Abstract

Multi-disciplinary Design Optimization is gaining popularity and is being used increasingly to design complex engineering systems. This requires integration of disciplinary analysis modules. A natural MDO strategy is based on a system analysis, used along with an optimiser. System analysis of a multi-disciplinary system calls for integration of disciplinary analyses. Disciplinary analyses typically evolve without any centralised coordination and are often distributed over an intranet. Integrating such analysis modules is usually quite complex. A framework that can simplify the task of integration and allow user to focus on the problem itself is required.

Based on the requirements published in the literature, MDO framework software is designed and implemented.

Keywords: MDO framework, distributed computing

Introduction

Complex engineering design e.g. aerospace vehicle design consists of multiple disciplines coupled with each other. The problem can be viewed as a system made up of multiple sub-systems. Performing the system level analysis is an involved task. System analysis includes each sub-system analysis and various sub-system interactions.

Any sub-system analysis usually translates into a disciplinary analysis or executing the analysis software for that discipline. These analyses software can be commercial software, in-house developed codes or legacy applications. For the purpose of performing system analysis various disciplinary analysis softwares need to be executed in some sequence. For a large number of disciplines it becomes humanly difficult to iterate over various softwares, generating input files and parsing output files to extract useful information. Therefore a ‘software integration system’ which helps user tie in all the disciplinary analysis softwares is required. Such a software integration system is termed as a framework.

Framework Requirements

Framework as a software system has to be developed, maintained and extended based on the needs of the engineering system designer. Framework as a generic software system and based on the research in the area of of MDO, set of requirements have been put forward by Salas and Townsend[1].

These requirements are divided into four cat-
1. Architectural Design
   - framework should provide intuitive graphical user interface (GUI)
   - framework should be designed using object oriented principles
   - framework should be extensible
   - framework should not impose unreasonable amount of overheads
   - framework should handle large problem sizes
   - framework should be based on standards

2. Problem Formulation Construction
   - framework should allow user to configure complex branching and interactive problem formulations easily without low-level programming
   - framework should allow to easily reconfigure the existing problem formulation
   - framework should support user in incorporating legacy and proprietary softwares
   - framework should allow to integrate several optimization methods

3. Problem Execution
   - framework should automate the execution and movement of data
   - framework should be able to execute multiple processes in parallel
   - framework should support process execution distributed across heterogeneous computers
   - framework should allow user to operate in a batch mode

4. Information Access
   - framework should provide database management features
   - framework should provide capability to visualize intermediate and final optimization and analysis results
   - framework should provide monitoring capability for viewing the status of an execution, including system status
   - framework should have some mechanism for fault tolerance

These top level requirements form the basis for the design of our framework.

Design Methodology

The basic objective of the framework is to allow user to focus on the given MDO problem rather than get stuck in integration issues. At the time of the development of the framework, the problems being attempted were all based on disciplinary analyses programs developed in-house. Hence the source code for each of these analyses programs was available. This simplified the framework design, and allowed us to look at subset of the requirements. Following assumptions were made in the first stage of implementation.

1. *Source code is available for analysis modules.* This constraint allows us to modify the source code for integration.
2. *All codes running on single computer.* This assumption removes the complexities of distributed computing.
3. *All system variables are scalars.* This assumption simplifies handling of the data.

The requirements that were focused on were
• providing automated mechanism for exchange of data between different analysis modules

• integrating available batch mode optimizers with analysis modules to perform simple design optimization studies

To be able to design for automated data exchange it is essential to understand the nature of the disciplinary analysis with respect to input and output.

![Figure 1: Anatomy of Analysis Module](image)

As shown in figure 1 most of the analysis modules read the input data at the beginning of the analysis module. Data is read from the input file and/or keyboard input. Output data is written at the end of the analysis module to the screen and/or output files. To exchange data between two analysis modules, it is necessary to understand the data required for each analysis module and the format in which data is accepted by analysis modules. It is easier to exchange data when the participating analysis modules are known a-priori. When developing a framework such information is not available. Therefore an entity termed as Data Server, is introduced, which will handle data exchange requests and create a data exchange link between analysis modules.

**Data Server**

Data server is designed as the repository for storage of data that needs to be exchanged between disciplinary analyses. The design of data server is stateless, so data server does not have to store information about analysis modules exchanging data.

Data server is executed as a separate process and all the analysis modules communicate using Inter Process Communication (IPC) mechanism. This communication is abstracted into a library with easy to use interface.

![Figure 2: Data Server](image)

As seen from Figure 2, input wrapper reads the input data required for analysis module from the data server. The disciplinary analysis performs the calculations, and output wrapper writes the output to the data server. So data server holds the input and output variables required for different analysis modules, thus creating an infrastructure for automatic exchange of data.

**Analysis Wrapping**

Before we can use any disciplinary analysis software in the framework it is required that the disciplinary analysis software can communicate with the framework and in particular to
data server. Depending on whether source of the disciplinary analysis is available or not,

- Modify analysis code to incorporate communication interface between data server and analysis module to read input variables from data server and writing output variables to data server.
- Write program to generate the input file from the input variables in data server and parse output file to extract output variables.

Analysis wrapping is the process of integration of analysis modules into the framework either by modifying the source or writing programs to handle input/output files. This process can be involved depending upon the complexity of input and output files for analysis software.

First stage assumptions simplify analysis wrapping as the source code can be easily modified to exchange data with data server. However, to generalize the use of any analysis software under the framework, wrapping has to be done by parsing input/output files. Second stage of framework development focuses on building tools to help user build parsers for input and output files.

**Distributed Computing**

It is not possible to use all existing analysis software if we constrain the execution of analysis modules to single machine. In general framework should be capable of running analysis modules on any computer connected by a network. This emphasizes on another aspect of framework development i.e. distributed computing.

Various technologies for heterogeneous computing exist today. These include Remote Procedural Call (RPC), Parallel Virtual Machine (PVM), Message Passing Interface (MPI) to name a few. These mechanisms allow executions of processes on heterogeneous computing platforms and support network communication. Among the distributed computing paradigm Common Object Request Broker Architecture (CORBA) is gaining popularity.

CORBA based programs are inter-operable irrespective of the type of computer, network, operating system or programming language. In addition to remote execution, CORBA provides services for Naming, Transaction Monitoring, Error handling, etc. This makes CORBA ideally suited for distributed computing.

**Graphical User Interface**

Graphical user interface (GUI) is an important component of the framework that allows the system designers to use the framework effectively. It has been decided to use web-based GUI for most part of the framework interaction. This enables platform independent usage.

**Applications**

**Airborne Early Warning System Design**

Airborne Early Warning System (AEWS) is a multi-disciplinary system consisting of four disciplines.[3] Framework was used to couple the four disciplinary analysis modules to an optimizer as shown in figure 3.

**3D Air-Intake Duct Design**

Air-intake for an aircraft is a S-duct. Design of such duct using low fidelity analysis has been carried out using the framework.[4] As shown in figure 4 the data is exchanged between analysis modules not only thru dataserver, but through
Figure 3: Airborne Early Warning System using Framework

Figure 4: 3D Duct Design using Framework

files. The analysis modules running on different computers connected over the Internet.

Summary

Various framework aspects discussed cover most of the requirements listed earlier. Framework development uses standard and open technologies to protect the investment as well as maintain extensibility. Framework is designed as set of small components handling subset of functionality of the framework. This keeps the design of each component simple and independent of design of other components.

References


