a range of bracketing angles-of-attack and tail angles of incidence. For each load condition, the angle-of-attack is determined that generates the same configuration normal force as that of the nonlinear aerodynamic calculation. The nonlinear corrections are calculated as the difference between the Z components of the linear panel loads, interpolated at the angle of attack that matches the normal force, with the nonlinear panel loads that have been transferred to the linear aero grid.

The Rigid Trim calculation for the cruise condition uses the derived linear aerodynamic grids along with the nonlinear corrections to compute the angle-of-attack and tail angle of incidence, that produces the target lift coefficient (as per the weight estimation), with no net pitching moment. The resulting surface pressures and the induced drag coefficient are then determined by the final trim angle and tail angle of incidence.

Once the Rigid Trim process has been completed, the Polars, Performance, and Ground Scrape sequence of processes may proceed in parallel with the Displacements, Loads Convergence, and the Stress & Buckling process sequence.

Aircraft drag polars are calculated for the current design over a range of Mach numbers, altitudes, and angles-of-attack so as to provide the required input to the performance module. At each mission condition, lift and drag due to lift are computed from linear aerodynamics calculations that span the Mach number range. Empirical corrections are applied to drag due to lift. In addition, wave drag and friction drag are computed to obtain the total drag.

The table of lift coefficients and coefficients of drag components at the mission points are input to the performance module, the outputs from which include range, take off field length, landing field length, lift off speed, approach speed and time to climb to cruise.

The ground scrape process provides a basis for constraints such that the aircraft tail will not scrape the ground on take off or landing. The ground scrape constraints are formulated as limits on the maximum value of the take off and landing gross weights for avoiding tail scrape. Additionally, ground clearances are computed for selected airframe and engine locations at the take off and landing angles of attack and a given pitch angle.

The trimmed cruise loads are input to the Displacements module. Within the Displacements module, the pressure loads on the aerodynamic grid are transferred as consistent loads at the nodes of the structural mesh. The cruise displacements obtained from a structural analysis are then input to the Loads Convergence module.

Converged loads for selected flight conditions are obtained through an aeroelastic analysis. At each load condition, the aircraft is trimmed for a consistent set of loads, angle of attack, and tail incidence angle. The trimmed loads are transferred to the structural nodes. Displacements computed from a structural analysis are then used to deform the aerodynamic grid. The aircraft is then trimmed at this new configuration and the trim loads transferred to the aircraft structure, followed by another structural analysis. This process is repeated till the trimmed loads (and the corresponding displacements) converge to a pre-set tolerance.

These converged loads for the selected load conditions are input to the Stress and Buckling module. In this module, the Hoffman Stress Failure Index (SFI) is computed for every face sheet for each of 2260 elements for 7 flight conditions. A buckling load factor (BLF) is computed for each of the elements in terms of the in-plane stress resultants.

All components of the Analysis part of the HSCT4.0 problem have been implemented as objects in the current CORBA/Java environment. Some objects developed for the earlier examples, such as the Geometry and Nonlinear Corrections objects from the Standalone Aerodynamic Optimizer, were used without modification.

Conclusion

A component-based computational environment for implementing aircraft design problems has been described. The environment is conducive to taking advantage of the parallelisms inherent in the problem and distribute the individual disciplines on machines most appropriate to their needs while insulating the developer and the user from the complexity of the underlying communications constructs. CORBA and the Java programming language are used to encapsulate discipline codes as “objects”.

Under the CORBA a multidiscipline analysis and optimization system has been developed that is capable of concurrent analysis. The primary disciplines considered in this paper are aerodynamics and structures coupled with a formal optimization technique. All objects are implemented using Java, with each CORBA object having a clearly defined interface. Java is a general-purpose concurrent class-based object oriented programming language. Java is robust, architecture neutral, portable and with just-in-time compilers, approaches speeds comparable to those of languages such as C and FORTRAN. While a pure Java solution is nice in principle, realistically, for an application such as aircraft design it becomes necessary to use codes written in other languages. This is permitted through Java’s Native Interface Techniques, which permits codes written in other languages to be called from a Java program. The Java JDBC API allows Java programs to communicate and interact with a central database. JavaBeans have been used to implement the objects as reusable software components that permit user interaction and visual object connectivity. JavaBeans allow the user to select properties at design time or access at run time.

Three design problems were presented. The first example is the “Simple Airplane Design Problem” which is multidisciplinary in nature. The contributing disciplines are aerodynamics, propulsion, performance and structures coupled with a formal optimization technique. Results show excellent agreement with earlier benchmark results. The second example is the “Stand-alone Aerodynamic Optimizer”. Results of the optimization compared identically with those obtained by the conventional process. Good design convergence was achieved within ten design cycles. Finally the implementation of the analysis portion of a large scale high speed civil transport design optimization problem is described.

A component-based multidisciplinary analysis and optimization design framework that will allow global coupling of several uncoupled engineering disciplines has been demonstrated using the CORBA. Contributing disciplines function concurrently as objects in the framework. Design trends compared well with results obtained from conventional techniques.

Acknowledgment

This work was supported by NASA Contract NAS1-20431.

References

1. D'Vari, R., and Baker, M., "A Static and Dynamic Aeroelastic Loads and Sensitivity Analysis for Structural Loads Optimization and Its Application to Transport Aircraft," AIAA Paper 93-1643, April 1993.
2. Raymer, D.P., Aircraft Design: A Conceptual Approach, AIAA Education Series, AIAA, Washington, DC, 1992.
3. Radovchich, N. A., "Some Experiences in Aeroelastic Design of Structures[PADS].", Recent Experiences in Multidisciplinary Analysis and Optimization, Part 1, NASA CP-2327, April 1984, pp 455-503.
4. Ladner, F.K., and Roch, A. J., " A Summary of the Design Synthesis Process," Society of Aeronautical Weight Engineers Paper 907, 1972.
5. Salas, A. O., and Townsend, J. C., “Framework Requirements for MDO Application Development,” AIAA Paper 98-4740, September 1998.
6. Townsend, J. C., Weston, R. P., and Eidson, T. M., "A Programming Environment for Distributed Complex Computing. An Overview of the Framework for Interdisciplinary Design Optimization (FIDO) Project, NASA TM 109058, December 1993.
7. Steve Vinoski, “CORBA: Integrating diverse applications within distributed Heterogeneous Environments”, IEEE Communications, Vol. 14, No. 2, February 1997.
8. Object Management Group, The Common Object Request Broker: Architecture & Specification, 2.0 ed., July 1995.

9. Carlson, H. W., and Walkley, K. B., “Numerical Methods and a Computer Program for Subsonic and Supersonic Aerodynamic Design and Analysis of Wings with Attainable Thrust Considerations,” NASA Contractor Report, NASA CR-3808, August 1984.

10. Giles, G. L., “Equivalent Plate Analysis of Aircraft Wing Box Structures with General Planform Geometry”, Journal of Aircraft, Vol. 23, No. 11, 1986, pp. 859-864.

11. Vanderplaats, G. N., “CONMIN - a FORTRAN Program for Constrained Function Minimization User’s Manual”, NASA TM-X-62282, August 1973.

12. Weston, R. P., Townsend, J. C., Eidson, T. M., Gates, R. L., “A Distributed Computing Environment for Multidisciplinary Design”, AIAA Paper 94-4372, September 1994.

13. Sherrie L. K., Biedron, R. T., Rumsey, C. L., "CFL3D User's Manual (Version 5.0)", e-mail: r.t.biedron@larc.nasa.gov, NASA Langley Research Center, November 1996.

14. Biedron, R. T., Samareh, J. A., and Green, L. L. “Parallel Computation of Sensitivity Derivatives with application to Aerodynamic Optimization of a Wing,” To appear in the proceedings of the 1998 NASA Computational Aerosciences Workshop, Moffett Field, CA., August 1998.