PARAMETRIC OPTIMIZATION DESIGN SYSTEM FOR A FLUID DOMAIN ASSEMBLY

by

Matthew J. Fisher

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

Brigham Young University

August 2008

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Matthew J. Fisher

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	C. Greg Jensen, Chair
Date	Steven E. Gorrell
Date	Kenneth W. Chase

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Matthew J. Fisher in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

C. Greg Jensen
Chair, Graduate Committee

Accepted for the Department

Matthew R. Jones
Graduate Coordinator

Accepted for the College

Alan R. Parkinson
Dean, Ira A. Fulton College of Engineering and Technology

ABSTRACT

PARAMETRIC OPTIMIZATION DESIGN SYSTEM FOR A FLUID DOMAIN ASSEMBLY

Matthew J. Fisher

Department of Mechanical Engineering

Master of Science

Automated solid modeling, integrated with computational fluid dynamics (CFD) and optimization of a 3D jet turbine engine has never been accomplished. This is due mainly to the computational power required, and the lack of associative parametric modeling tools and techniques necessary to adjust and optimize the design. As an example, the fluid domain of a simple household fan with three blades may contain 500,000 elements per blade passage. Therefore, a complete turbine engine that includes many stages, with sets of thirty or more blades each, will have hundreds of millions of elements. The fluid domains associated with each blade creates a nearly incomprehensible challenge.

One method of organizing and passing geometric and non-geometric data is through the utilization of knowledge based engineering (KBE). The focus of this thesis

will be the development of a set of techniques utilizing KBE principles to analyze an assembly which includes multiple fluid domains. This comprehensive system will be referred to as the Parametric Optimization Design System (PODS).

ACKNOWLEDGMENTS

I wish to thank my advisor Dr Jensen for the support and direction that he has given me, along with the assistance of my committee. To my fellow graduate students, I want to thank Jon Lund, Brett Black and the rest of the ParaCAD Lab for all of their help and support. From Altair Engineering I want to acknowledge all of the HyperMesh advice I received from Matt King. Sheldon Hancock gave me enormous help with understanding different techniques with both C# and database design.

Finally, I would like to thank my wife, Sheila, and my children, Jackson and Maggie, for their patience and encouragement, and for the sacrifice that they have made while I completed this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

In industry, intense competition creates pressure to continually upgrade and produce future product innovation. The primary motivation of any product development enterprise is to shorten the time necessary in each step of a particular product lifecycle. One of the most difficult steps exists with the design and analysis of a product. With any given product, there may be many solutions that meet or exceed the final design goals and many that do not. Selection of the correct design is often based on look, feel and the engineer's judgment. So many variables can play a part in determining which product design will be successful, and yet, because of the need to produce new products quickly, choices are often made rashly and erroneously, leading to turn-backs, recalls and outright cancellation of programs. Such errors can be costly in terms of development time market share and liability.

The early design process consists of three main steps: modeling, analysis and optimization. The modeling stage should be fairly iterative, considering all of the different solutions that exist for any given problem. The analysis stage shows which designs are feasible, but without optimization requires many iterations to achieve the desired goals. Finally, the optimization stage includes both of the previous stages, since optimization uses the results from analysis to adjust designs to approach an improved solution. If the early design process is automated, so different models can be optimized

quickly and efficiently, more designs can be reviewed and evaluated in shorter periods of time. In addition, decisions can be made with more data and analysis to support them.

In fluid analysis, optimization of a single fluid domain is relatively simple as long as the number and type of features does not change from model to model; however, when the number of domains or the number of features changes within a model, normal optimization fails. The result is then dependent upon existing knowledge or a best choice of working prototypes. For example, the number of ribs in an airplane wing has historically been selected based on a working knowledge of what has worked in the past. Recently, Altair OptiStruct was used to improve Airbus A380 leading edge rib design, which resulted in a 40 percent weight savings (Schuhmacher, 2006). Computer programs are increasingly able to find better results to existing designs. Using our existing knowledge of structural analysis and fluid dynamics in computer-aided engineering, astonishing realities are appearing.

CFD often requires enormous numbers of nodes and elements when analyzing 3D models. Historically, the computational power of the standard desktop computer has not been able to handle the number of nodes and elements required for multiple domains, but that is changing. Memory is becoming cheaper and faster. Multiple processors shorten computation time. Large models no longer require super computers to run thousands of iterations. With this increasing capability, methods must be developed to improve the process of early design, making it more efficient and more effective in arriving at superior designs.

Most of the problems in CFD lie in the computational power required to optimize multiple fluid domains, each with discreet features or instances within those domains.

Additionally, CFD by itself is not accurate enough for engine design. The problem presented for analysis in this thesis is the optimization of a full 360 degree 3D model of a jet turbine engine. To accomplish this end, PODS was developed. A key component of PODS includes the storage of all parameters and results within a common database system. By using a common database, the information can be assigned, maintained transferred and updated easily and quickly. A custom routine was developed for each commercial software package that interfaces with the database. In order for these routines to work, additional software components were developed for PODS. These include:

1. Parametric dimensioning of all models
2. Parametric assembly constraints or associatively between components
3. Parametric meshing of fluid domains
4. Rapid storage and retrieval of key parameters
5. Rapid convergence of iterative design searches

All of these components must be integrated efficiently and effectively into a comprehensive and user-friendly system, PODS.

1.1 Problem Statement

The difficulty in the optimization of a 3D jet turbine engine lies primarily in the lack of associative parametric modeling necessary to adjust and optimize the design and the computational power required to analyze a given design. All model driven optimization requires some type of parametric model. Either parametrics can be handled with a model that contains parametric dimensions, or with a program that recreates the

model for each iteration. The CFD analysis also requires that grid independence is achieved with respect to both length and time scales.

With a jet turbine engine, or any assembly of fluid domains requiring CFD, each part that is rotating relative to another requires a separate fluid domain for analysis. As an example, the fluid domain of a simple fan with three blades may contain 500,000 elements per blade passage. Therefore, a complete engine that includes a fan, compressor, combustor, high pressure turbine and low pressure turbine, with sets of thirty or more blades or passages each, will have hundreds of millions of elements. The automated modeling, CFD analysis of an assembly that contains multiple parts, each containing many instances of a feature, is problematic at best. The optimization of an assembly of fluid domains that is completely associative, and each component parametric, requires a system to pass data from one program to the next.

One method of organizing and passing geometric and non-geometric data is through methods that utilize KBE. KBE attempts to incorporate all engineering knowledge, both geometric and non-geometric data, into the design. The focus of this thesis is the development of PODS, which utilizes KBE principles to analyze an assembly of fluid domains.

1.2 Thesis Objective

The objective of this research is to formulate a knowledge-driven process that facilitates the analysis of multiple complex fluid domains by achieving the following:

1. Expand on the idea of CAD-centric designs by storing attribute data for discrete features in a knowledge based domain, i.e. Knowledge-centric.

2. Expand on previous research, based upon attribute data reuse, by automatically passing attribute information to discrete features within an assembly, while maintaining distinctions between instances of a module.
3. Create a system that will facilitate integration of Knowledge-centric methods with legacy models and assemblies.

1.3 Delimitations of the Problem

Two difficulties arise within the automatic meshing routines that will not be addressed in this thesis, e.g. boundary layer and interface mesh creation in batch. Boundary layer creation in batch should be addressed when the mesh can be refined enough to warrant this technique and the software is capable of handling this type of mesh. Interface mesh creation is not currently handled within the meshing system and has been built into the PODS program, but is limited in the size of the mesh that it can handle. New routines should be developed in order to refine the mesh to an acceptable level. These two things will be left for future development.

Additionally, the full engine assembly problem will be simplified to a stator, a rotor and a duct as shown in Figure 1-1. An engine can contain a fan module, multiple compressor modules, a combustor module, multiple turbine modules and enough ducts to connect each module. A single module contains sets of stators and rotors, while a combustor module may be treated as a duct. In viewing a full engine as different combinations of stators, rotors and ducts, the extension of this work to a full engine becomes a matter of running additional loops. Further research should be done using supercomputers to optimize an entire engine using these techniques.

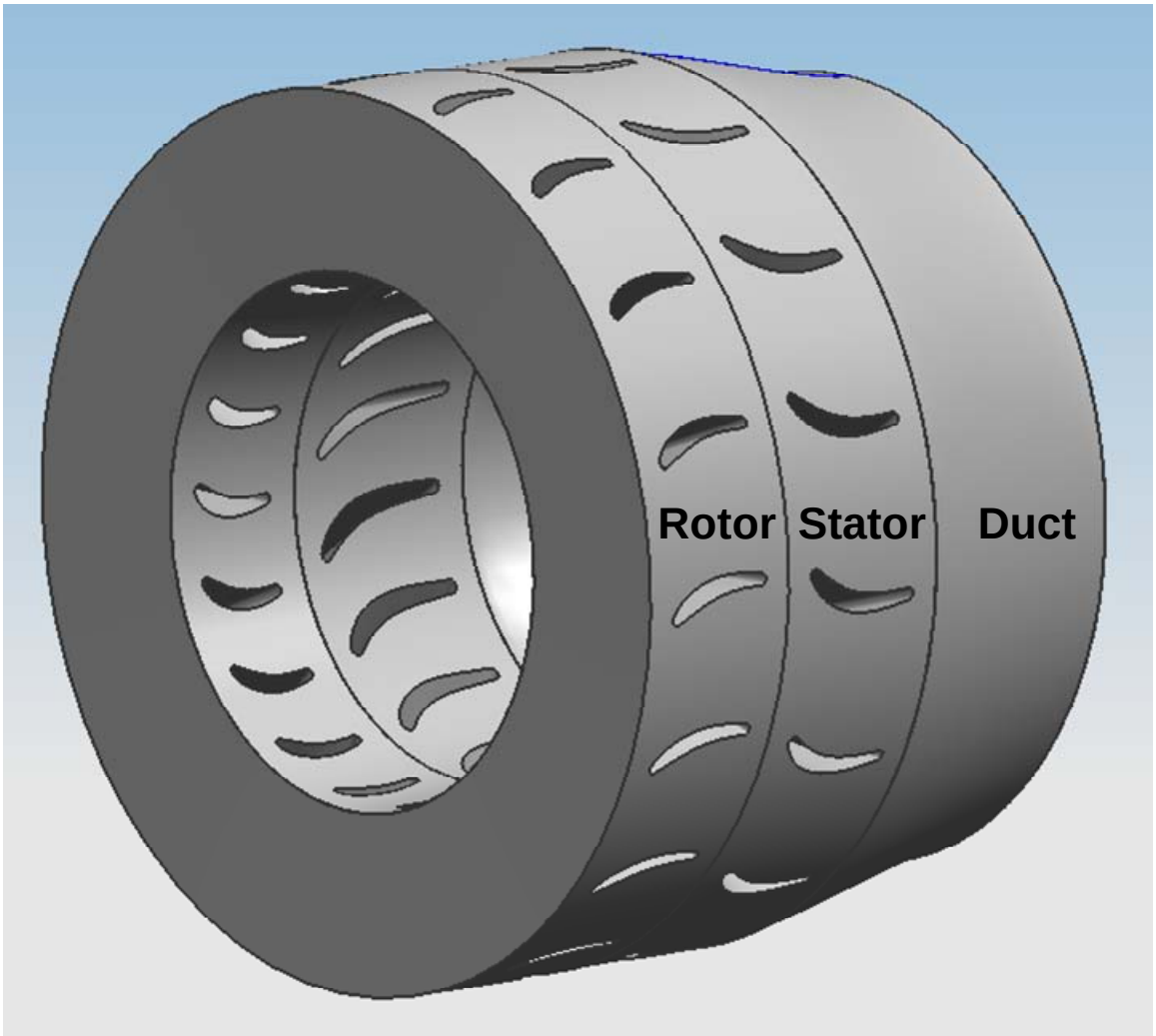


Figure 1-1 The test case will be simplified to a rotor/stator/duct combination

2 Literature Review

There are four main areas of interest in literature that are evaluated in the following sections. Namely, parametrics, KBE, optimization and jet engine design. As has already been discussed, parametric design is critical for optimization of a fluid domain assembly and an integral methodology of PODS. Without a parametric model, it becomes impossible to run iterations on the design. KBE has been incorporated into the program to handle the design data and engineering knowledge. Using KBE with optimization is of interest as an efficient way to pass knowledge to the optimization routines and change the fluid models. Finally, the research that has been done in the area of jet engine design is directly applicable to this thesis. The literature for each of these topics will help to explain the methods behind and foundation on which PODS is architected.

2.1 Parametric Equations

Parametric equations are the foundation of parametric modeling. Through the use of parametric equations a single domain can be parameterized in a way to be completely robust. Parametric equations are also used in the core of any CAD modeling program. A parametric equation is one that defines many different sets of data depending on the inputs to that equation. As an example, let us look at the implicit equation:

$$f(x, y) = 0 \quad \textbf{(2-1)}$$

This can be expressed as the two following parametric equations:

$$x = \frac{x(t)}{w(t)} \quad \textbf{(2-2)}$$

$$y = \frac{y(t)}{w(t)} \quad \textbf{(2-3)}$$

This form is referred to as a rational curve. The explicit form is expressed:

$$y = f(x) \quad \textbf{(2-4)}$$

Parametric equations for representing surfaces are simpler for use in computer applications. This results from having independent equations in the form of x, y and z parameters. Parametric equations can also be expressed in one of two forms, as power series or as a Bernstein polynomial. The following is the power series form:

$$P(t) = \sum_{i=0}^{n} p_i t^i \quad \textbf{(2-5)}$$

The Bernstein polynomial form is as follows:

$$B(t) = \sum_{i=0}^{n} b_i \binom{n}{i} (1-t)^{n-i} t^i \quad \textbf{(2-6)}$$

Where the binomial coefficient is defined as:

$$\binom{n}{i} = \frac{n!}{i!(n-i)!} \quad \textbf{(2-7)}$$

However, the Bernstein polynomial has some great advantages. Either of the parametric equation forms can be used to describe a Bezier curve. Bezier developed a system to make it simpler to describe and control curves through the use of control points. One convenient result of Bezier curves is how they relate to the Bernstein

polynomial. The coefficient b_i also matches the coordinate of the b-spline control point. This form makes it very simple to control a spline, or even a surface, in 3D space (Sederberg, 2007). This background in parametric equations and specifically Bernstein polynomials provides the basis for creating a parametric airfoil in jet turbine design.

2.2 Parametric Modeling

Can a model be built with established relationships to allow for the modification of a parameter and have the model update accordingly? Parametric models are built on parametric equations but take the idea one step further. Where parametric equations can control a surface or shape, parametric models use similar equations to relate surfaces and shapes to each other. There has been a great deal of literature addressing the creation of CAD models with parametric ability. One author discusses the difficulties that arise between the two different modeling techniques employed by CAD programs, when trying to create parametric entities. The first method is called Constructive Solid Geometry (CSG) and the second is boundary representation (b-rep). Both methods have their advantages for different applications (Shapiro and Vossler, 1995).

CSG is a method which employs definitions within Euclidean space to define a solid object. These rules include, but are not limited to, the solid being movable, having an inside and an outside, and being able to perform Boolean operations where the resulting geometry is still solid. In other words, CSG is based on the parametrics of combining shape primitives (Requicha, 1980).

The b-rep method defines the boundary of a shape or object, and has some advantages over the CSG method in its ability to define very distinct shapes. In

comparison the CSG method is dependant on the manipulation of primitives. One of the disadvantages of the b-rep method lies within its ability to identify inner and outer regions of a given shape. Using both methods in solid modeling overcomes many of the disadvantages found by using either method alone. Most CAD programs use hybrid approaches to model solid objects. For example, in order to create a solid that is based on a non-primitive shape, the shape may be sketched and swept along a path. With the basic shape defined, dimensional controls can be integrated which make the model updatable.

Parametric modeling involves the parameterization of the basic sketches or shapes employed. Constraints become a key component to ensure that the design intent remains unchanged, even with modifications to specific parameters. (Mendgen and Anderl, 1995). Constraints allow the user to apply general knowledge about the design in a way that traditional dimensions will not. The combination of dimensions and constraints make parametric design possible for both models and assemblies. In most commercial packages today, all of these techniques are employed with parameterization. Parameterization integrates user defined equations into the dimension fields. This allows dimensions to reference each other or specified constants. The user then has the ability to set limits on the amount or way a model changes. If done correctly, a model will always update in a feasible way. Full model optimization is dependent on parametrics.

> "Since the optimal shape is highly dependent on the design parameterization, definition of an adequate parameterization with manufacturing in mind, will avoid impracticable designs. Also, changing the part's geometric shape every design iteration, to update design parameters, which as been traditionally a complicated and tedious process, is now totally automated. Indeed, all that is required is to update the values of the independent set of variable design parameters." (Lindby and Santos, 1994, 1485)

Lindby and Santos first proposed using parametric solid modeling for shape optimization. At the time, recent advances made it possible to link solid modeling and meshing programs and then run optimization. Parametrics made this possible due to the ability of a parametric model to change continuously or discretely within certain limits. Further advances apply parametrics to the assembly level, allowing dimensions, equations and constraints to relate to different components of an assembly.

Along with the development of parametrics, modeling programs come with an Application Programming Interface (API).The API allows a user to either manipulate the modeling program itself or access the same commands available through the use of the Graphical User Interface (GUI). APIs allow an engineer to build a model using a programming language rather than a Windows style interface. Rather than design a parametric model, a programmed model can take inputs and rebuild the model with new parameters for each iteration.

> "The programmatic approach requires two major factors. First, an engineer with computer programming skills as well as in depth knowledge of a CAD's particular macro language. These languages can be quite complex, are not often taught in major universities or industry and are not well documented. Second, the time required up front to create the parametric program is often more time-consuming then creating a parametric model interactively." (Rhome et al., 2000, 661)

As an example of how a programmatic approach can be used, Rhome discusses development of an independent program that would allow a user to supply desired shapes and Cartesian values in order to build a model. Rhome's method includes routines for interfacing with multiple commercial CAD packages, and builds the model in those packages. One drawback is the program will only support primitive modeling; otherwise the complexity of modeling in this manner requires the same learning curve as learning a new CAD package. However, the idea of converting the model in one commercial

package to another has great potential, especially if an interactive model could be converted to usable code for parametric programming of models. NX4 has a function called Journaling that will, with limitations, record interactive modeling as JAVA, C++ or VB code, which are all languages supported by their API. Journaling may virtually eliminate the need for skilled programmers to create parametric models and set up optimization programs. Even until Journaling is fully developed, interactive modeling still remains the simplest and quickest way to create parametric models.

2.3 Knowledge Based Engineering

With either a fully developed parametric or a programmatic model, additional information can be stored for use in meshing and analysis. KBE is an acronym that has been used in several ways. In its simplest form, KBE is engineering design "based on knowledge, reasoning, perception, and common sense" (Vogel, 1989, 6). The following list shows some parameters that can and should be incorporated into KBE:

- Element size
- Element type
- Optimization routines
- Structural analysis
- Failure analysis
- Dynamic loads
- Vibration response
- Deflections
- Heat Treatment
- Metallurgy
- Corrosion
- Fastening/Welding/Joining
- Tolerances
- Component part interactions
- Component part interference
- Control systems
- Electronics
- Cost
- Man hours
- Time to market

In contrast to KBE, Computer Aided Engineering involves computation intensive processes that require advanced computers (e.g. CFD, FEA). These processes involve

solving complex systems of equations to find specific analytical results. There is an art or feel to CAE with certain parameters that lends itself well to be utilized in KBE applications. Mesh creation and refinement easily fall within the realm of KBE. CAE and KBE have traditionally been separate, but many have recognized the need for further integration of the two.

> "Integration of high-end mechanical computer-aided design (MCAD) systems with design activities and design knowledge would allow manufacturers to automate their engineering and manufacturing processes, capturing the design rules, experience and expertise and leveraging it during new product development" (Zhongtu et al., 2004, 1).

Zhongtu also describes a method of integrating design knowledge into a CAD system, which allows for simpler execution of tasks within the CAD environment. Another system of integrating design intent or knowledge exists through the use of iSIGHT-FD. iSIGHT-FD allows the user to integrate models and design tasks and then optimize the solution. This program is able to access commercial and custom programs through the use of command line or batch commands. With experience, individual programs can be written to run any additional routines that are not part of the standard functions of iSIGHT-FD. This allows KBE to be integrated into each stage of the design process, from CAD through post-processing.

One of the earliest examples of integrating KBE with CAD/CAE was proposed by Brown. He explained that one of the largest problems with incorporating knowledge into the program is that knowledge is hidden from the user. It then becomes necessary for the user to rely on the output data and trust that the algorithms were developed correctly. Brown proposed the use of a scripting interface to supply the rules and knowledge, and then output equations that are reviewed by the engineer and then evaluating the equations with a number of given parameters to access the results (Brown, 1988). Brown's method

ensures that the output is still valid, but requires the user to learn the scripting interface. MathCAD is a good example of a scripting interface that can be viewed to check accuracy of all equations that have been entered. The disadvantage is the processing time required for these types of interfaces.

Another aspect of KBE important to this thesis, involves the process of passing geometric and non-geometric data from parts to sub parts or from assemblies to subassemblies. The importance for having data transferred automatically within assemblies is more understandable when different groups of an organization are involved and subassemblies from those groups need to fit together properly. If KBE is integrated into the model, all of these interfaces will automatically update (Chern, 1992). This idea is essential when dealing with a jet turbine engine. With all the different components that are designed by different groups, it becomes detrimental that all knowledge be passed from group to group.

As a more broad definition, KBE is being developed by Siemens in a package called Knowledge Fusion (KF). The intent within KF is to accomplish both of the steps of storing geometric and non-geometric data in a scripting language for use within parts and also in assemblies. KF is an independent scripting language and user interface utilized by Siemens NX suite of programs. KFs objective is to integrate design intent into the model and aid the user by allowing more ability to control the model with KBE. The following block class is an example of KF script in NX:

```
#! NX/KF 3.0
DefClass: myblock (ug_base_part);
(Number Modifiable Parameter) length: ;
(Number Modifiable Parameter) height: 75;
(Boolean) height_sens: -same:;
(Number Modifiable Parameter) width: 75;
(Boolean) width_sens: -same:;
(Boolean Modifiable Parameter) same: FALSE;
```

```
(Child) BLOCK: {
      Class,      if(length:>25)then(ug_sphere)
else(ug_block);
      Length, length:;
      Width, if(same:) then(length:) else(width:);
      Height,if(same:) then(length:) else(height:);
      Diameter, length:;
      Center, Point(length:/2,length:/2,length:/2);
};
```

The example code shows how design intent can be built into the model by using logic statements to control the creation method. In the example script, the Length, Width and Height parameters are interdependent, and determine whether the resulting solid is a sphere or block. The difficulty with KF lies in the need to learn a new system and/or new language. The KF package is not documented well enough to handle a model of great complexity. Some other limitations arose when trying to interface with different database systems.

One of the largest problems with current analysis software is that convergence on a correct solution relies on the users' familiarity of the system in order to give the correct inputs. With the advances in CAD/CAE software that have been made since 1988, engineers need to be able to enter knowledge into their design in order to perform optimization. Baizet discusses the complexities of applying KBE on a large company level. Training of employees in the use of new systems as well as collecting all the available knowledge from engineers and scientists raises many difficulties (Baizet et al, 2003). KBE systems become a good alternative that is quick and efficient for interfacing with every step of the design process; enabling accurate engineering analysis by including all available knowledge. Another KBE system was used to automate the optimization of a structural analysis of an engine mount-bracket. The KBE system proved that automated modeling using KBE is in many ways more flexible for optimization than

interactive CAD modeling when interfacing with analysis software. KBE is more inclusive of the entire design intent than most CAD models (Pinfold and Chapman, 2004). Integrating KBE into a programming interface with a parametric model would, however, be more flexible than either one alone.

Recently, development in the direction of parametric CAD/KBE integration has led to design improvements for single models or components. The focus has been the development of a tool to optimize one specific part. A method of using KBE and Concurrent Engineering to design a compressor rotor was demonstrated. A team was assembled to develop a system for designing a rotor. From this team, all the working knowledge, of how to design a rotor, was pooled into a system of programs. From the gathered knowledge rotors were successfully designed in a fraction of the time that the previous process took. The conclusion was that all of the engineers and analysts current knowledge could be captured in a system, thereby reducing development time. Four rotor designs were selected after running the system for only 5 hours as opposed to the 17 man-days that were required before (Marra, 1995). The results are impressive, but the time up front to develop the system for each future part is daunting. The answer is to create a system that can read and modify any existing parametric model.

Lately, there has been a great push to use Product Data Management (PDM) systems with CAD. One study discussed a method to track assembly data in a database by storing all parts, interfaces, links and associated files in an easy to use structure. The main purpose of this study was to look at part interactions and what manufacturing methods were to be used (Bowland and Sharma, 05). A similar system was developed for creating

a KBE structure to store and retrieve engineering rules using a Microsoft DB. They were able to successfully gather and store rules provided by engineers (Fan et al., 02).

Most of these efforts have focused on either the files or links between files for their method of storage, or on how to establish the database scheme. Another approach was developed which stored parametric data in a product lifecycle management system (PLM). The method focused on how to set up the DB and the effectiveness of storing the data using a PLM system (Lund, 2006). The integration of Multidisciplinary Design Optimization (MDO) has been investigated in regards to PLM and model reuse. One study showed that the integration of MDO and PLM can be used to optimize a solution (Fife, 2005). If all aspects of the design process are DB driven, many more possibilities come into view. The common idea used in these examples was to take all of the geometric and non-geometric data and store it in a central database, which is accessible at all levels of the design process.

Passing knowledge or attributes from a part down through a meshing program and then on to analysis has been successfully accomplished at BYU by King. A utility program that stores attributes for a discrete feature and propagates that data to all instances of the base feature was developed. The system was based on CAD-centric technology, meaning that all KB information was stored in the CAD model. Each subsequent program received its input straight from the CAD model. This helped to overcome some of the difficulties seen when dealing with a discrete feature problem. As the number of blades or components changes in an assembly, the data is passed to the new feature. A key concept when trying to optimize a model with discrete features is the effective passing of data from one feature to another. Effective data passing allows faster

cycle times for iterations with different numbers of features in an array. This methodology further decreased cycle time when modifications to the mesh were needed for grid resolution (King, 2004).

Two key differences exist between King's study and this thesis. First, the models faces were tracked by knowing the order that the faces were saved from NX and then imported into HyperMesh. Rather than depend on the order of faces, PODS relies on the existing import capabilities of Altair HM. Second, rather than creating a CAD-centric system, the PODS focus is to have a DB-centric system. An extension of King's research was also developed which followed a DB-centric reasoning. A DB-centric system helped to automate the CAD to post-processing stage and also aided in the formation of data sets that could be required for different post processors. Flexible data sets also allowed any number of parameters or data to be stored, including mesh size and type (Baker, 2005). None of the research to date addresses the question of how to apply these techniques to an assembly. Another possible application arose from the research that would allow the storage of arrays of information that could be associated with a given feature. In building an associative parametric fluid domain assembly, many obstacles must be overcome, especially when dealing with the number of sections (modules) and blades involved in a full engine assembly.

2.4 KBE and Optimization

CAD software has primarily only included data regarding geometric entities. Incorporating other non-geometric data has only recently been promoted. Systems for handling non-geometric data are becoming more available.

> "A knowledge-based system is a computer program which possesses a set of facts about a specific domain of human knowledge and, by manipulating these facts intelligently, is able to solve problems which ordinarily require human intelligence. The development of knowledge-based systems is an attempt to emulate the human process of problem solving." (Balachandran, 1987, 93)

Further research discusses how the use of KB systems in optimization is "promising, but further research is necessary" (Balachandran, 1987, 115). There has also been focus on which optimization algorithms work well with "analytic and heuristic knowledge" (Borup and Parkinson, 1992, 137). Analytic is described as being based on physical laws and heuristic knowledge as being design rules, manufacturability, as well as others. KB and optimization analysis is driven by the fact that the design space is often not smooth. The design space may consist of families of curves, with discrete jumps from one to another, or constraints may block search algorithms from seeking a "zero slope" optimum. Therefore, derivative-based methods may not be effective. Other techniques (i.e. genetic algorithms, simulated annealing, flexible polyhedron search and function approximation) appear better suited to this type of decision making (Borup, 92).

A method for optimizing a bladed disk design using the energy stored in, or dissipated by, the part for the design objectives was proposed. The energy based research showed that most optimization was based on key design problems that did not always fit, or were not addressed, in the big picture. Specific examples of analysis were pointed out of friction, resonance and aerodynamics being resolved individually, with the only interaction between the results being iterative. The conclusion was that: "Current design methods require seemingly endless iteration among functional design groups, particularly between aerodynamic and structural groups" (Layton and Marra, 2000). The same can be said when trying to analyze full assemblies, like rocket motors, jet engines and

automobiles. Each component within a model often has its own group of design specialists, which has little interaction with other design groups. The integration of a complete assembly then becomes a very time consuming iterative process. This thesis attempts to respond to the needs of distinct design groups by analyzing each component concurrently as one full assembly of fluid domains.

2.5 Jet Engine Design

Significant research has been done for full jet engine analysis, however, it takes many weeks or even months to assemble and set up all of the meshes within an engine. “The reuse of well-tested and optimized design objects is an important aspect for decreasing design times, increasing design quality, and improving the predictability of designs” (Altmeyer et al. 1997, 754).

An application to automate the modeling and analysis of an engine component has been developed. The modeling development included the entire 2D engine cross-section or flowpath, however, the optimization routines included only one component or module (e.g. a compressor stage or low pressure turbine). “Considerable work will be required in order to fully integrate this methodology into the design of the entire engine as opposed to a single module” (Delap, 2003, 94). If this process can be automated, then the cycle time for an engine design can be shortened considerably. If the engine components are robust enough, then parts can be swapped with different fidelity parts and the process can run be and rerun in a matter of hours, considering many different configurations.

Traditionally, most CFD analysis consists of simplifying the model down to a 2D flow problem with established multiple feature meshes and sliding faces. This simplifies

the problems in a way that allows for faster solutions and more iterations. The idea is to resolve all meshes down to the simplest model possible. As an example, the flow through a pipe is very well understood. Even though the pipe has length and radial dimensions the problem can be simplified. The question that is asked of all analysis is if there is a line of symmetry that can be exploited. Figure 2-1 shows an orifice with a quarter section removed for visibility.

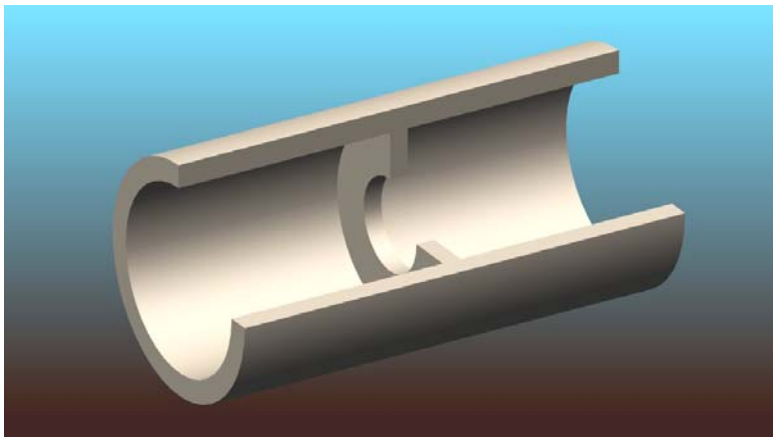


Figure 2-1 Three dimensional orifice

With an orifice, the first line of symmetry is a 180 degree cross-section of the orifice. The symmetry question is repeated until all lines of symmetry have been exploited. In the case of this orifice, there is one more line of symmetry along the axis of the pipe. By employing axial symmetry, the analysis is simplified to half of a 2D cross section. A tutorial written at BYU used this method for the final result shown in Figure 2-2. The number of elements required to mesh this surface is a fraction of what is required for a 3D analysis.

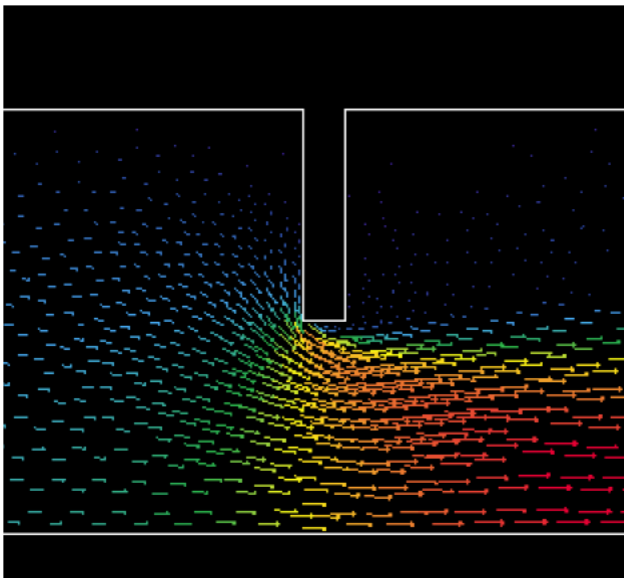


Figure 2-2 Velocity profiles through an orifice (BYU, 2006)

In jet engine models, a 2D cross-section does not account for the radial flows that are caused by revolving rotors and turbines. During preliminary design most engine analysis examines the engine parameters at the median diameter of the blades that leaves open the possibility of varying temperatures, pressures and velocities at differing radii. Most 3D analyses that have been done only address one component at a time (Weiss, J.M. and Kelecy, F.J., 1999) (Fluent News, 2007). In order to optimize an engine assembly, iterative methods are employed by passing data from one component analysis to the next. Predicting and fully understanding all of the interactions within a jet engine is seriously limited until optimization of a full 3D model is achieved. To optimize an engine assembly efficiently, better routines are needed.

To date, very little research has been done applying KBE and CFD to jet engines. A system was developed for analyzing jet engine systems, which included development of new algorithms based on working knowledge, as well as incorporating existing

systems and tying them together. This system was specific to 2D CFD jet engine analysis. (Gronstedt, 2000) In relation to such advanced systems, one researcher stated:

> "With the rapid development of the computational fluid dynamics of TM (Turbo Machinery) toward fully 3D flow with ever-increasing speed and loading and with separation and/or cavitation, some new basic problems arise and need to be explored in a more rigorous theoretical way, others need to be reconsidered and handled in a more general setting." (Liu and He, 1997, 4)

Analytical work also needs to be applied to the interactions that are occurring within the stages of a compressor or turbine, as well between components. The interactions are largely overlooked. A method to integrate the analysis for a compressor and a combustor, using two different flow solvers, namely a Reynolds-Averaged Navier-Stokes (RAN) and a Large-Eddy Simulation (LES), was achieved (Schluter et al., 2005). Schluter's research was focused primarily on the solver interactions with the idea that full jet engine analysis be the resultant goal. Again, this thesis attempts to address some of these concerns by creating a system that aids in 3D optimization and achieves this goal.

It is apparent that the tools currently employed in industry do not have the capability to easily solve a problem of this magnitude. A method is hereby proposed for use with full assemblies, that automates the process for fluid and structural analysis. The flow analysis is of particular concern, since there is little that has been done optimizing a complete engine. Structural analysis can be handled quite easily by doing it piece-by-piece, but this is not the case with a full flow field. In addition, further development should include multidisciplinary design that would examine a response surface using FEA, CFD, manufacturing, cost and other related analysis of interest to this application. (Hogge, 2002)

3 Method

PODS includes the development of five different program interfaces for commercial software in order to achieve the objective of this thesis. The handling of discrete features and interfaces can be done in several ways and this chapter will discuss the benefits and disadvantages of each. The process could be extended to different combinations of modeler, mesher and solver marketed by various vendors, as long as they have an API or scripting language interface. To fully integrate five commercial programs with different APIs presents a challenge worth overcoming. The following areas outline the order of how to create an integrated system:

1. Database interaction and scheme.
2. Modeling techniques and model conversion.
3. Mesh generation methods.
4. Fluid analysis.

The following chapter will present methods to accomplish the goals of this thesis, along with arguments and reasons for the methods that are used.

3.1 Database Storage

Every company creates a method to pass and store data. Drafting is one method of communicating data from one person to another, in this case from the designer to the

machinist or technician. Often components of large assemblies are designed by different groups within an organization and even by competing organizations. When different groups design mating components, certain dimensions are often defined with set dimensions in order to maintain mating conditions; however, even these dimensions can be optimized if the models and databases are built correctly. Any dimensions or parameters that are chosen by users or companies to optimize will be referred to as focus parameters. These focus parameters may include size, machinability, or cost related conditions. There are several methods to pass data from person-to-person or from program-to-program some of which are utilized in PODS.

One of the simplest and quickest methods to pass data from one program to another is to create programs with the ability to access the data directly from the previous program. If each program can access and modify the data directly, then there are no intermediate steps requiring additional file storage or translation time. Generally, because the information is stored in a proprietary format, this only happens between cooperating organizations.

Another alternative involves storing the data in text files for simple file parsing. An advantage of text file storage is that the use of any text editor may be used to read and edit the stored data or a file parser can be written to process the data automatically. The disadvantage lies in that each program that uses the text file must have an interpreter program to store and retrieve data from the file.

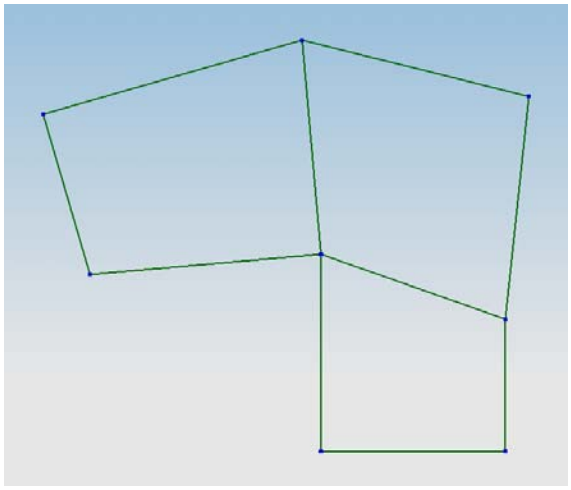
A possible tool to address this concern, called attribute standardization, was discussed by Baker. Instead of storing the data in a text file, a standardized format is set up in a common database. The database can be customized, or just used with the standard

attributes. These are generally attributes of mesh type, structure and material (Baker, 2005). Attribute standardization had little impact, and was not used, in this thesis, since the attributes of concern consisted mostly of boundary conditions.

Another approach, called Knowledge Fusion (KF), is meant to be a method of creating a knowledge-based model. KF idea uses a scripting language to create and modify the files. The disadvantage is that there is not a good central repository of all of the knowledge. KF does have some methods for accessing certain kinds of databases, but it is very limited and proved to be much easier to link to MySQL through the C++ and C# APIs.

The advantage of a central database includes the ease of storage and the ability to set up relational database structures. In a relational database, duplicate data is undesirable because it increases the risk of data not being handled correctly. As an example, a quadrilateral is defined by four corners or eight dimensions in a two coordinate system. Each quadrilateral consists of four corner points, which in turn consist of two coordinates, x and y. If an additional quadrilateral is drawn using any number of the original four points, then any one point may belong to more than one quadrilateral. In Figure 3-1, there are 8 vertices that are part of one, two or three distinct quadrilaterals.

One method of storing this data is to use a single table of quadrilaterals as in Figure 3-2. This type of structure is considered to be a flat model, and is very simple and straightforward to use. The query, "`SELECT * FROM quadtable;`", will return all of the quadrilaterals and all of the columns that are part of this table. If there were three quadrilaterals then 30 fields will be returned. In a relational database, each distinct point only needs to be stored one time.



QuadTable
QuadID
Point1x
Point1y
Point2x
Point2y
Point3x
Point3y
Point4x
Point4y
Color

Figure 3-1 Three quadrilaterals with coincident vertices **Figure 3-2 Flat model structure**

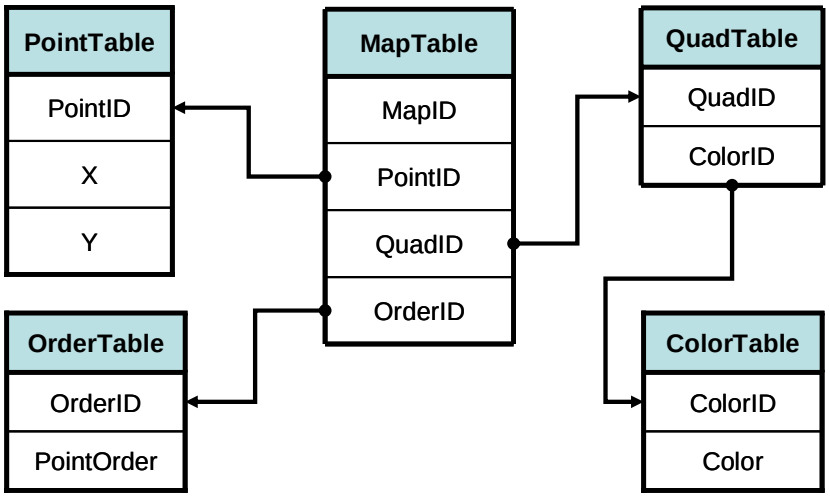


Figure 3-3 Relational database structure

In Figure 3-3 a simple relational database is shown. Each point is only stored one time, as is color and point order. The idea behind a relational database is to normalize the data so that the information is only stored in one place. Instead of mapping four nodes directly to a quad, the quad and nodes are mapped using the MapTable. In other words, each MapTable entry will link to one quad and one node, requiring four MapTable entries to define the entire quad. This prevents redundancy within the database and makes all queries more efficient. With this type of structure built into a database, the file parameters are found with this type of query:

```
SELECT quadid,pointorder,x,y,color
FROM quadtable
LEFT JOIN maptable USING (quadid)
LEFT JOIN pointtable USING (pointid)
LEFT JOIN colortable USING (colorid)
LEFT JOIN ordertable USING (orderid);
```

For the structure in Figure 3-3, the total number of fields returned for the three quadrilaterals in Figure 3-1 will be 45. With this method, the point can be called into use by any number of shapes, which saves storage space, reduces search time and most importantly, reduces the possibility of redundancy.

When iterating through model designs, instead of copying the file, storage space is saved by only storing focus parameters in a database. Database storage also saves time by simplifying the process of searching through previous iterations for the optimum. True engineering solutions do not always lie at the calculated optimum since all knowledge is difficult to capture in the design. Often other considerations must be looked at, like manufacturability, to determine which design is the most feasible. Because of this a relational database is difficult to implement.

For storage of data of a rotor or stator, a database scheme was needed that could easily store all the data that defines a rotor. The parametric rotor that was designed had a total of 95 expressions that could be stored and altered through the database. Any or all of these expressions are alterable and needed a quick method of storage. The relational database provides a wonderful framework, but a great deal of time is needed to set it up for a particular model. Since the idea of this thesis was to create a program to handle any model, no matter what expressions were chosen as focus parameters, a flat model database scheme was chosen.

3.2 Model Selection and Conversion

Modification to the design begins with manipulation of a model or assembly within the CAD domain. The model can be built with any desired method, be it interactively or programmatically. The only requirements, which restrict the choice, are ease of use and speed of iterations. The system chosen in this thesis focused on interactive models, since this is the easiest and most commonly used method throughout industry. The advantage of being able to open any interactively created model and use PODS is the potential to reach a much broader customer base. The time necessary to set up and optimize each model is drastically reduced by using PODS.

3.2.1 Blade Parametrics and Assembly Associatively

It is important that all aspects of the model be robust enough that, as variables change, the model will not fail. With PODS, robustness becomes even more critical during the optimization phase. If the initial model were to fail, or any subsequent iteration, all iterations after that point have the potential to fail. An airfoil is a cross-

section of a blade or wing. In the test case the blade is defined by the airfoil shapes at both the inner and outer edges. The airfoil for the compressor blade is addressed in more depth in section 4.2.1 but the method shown here with the following outline:

1. Create two law-defined splines
2. Create a parametric spline with controls for starting and ending, angles and curvatures
3. Offset the parametric spline by law control using the law-defined splines
4. Use bridge curves to attach the ends of the offset curves to form the leading and trailing edges of the airfoil, matching slopes where dictated by law controls.

With an assembly, it is important that the parts be associative with each other. Many methods have been examined for efficiency and effectiveness. NX has two methods that attempt to provide associativity in an assembly, interpart expressions and WAVE. Interpart expressions allow the user to set expressions applied to one part equal to expressions in other parts. WAVE defines geometric objects that are extracted from other parts, so that as the parent part changes, the child part inherits those changes. Both of these methods are purely top down associations and require careful planning from the beginning.

Another method involves creating the part programmatically. Most engineering knowledge within a company is contained in other forms, i.e. spreadsheets, databases, etc., and needs to be incorporated into models. As the knowledge changes, the models need to reflect those changes. Using the API of any 3D CAD package allows the instant creation of the model based on the existing data set. This model is easily updated by running the program and rebuilding the model with the new data set. The disadvantages

to this method, are the time required creating the program and the storage required for so many different design iterations.

Rather than using the time to develop new code for each model, one program can provide the data handling to modify existing interactively created models. All of the focus parameters are contained within a database, allowing the model to be a legacy model, as long as it is parametric. The program accesses the database and updates the original model for each iteration, instead of creating a new copy of the model. The database becomes the repository for all data iteration of the optimization routine. The database contains all of the dimensions, the mesh parameters and the solver parameters for each step of the design.

3.2.2 Mesher Preparation

The initial model must be parametric, at least with respect to any parameters that need to be optimized. The desired outcome is dependent on the ability of the model to update without crashing with subsequent iterations of the design. The model also must have the focus parameters with accessible names through an expression editor. At this stage, a user defined program is necessary to adjust the model. Each face within the model must have a boundary condition assigned. These will establish what properties will be passed on to both the mesher and solver.

If one of the design parameters includes the number of discrete features within a part, and the part number decreases or increases, the associated data must be handled correctly. When the design parameter increases, then the attributes assigned to the base feature must be passed to all the other features. If it decreases, the extra data that accompanied those features must be deleted.

The model must pass the data that indicates which surfaces are interfaces at each stage of the process. This may include the number and order of parts, or just designate the interfaces individually. This could be handled either way. Every feature within the model must have an attribute to identify what kind of boundary conditions exist for the given face. In this case, all design parameters and attributes will be stored and retrieved from the database; which provides the access necessary for the model editor.

3.2.3 Model Iteration

The same information initially set up with the model editor must be accessible in order to update the model during subsequent iterations. Two basic steps must occur, the focus parameters must be changed and all attributes must be verified. The focus parameters may include the number of features or component modifications within an assembly. The face attributes also need to be updated depending on what changes occurred to the features and components of a model.

3.3 Mesher

The mesher must have the ability to access the given data and apply a mesh that the solver can handle. There is an art to meshing that is difficult to incorporate into a program designed for any arbitrary model geometry. The challenge is to provide an updated mesh for each model iteration when the geometry has been changed fundamentally, such as, a change in the number of surfaces or a sudden change in slope.

One method meshing software employs to handle slight modifications to the model is by morphing the mesh along a given edge. As the edge length changes the nodes along that side shift slightly. If the geometry does not change, or only slight dimensional

changes occur, then the mesh remains intact by stretching or compressing the mesh, which creates potential stability and accuracy problems, and does not account for any feature or subassembly modifications. Any change in the number of faces within the model is not supported by morphing methods. Even the addition of a chamfer to an edge will require a complete re-mesh.

When remeshing is necessary, batch meshing is employed. With batch meshing each iteration creates a new file that attempts to mesh the model with a set of parameters that define the mesh creation. The meshing software used for this thesis does not have the ability to mesh assemblies, or to apply boundary layers, in batch. A program must be written that will create the mesh with appropriate boundary conditions and correct mesh interfaces between parts.

The final method, the one used in this thesis, utilizes a programmatic approach. This approach helps to overcome all of the obstacles faced by the morphing and batch methods. PODS uses program that accomplishes the following 5 tasks:

1. Import the model from NX.
2. Mesh each fluid domain.
3. Create identical interface meshes.
4. Prepare faces so the solver can import them automatically.
5. Export the model in a format that is usable by Fluent.

With the completion of this program, the meshing software can be used effectively in an optimization routine. This provides PODS with a parametric method of meshing the model. During each iteration, the program creates a new mesh for the new

model. Then there isn't any difficulty dealing with changes in the number of features or components.

3.4 Fluid Analysis

Fluent has the capability to analyze a fluid assembly by the use of a sliding mesh. For each time step, the model either translates or rotates the meshes at an interface and recalculates the interactions of the shifted meshes. The fluid analysis can handle sliding meshes at interface boundaries with very little set up when done interactively; however, with iterative methods, a programmatic approach works best.

Fluent has several methods for providing commands. The first is through the GUI, which is ineffective for an optimization routine. The second involves recording a journal file. The journal file is a text file of commands for Fluent. This requires the user to start a recording and follow all of the steps required by the GUI to run an analysis and report on the result. The third method uses the Text User Interface (TUI). The TUI uses text commands for all processes that are very similar to the GUI controls. The final method uses a journal file that is written using the TUI commands. The TUI is the simplest of the methods to manipulate programmatically.

The updating of a journal file requires that the correct attributes be passed from the modeling program, the mesher and the database. Another program was written to ensure that all of the settings in Fluent are correct. This component of PODS handles all feature and component changes by re-writing the journal file with the correct commands. Outputs are then exported to a database for use in further iterations.

3.5 Optimizing the Solution

The optimization can be handled in a couple different ways. A program could be written to run specific algorithms or a commercial package like iSIGHT-FD or Altair HyperStudy could be used. The advantage of writing a program is the speed at which a specific program can run. For an application to be the most efficient, a program that is specifically built for this purpose is the best. However, the programming of any optimization routine takes time and money. Each algorithm that is added makes the program that much more complicated. The efficiency of the program has to be weighed against the time it takes to create the program.

In most cases, the commercial programs are more than sufficient. In the case of a jet engine, so many variables need to be considered, several different optimization techniques will be required to find a solution. The processing time lost is negligible when compared to the routines run by the meshing and solving software. The key is to have a program that can access all of the components freely.

iSIGHT-FD is built as a multidisciplinary design tool with the ability to store results in a database. iSIGHT-FD comes with built-in functionality to interact with several commercial programs and for those that iSIGHT-FD does not, a routine is supplied to run external code through command line methods. The method for running external code works for both commercial and custom code. iSIGHT-FD can also be set up to store the results of any run automatically when setup with a MySQL database. The data is then accessible through iSIGHT-FD interactively. This thesis relies heavily on iSIGHT-FD to setup the workflow and run the optimization routines. Additionally, iSIGHT-FD's built-in MySQL storage is used for the results of each iteration.

4 Development

The following chapter of this thesis will address how PODS was implemented and what difficulties were overcome during this development process. Five specific commercial packages were integrated to create a comprehensive design tool, which combines CAD, CAE and expert design rules, with optimization to automate major segments of product development. MySQL was selected for the database; the modeling program is Siemans NX4; for the meshing program Altair HyperMesh was used; Analysis and post-processing was done using Fluent and finally, the design optimization was completed using iSIGHT-FD. The above programs were selected for their ease of use and availability in industry.

To interface with these packages, five custom applications were programmed which comprise PODS. Each subprogram accesses the database and updates a corresponding design tool. The External and Internal NX (ENX and INX) programs both interface with NX4; the TCL script creates the HyperMesh model; the Fluent Journal Writer (FJL) program creates the journal files for Fluent and finally, the DBInsert program inserts the results and next iteration parameters from iSIGHT-FD into the database. Figure 4-1 shows the core of the pods program and all of the program interactions. This fully developed tool marks the achievement of a comprehensive system for parametric optimized design.

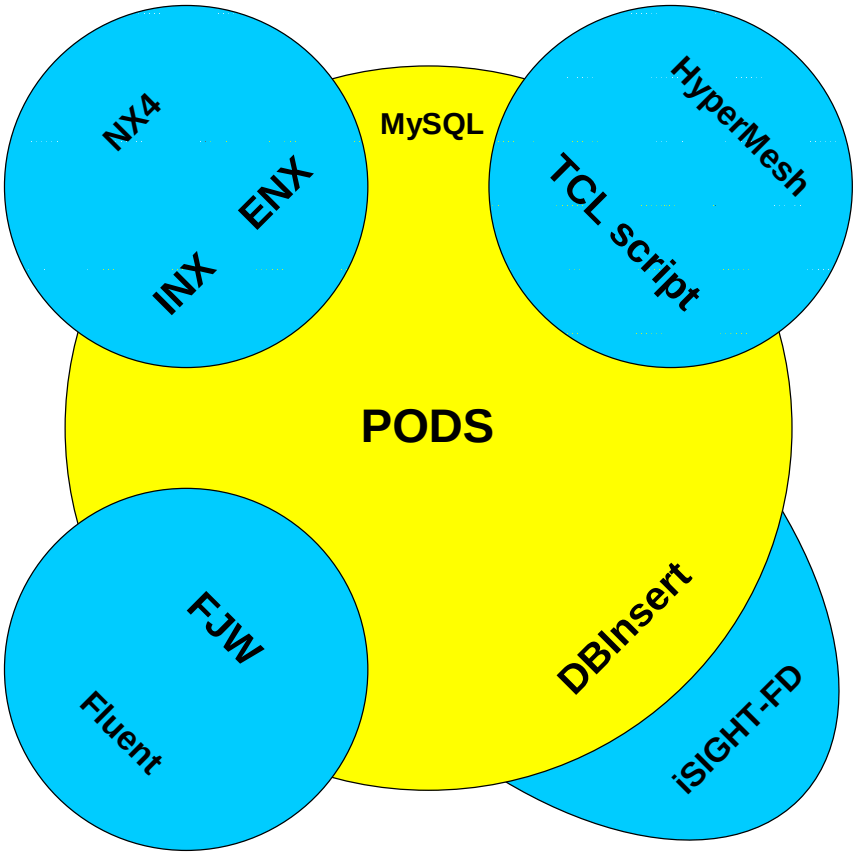


Figure 4-1 PODS program interaction chart

4.1 Database

The heart of this comprehensive design tool is a common database, which stores all the design parameters and rules and may be accessed by any commercial package used in the design process. There are many database programs that are used throughout industry. MySQL was selected, because it is a free program and has a simple to use C language API (Application Programming Interface) for accessing the stored data. Because of the way NX imports and exports expressions, the flat model was used. The full potential of using a relational database was not achieved in this thesis as Chapter 3.1

discussed. The initial design of this thesis included the concept of storing all of the named expressions in the database, so optimization could be performed on any expression or component. With all named expressions imported to the database, the program has the versatility to work with any model. PODS used several parameter tables to cover the complete range of parameters required by the test case model. The DB also included the necessary data for ordering the assembly in a specific order so that mesh interfaces could later be assigned.

iSIGHT-FD was chosen to run the DOE and optimization routines, however, it has very limited DB capabilities. The beauty of iSIGHT-FD is its ability to run any external applications. To overcome the DB limitations, a utility program was written to insert new entries into the parametric table. This program, called the DBInsert program, was very easy to write in C#. Besides the DBConnect function, which was already developed for some of the later programs, all that was needed was the following code.

```
public Insert(string[] args)
{
      foreach(string a in args)
            InsertInto = InsertInto+" "+a;
      DBConnect();
      con.Open();
      MySQLCommand cmd = new MySQLCommand(InsertInto,con);
      cmd.ExecuteNonQuery();
      con.Close();
}
```

With DBInsert, an SQL statement can be executed. The input for the program is the command line args. All that needs to be done is to call the program and tack on the SQL statement in the command line. The DB was built with eight separate tables to hold all of the parameters for the model. The total number of parameters that could have been used to optimize the test case model came to 76. The flat model DB proved to be the simplest structure to incorporate into the model, although not the most efficient.

4.2 Model

There are three main hurdles that need to be overcome to complete the modeling portion of this process. 1) The model must be built in a way that allows immediate update. When any design parameter is updated, then the model must also be updated. Updates require that the model be both parametric and associative. 2) A utility program is also required to interface with the database in order to update the expressions within NX4 and assign boundary conditions to each face. 3) During the iteration process the model must be updated according to the new values that were found during that iteration. With the above three requirements satisfied, the model is ready for meshing, solving and optimizing.

4.2.1 Blade Parametrics and Assembly Associativity

The full model begins with the parameterization of the airfoil at the inner and outer edges of each blade. An axial compressor of a jet turbine engine is designed to compress the air in stages, preparing the air for more efficient combustion in the combustor. Each stage includes a rotor and a stator. The rotor consists of a row of blades attached to the hub of a motor, which rotates in order to compress the air, while the stator's blades are attached to the case of the motor, with the intent to redirect the flow of air back into an axial direction. Creating a parametric compressor can be difficult, since the shape of each stage's blades is distinct from beginning-to-end and for each rotor and stator. The following process explains how to create airfoils in NX4 such that, even without prescribed limits, the airfoil model does not become unstable for the solid blade creation. It is possible with this method to have some undesirable surfaces in the final

design, and is solved by applying appropriate limits to the model during creation, or later, during the optimization routines. Without these limits, the optimization routine will result in failed runs, mainly due to surface contact between blades.

The airfoil creation process presented in the following section is different from, and achieves a level of parametric control way beyond, what is currently practiced in industry. The initial shape of the stator or rotor section is controlled by a quadrilateral sketch as seen in Figure 4-2. Each of the four corner points are labeled and stored in the DB for modification. The points are controlled in the database to ensure that one section links up with the next. If the trailing corner points of the rotor do not match up with the stator leading corner points then the meshing routines will fail. Figure 4-2 creates the outer surfaces of the fluid domain. Revolving the section and subtracting the blades from this domain will complete one portion of the total fluid domain assembly.

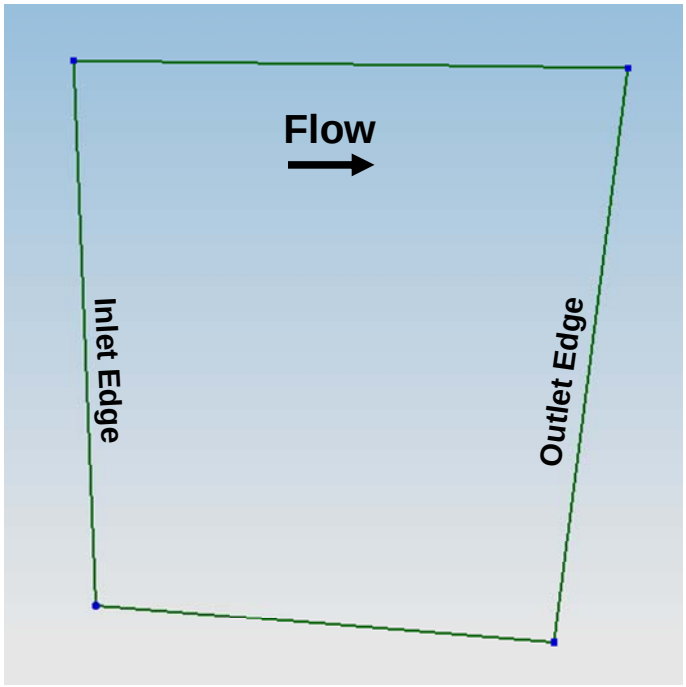


Figure 4-2 Section corner points

The airfoil design begins with a sketch which will provide the inputs for the beginning and ending curvatures of the backbone spline or camber line of the airfoil, as well as the leading edge angle, trailing edge angle, and the rotation angles. The angle control sketch shown in Figure 4-3 shows the control features that constrain the airfoil angles and also controls the gap before and after each set of blades, using the gap# dimensions. The green dashed line defines the sketch to which the camber line will be constrained with the dimensions shown in blue. The angle control sketch is constrained to the section corner points sketch (Figure 4-2) as seen as the blue central horizontal line. The angle control sketch provides the ability to manipulate all of the airfoil parameters except the offset distances of the upper and lower edges.

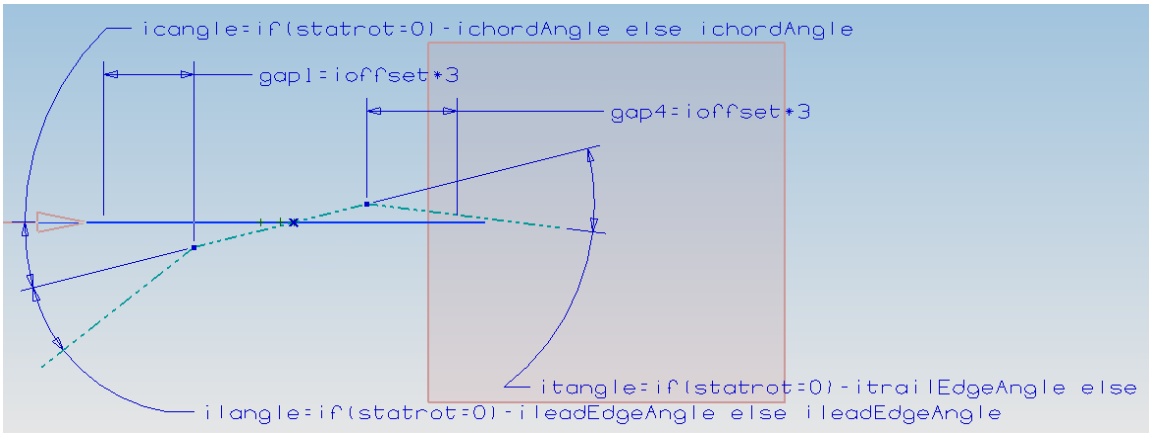


Figure 4-3 Angle control sketch

With the completion of the angle control sketch the backbone spline can be created. The simplest method in NX4 to create the backbone spline is to use a Bridgecurve that connects the first and last lines of the angle control sketch to as in Figure 4-4. In this case the Bridgecurve was constrained with both tangency and equal

slope at the end points. Using Bridgecurves creates a spline that is completely controlled by the angle control sketch parameters.

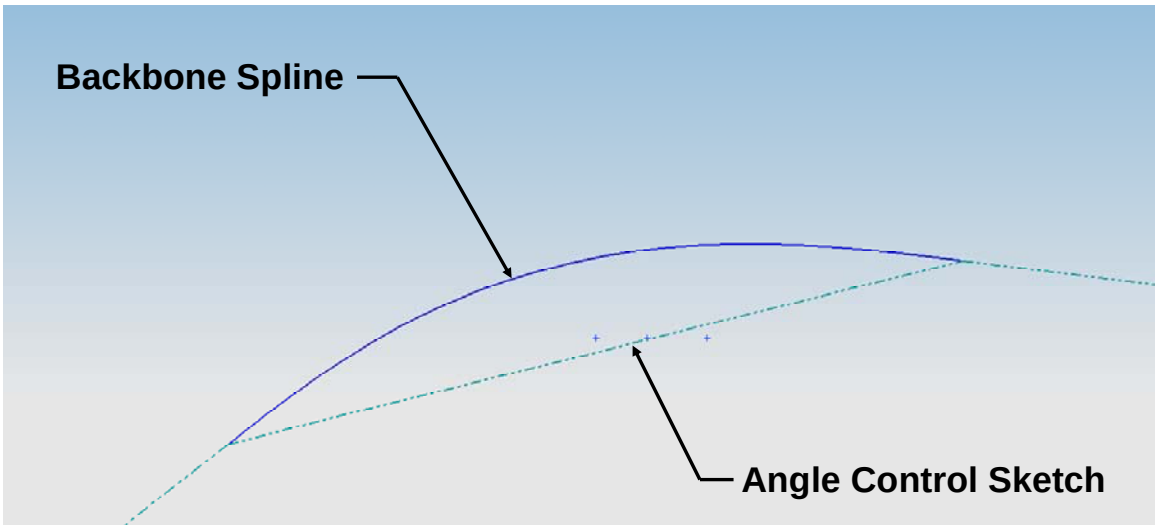


Figure 4-4 Backbone spline

Each blade is built using multiple parametric equations which control the airfoils at the hub and tip of each blade array. The parametric equations are modified through the manipulation of the equation constants, and are used when creating the Law Curves that define the offset from the backbone. Integrating the use of Law Curves and parametric equations is accomplished by defining the Law Curve with parametric equations in the expressions dialog. The Law Curves are defined using Bernstein polynomials as seen in Equation 4-1 and Equation 4-2. The constants a_i and b_i are the x and y coordinates for the control point i. The distance t is the fraction of the total distance along the curve. These two equations define the x and y location of each point i along the Law curve. Four distinct Law curves are defined for the top and bottom offsets for both the tip and hub of the airfoil.

$$x_i = a_0^i*(1-t)^3 + 3*a_1^i*t*(1-t)^2 + 3*a_2^i*t^2*(1-t) + a_3^i*t^3 \quad \textbf{(4-1)}$$

$$y_i = b_0^i*(1-t)^3 + 3*b_1^i*t*(1-t)^2 + 3*b_2^i*t^2*(1-t) + b_3^i*t^3 \quad \textbf{(4-2)}$$

One of the problems with curves in general is that they can loop on themselves. When a line loops that is being referenced by a surface, the surface will fail. To overcome this, a combination of Law Curves and Offset Curves is required. Because of the way that the Law Control works, the lines need to be defined such that the x-coordinate distance from each Law Curve will define the distance along the Bridgecurve (Equation 4-1) with the corresponding y-coordinate defining the offset distance (Equation 4-2). Additional y-equations are used to define the top and bottom offsets of the airfoil at both the tip and hub of the blade. Figure 4-5 shows the top and bottom offset curves with the backbone spline still controlling the lead, trail and chord angles.

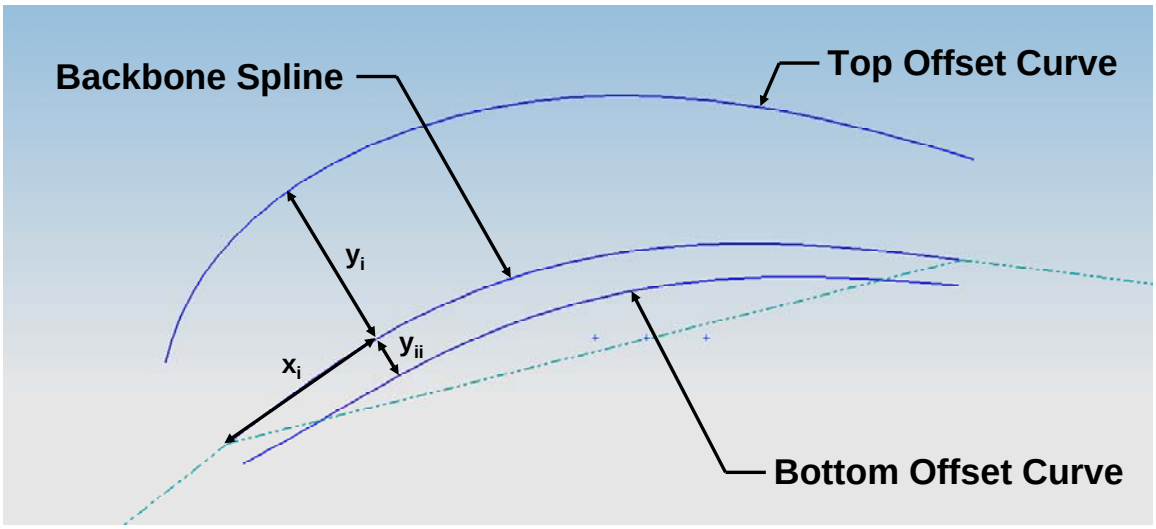


Figure 4-5 Backbone spline with offset curves

The advantage of this method is that the Offset Curves function never loops. Even if the backbone curve loops, the offset curves will simply have a kink. Limits can be added to prevent this scenario. Use of parametric equations, Law Curves and Offset Curves creates an unbreakable airfoil, but the resultant curve is not always desirable. A surface will always be created and errors will never occur during the model update stage. Within the Law Curve, each of the variables is controllable from the database. The top and bottom edges of the airfoil can be created by offsetting the curve with law control using the equations already defined in the expressions dialog.

The final step for creating the airfoil is perhaps the easiest. Closing the airfoil into a closed loop involves simply joining the starting points of the offset curves together with a Bridgecurve and repeating this process with the end points. The completed airfoil, as shown in Figure 4-6, is fully parametric and easily controllable through the expressions dialog.

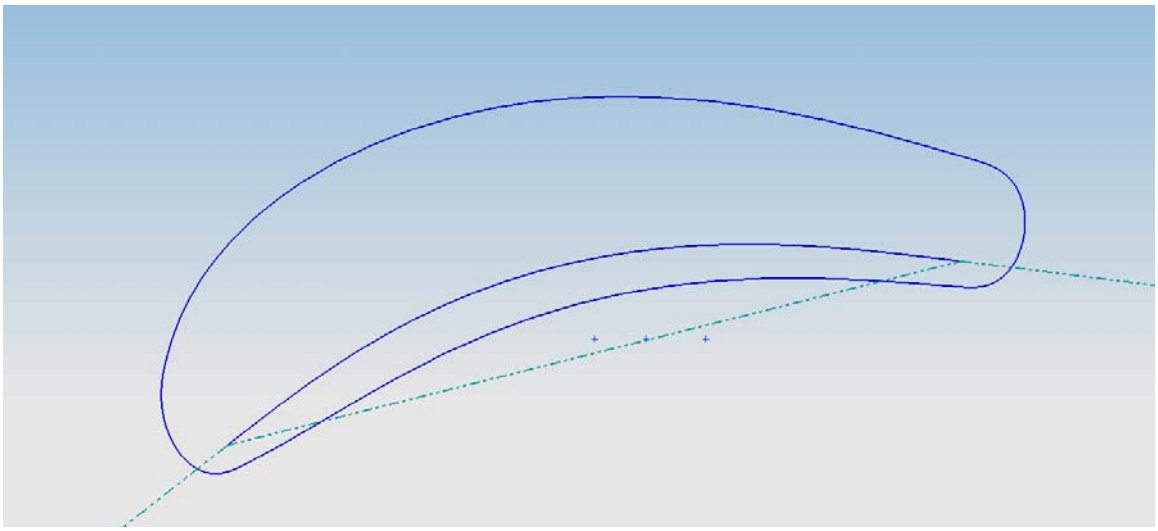


Figure 4-6 Fully constrained airfoil

By duplicating the airfoil creation process at the top and bottom of the flow path sketch and then connecting the sketches with a surface mesh, a blade is formed. Creating the solid geometry is accomplished in NX4 by using a Through Curves Mesh. With two independently controlled airfoils, at the top and bottom of the blade, any twist and attack angle can be achieved. Intermediate airfoils could be introduced to achieve even greater control of the blade. The blade needs to be subtracted and patterned within a 3D fluid domain that is constrained to the section corner points and revolved around the appropriate axis.

The parametrics of the blade instances are handled automatically by NX. The Instance Feature command creates expressions automatically that allow manipulation of the number of features, however they do not automatically transfer non-geometric attributes. The model is now complete for use as either a stator or rotor with the insertion of a logic statement within the expressions dialog that controls the direction of the offsets and the angles used.

With multiple domains, face contact is important to maintain from part-to-part, and becomes simpler using database storage. If all variables are stored with the database, then each part can be updated and the concern for top-down or bottom-up construction becomes irrelevant. All edge positions are controlled within the database.

4.2.2 Face Extraction and Attribute Assignment

HyperMesh has an import function for NX4, which allows for simple conversion into the meshing environment. Within the import command are controls that allow the imported model to be divided into what HM calls Collectors. Using the HM import

routine, boundary conditions become a little simpler to pass from one NX4 to HM. The difficulty lies in the options that are available for the HM import command.

One drawback of HyperMesh import options is that they are mostly part related i.e. part name, part number part tag or part instance. The simplest option that allows manipulation within a part is the layer import field. None of the import functions allow for individual face assignment into different Collectors unless each face is a distinct part. Another drawback of HM import is found in the NX4 options. Moving features from one layer to another is a simple process, but requires each face to be modeled distinctly, rather than as a solid. Modeling in this way would be inefficient; therefore, a simpler method is to write a program that will extract faces from any solid model and assign those faces to a user-assigned layer.

The initial step of the face extraction process includes the creation of a GUI that provides the ability to select any face, or combination of faces, and assign boundary conditions to those faces. The GUI is part of the INX program. The GUI specifics are shown in Figure 4-7 and will be discussed hereafter. After all of the faces have been assigned boundary conditions, each face can be extracted. The GUI populates the list of all of the faces and bodyfeatures within the part. From the list, the faces and bodyfeatures of a particular BC are selected by the user. The bodyfeature selection will highlight all of the faces in a particular array, rather than forcing the user to select all of the faces in an array individually. The next step is to select from the boundary condition dropdown list, which boundary type is to be assigned. Choosing the Apply button assigns the BC attribute and applies a corresponding color to the face. The color is for ease of use for the operator, so that they know which faces have had BCs assigned to them. Only five BCs

were programmed into PODS; however, others could be added for either additional fluid BCs or for structural applications. Figure 4-8 shows the BC dropdown selections. The colors that correspond with those BCs are red, orange, brown, light blue and blue.

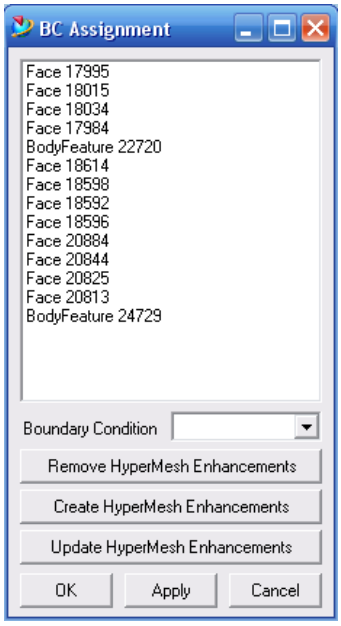


Figure 4-7 BC Assignment GUI

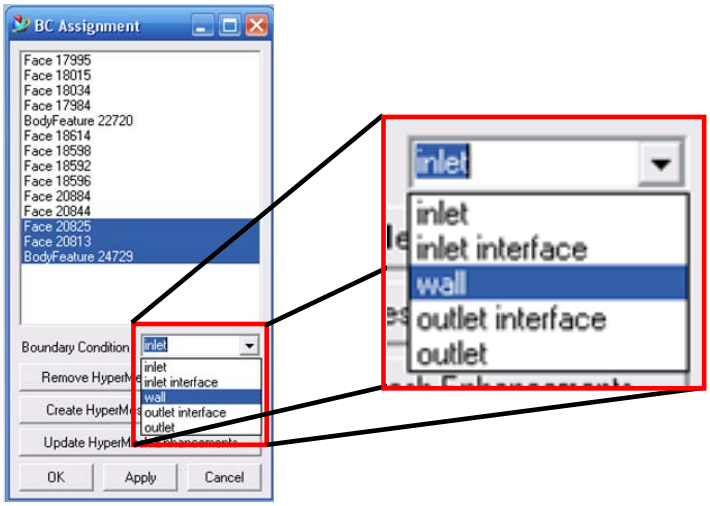


Figure 4-8 BC dropdown menu

Near the bottom of the GUI are three unique buttons, Remove, Create and Update HyperMesh Enhancements as seen in Figure 4-7. By hitting the second button, Create HyperMesh Enhancements, the INX scans all of a solid's faces to ensure that they all have boundary conditions and then makes copies of those faces. As face copies are made, the boundary condition attributes are also copied to the new face, and the resulting face is moved into an appropriate layer for HyperMesh. The layered face copies are the key to utilizing the Hypermesh import function. None of the BC data will actually be passed from the CAD package to the mesher. The Create HyperMesh Enhancements function creates importable data with appropriate non-geometric data for downstream processes in HyperMesh, Fluent and iterations.

If at any point the user wants to return to the pristine state, the first button can be clicked and all enhancements will be removed. The final button, Update HyperMesh Enhancements, accesses the DB and updates the model with all of the latest values entered.

4.2.3 Iterative Models

The steps for iterative models are primarily the same. To run NX and update the model in batch, the External NX (ENX) program was written. The ENX is an executable file, while the INX is a library for use within NX4. The ENX runs without any user interactions, but accomplishes the same function that the Update HyperMesh Enhancements button performed within the INX program. The file performs all of the steps of the following pseudocode:

```
Open the model or assembly
For each subassembly
      Erase all existing face copies
```

```
      If all faces have BC attribute
            Continue
      Else
            Fail
Access the DB
For each subassembly
      Extract the parameters from the DB and update the
      subassembly
      For each array
      For each instance of the array
      Find the parent feature of the array
            Copy the parent features attributes
      Ensure that all faces have attributes
      Copy all faces into designated layer
Save model or assembly
If any of the above steps fails
      Notify the user
```

One of the difficulties to be overcome for INX and ENX was how to handle the instances within a part. The problem was twofold; 1) How to list all of the faces of each part, and 2) How does the program handle changes in the number of instances? The listing of every face in the INX GUI and having to select each one became tedious with multiple parts and many instances. INX was written to include instances within a part as one group. In this way, individual faces can not be assigned different boundary conditions. A limitation of the program is that multiple instance arrays are not addressed. If the user changes the number of instances in a part after creating the HyperMesh enhancements, the extracted faces will not update. The copies are not associated with the face from which they were extracted, which means that after changing to a parts instance array, the model requires an update within INX or ENX. INX and ENX will delete all of the extracted faces, as the first step, and then ensure that all faces within the array have been assigned boundary conditions. Boundary conditions are reassigned by inspecting the base feature of the array, and then copying the attributes to the rest. The final step is to re-extract the faces.

4.3 Mesh Generation

HyperMesh comes with a batch mesher, however, it does not address multiple domains or assignment of mesh interfaces, therefore, this process was developed independently using Tool Command Language (TCL). TCL is a scripting language that is easy to learn and meant to interface with manufacturing tools without needing to learn C or other high level programming languages. HyperMesh has developed their programming interface around this language. Because of the nature of dealing with different numbers of parts, the importance of meshing each part domain distinctly becomes apparent, but the interfaces between domains must have identical meshes for Fluent. Fluent requires all of the domains be included as a single file when importing in the Nastran file format.

TCL has a command to run conditional loops, called "*for loops*", which allows this process to work. The following pseudo code demonstrates the process that was followed:

```
Import the UG surfaces
Open the DB connection
Mesh all the surfaces with the size from the DB
For each interface
      Access the subassembly order from the DB
      Copy the outlet face to inlet face
For each subassembly
      Equivalence nodes
Create a Fluid Collector for an automatic zone in Fluent
Tetramesh the volume
Rename the Face Collectors for automatic zones in Fluent
```

The process starts by importing the NX4 model using the layer and part name. Through the import process, each face is assigned to a Collector that will be renamed to reflect the boundary condition that Fluent needs. Table 4-1 has some examples of the component names after they are imported from NX. The mesh is created using the grid

size the DB, but currently is limited by the Equivalence Nodes function. Equivalencing (i.e. combining all nodes within a given tolerance into a single node) is necessary before a volume mesh can be created. Equivalencing removes unwanted duplicate nodes, which occur when adjacent volumes are meshed. After equivalencing the mesh, HyperMesh can fill the volume with a tetramesh.

Table 4-1 Examples of NX imported names and Fluent zone export names

NX Import	Fluent Export
101_rotor.prt	inflowrotor.prt
102_stator.prt	inflowstator.prt
103_stator.prt	wallstator.prt
104_rotor.prt	outflowrotor.prt
105_duct.prt	outflowduct.prt

Many of the HyperMesh macro commands are written in TCL script, including an export command for Nastran files. If the Collectors within HyperMesh are named appropriately, Fluent can import those Collectors directly into zones. When the file is imported into Fluent, all of the conditions are easer to assign.

4.4 Fluid Analysis

As has previously been discussed, Fluent was chosen as the CFD analysis application for this thesis. There are two distinct aspects with regard to how Fluent handles the data that is passed to it. The first is handled through the Nastran import that is built into Fluent and the second is handled through an additional program, the Fluent Journal Writer (FJW), which writes a journal file for Fluent to run in batch.

The Nastran file is a text file, which includes all the data necessary to import a model into Nastran. In Fluent, all of the elements are organized into zones which include the element condition assignments. Different zones are declared for each boundary condition and fluid volume. Fluent has designed one of their import functions to read Nastran files and sort the elements in them into the appropriate zones, if the Nastran file was initially exported correctly. The import/export process was successfully implemented.

The next step was a program to write or rewrite the journal file for actual execution. The journal file in Fluent is just a list of commands. To create a journal file that was updatable, depending on the number of subassemblies and interfaces, required an additional program. The Fluent Journal Writer (FJW) program handles several steps.

1. Deletes previous iteration output files.
2. Imports DB inputs for specific boundary condition values.
3. Assigns other standard BC values automatically (atmospheric conditions).
4. Determines which BCs should be interfaces.
5. Writes where and what data should be output after the run.

A focus of the FJW was to create a process that any and all of the BCs could be assigned through the DB. For the purposes of the current model, the only BC input used from the DB was the RPM of the rotor fluid zone. The current version of FJW is only set up to run a first-order unsteady problem with the specific temperatures and pressures for the current single stage model. Programming knowledge is required to modify these inputs; otherwise, the program accesses the database to ensure the proper order for the interface zones, as well as reassigning the existing zones to those interfaces. These two

features achieve the primary goal of being able to run analyses of an assembly of fluid domains. Future FJW versions should also include other conditions necessary to setup various Fluent models.

As a reminder, HyperMesh renamed all of the zones to be inlet, outlet or wall zones. Perhaps other zone names work as well, but none were attempted for this thesis. Knowing the proper order makes it a simple loop to write the journal file to change inlet and outlet zones to interfaces and define those interfaces. The final step that the FJW accomplishes is the export of any reported data. The FJW writes into the journal file a few lines telling Fluent what values to report. Again, required fields could be declared through the DB, but in this case were hard coded into the FJW.

4.5 Optimization Routines

The setup in iSIGHT-FD consists mainly of providing a workflow and method for accessing each program. iSIGHT-FD has a simple graphical user interface, which is extremely flexible with several available routines. The two design routines that were utilized for this thesis consisted of Design of Experiments (DOE) and Optimization methods.

iSIGHT-FD has limited database connectivity. With time this may improve but the current capability is limited to queries, plus reading and writing to a specific row, column or entry in the database. The main method that was needed for this thesis was the ability to insert new data into a table in the database; so, DBInsert was written. DBInsert is the final component of PODS and provides the ability for the data to flow from iSIGHT-FD to the database. The iSIGHT-FD workflow is built with all of the model's

focus parameters and all constraints for those parameters. Each iteration modifies those parameters, and the database is the means by which this information is passed to the other four PODS programs. The data is then accessible to all of the other programs for that design iteration.

DBInsert completes the development of the five programs that comprise PODS. Table 4-2 shows each of the programs and the functions that each program completes. A comprehensive system for parametric optimization was conceived, designed and implemented in code. A sample gas turbine design was chosen for a demonstration.

Table 4-2 PODS programs and their subroutines

Program	Subroutines
INX	Assigns BCs, sets up model for HyperMesh, updates model
ENX	Updates model, reassigns BCs, sets up model for HyperMesh
TCL Script	Imports model, creates mesh, assigns elements into zones, saves Nastran file
FJW	Creates journal file, imports Nastran file, sets up interfaces, outputs results
DBInsert	Stores data in database

5 Discussion of Results

The case study for this thesis consisted of the design of a rotor/stator/duct combination for the final stage of the high pressure compressor of a jet turbine engine. Individual applications were built to store and retrieve data from MySQL and others were built to transfer data to and from NX4, HyperMesh and Fluent. The command file for HyperMesh was written using TCL script and the journal file for Fluent was recreated using FJW. For all NX4 interactions the INX and ENX programs were used. All of these programs and files were accessed through iSIGHT-FD for the optimization portion of this work. The desktop computer, on which all of this research was completed, consisted of an Intel® Core™ 2 Duo 2.66 GHz processor with 1.6 GHz, 1.98 GB of RAM. Finally, each of these individual programs was integrated into PODS as a fully automated system.

The results utilized both DOE and Optimization runs in order to test the robustness of PODS. This study is not intended to show the optimal configuration for a jet turbine, but to prove the concept of parametric optimization. The following results are not intended to be a fully accurate fluid analysis of any existing motor or design, but to show that the process was able to optimize a complex assembly with feature instances and multiple domains. Other variables need to be included in the study, as well as corrections to the meshing algorithms, in order to achieve a complete and correct

solution. Additionally the optimization routines need to run until an optimum is reached using a much finer mesh.

5.1 Database Access Performance

After the initial setup for the CAD model, database access was easily performed by the PODS programs. The following sections in Chapter 5 will address the specifics with regard to each PODS program (INX, ENX, TCL script, FJW, and DBInsert). As an overall summary, every iteration is stored in the database tables for history and future retrieval if desired. Some additional information, which is only accessed within the necessary programs, is also stored in the database (i.e. mesh size and interface order). iSIGHT-FD also stored all of the results in the MySQL database. All program access with the database performed nominally.

5.2 Modeling Parameter Updates

PODS was built with the ability to modify or optimize any desired parameter for any assembly. For the case study of a gas turbine engine stage, fourteen factors and two objectives were chosen for the jet engine to test PODS. The parameters chosen are shown in Table 5-1 and the objectives are shown in Table 5-2. The bounds were chosen to prevent blade surface interference only. For clarification, the chosen factors can be seen in the angle control sketch in Figure 4-3. The chord angle (also referred to as a stagger angle) is used to control the angle of twist for the airfoil, while the leading and trailing edge angles control the shape of the camber line relative to the chord angle. The pressure ratios were chosen as the objective for Fluent to output.

Table 5-1 Focus parameters as factors

Parameter		Lower Bound	Value	Upper Bound
Rotor	Blade Count	10	25	30
	Internal Chord Angle	-20	-15	-10
	Internal Leading Edge Angle	10	30	30
	Internal Trailing Edge Angle	10	20	25
	Outer Chord Angle	-20	-15	-10
	Outer Leading Edge Angle	10	30	30
	Outer Trailing Edge Angle	10	20	25
Stator	Blade Count	10	25	30
	Internal Chord Angle	-20	-15	-10
	Internal Leading Edge Angle	10	30	30
	Internal Trailing Edge Angle	10	20	25
	Outer Chord Angle	-20	-15	-10
	Outer Leading Edge Angle	10	30	30
	Outer Trailing Edge Angle	10	20	25

Table 5-2 Objectives for study

POM Objectives (Maximized)
Rotor Total Pressure Ratio
Stage Total Pressure Ratio

The INX and ENX programs were shown to be fully capable of extracting the necessary parameters from the database and updating the interactive CAD model to a new instantiation. With each iteration in iSIGHT-FD, the database is updated and then the NX model updated through ENX. Figure 5-1 shows two different iterations, proving how well ENX performed. The update from the left-to-right of Figure 5-1 includes subtle changes to most of the fourteen factors, while the change to blade count is quite obvious. The only constraints imposed on the model were to ensure that the model and mesh were still usable; otherwise, the model was allowed to change with very broad limits. The idea

behind this method of testing was to see if the optimization routines would result in a feasible design when restrictions were not applied. The most apparent change is in the number of turbine blades used. The solidity of a stator or rotor is the ratio of the chord length to blade spacing. In general, the solidity should be approximately 1.0 or greater. The focus of the stages in Figure 5-1 is to show the flexibility of the CAD model, not to show a realistic solidity. Solidity constraints are another example of knowledge that can be stored and retrieved from the database.

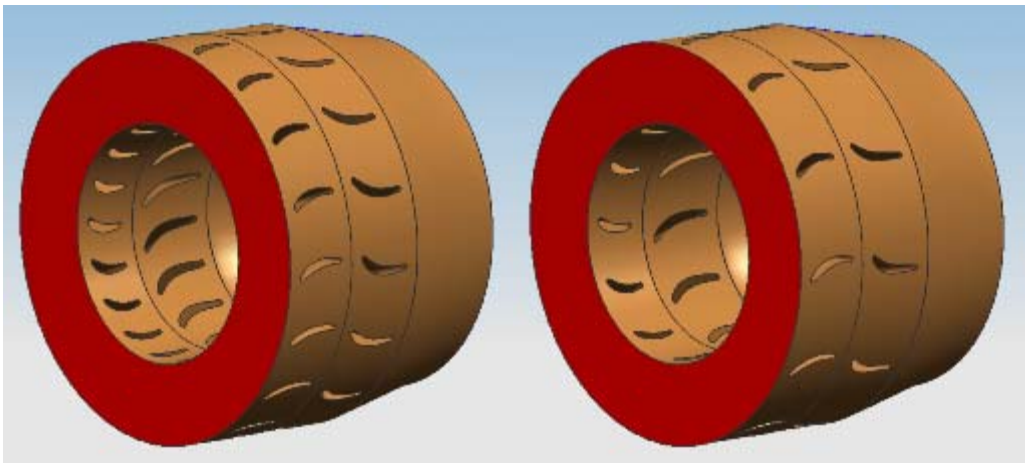


Figure 5-1 Distinct iterations created using INX

5.3 Mesh Creation and Study

The inputs for mesh creation proved to be fairly delicate. The size of the mesh affected how well the domain could be optimized, especially around the blades. Tetrahedral elements were used for the volume mesh, which required the surface meshes

to be triangular. Poor mesh quality resulted in the mesh not keeping very close to the overall shape of the blade.

A TCL script was written to create the mesh for all of the faces with a standard size. In order to ensure that Fluent can use the interface meshes, both sides of an interface must have identical meshes. To force identical nodes, the mesh is deleted from the inlet face of the downstream domain and then the mesh on the exiting face of the upstream domain is copied to the inlet face. Copying the mesh from one face to another requires the edges of the volume mesh be equivalenced (duplicate nodes are combined) in order to create the internal tetrahedral mesh. For the test model, the range of possible mesh sizes was limited to about 0.3 to 0.5 inch by the equivalence function, and used an equation for interpolating the tolerance for equivalencing any two points.

$$t = 0.3 - 0.5 * m \qquad \textbf{(5-1)}$$

Equation 5-1 uses the mesh size (m), returns a resultant tolerance (t) for the equivalence function. You can see in Table 5-3 that changing the mesh size by only 0.2 inch changed the mesh by greater than a factor of three. With the 0.5 inch mesh, Figure 5-2 shows the mesh generated in HyperMesh by this process.

Table 5-3 Mesh size comparison

Mesh Size (in)	0.5	0.3
Nodes	17879	65405
Elements	94914	352237
Time/Iteration (min)	3	17

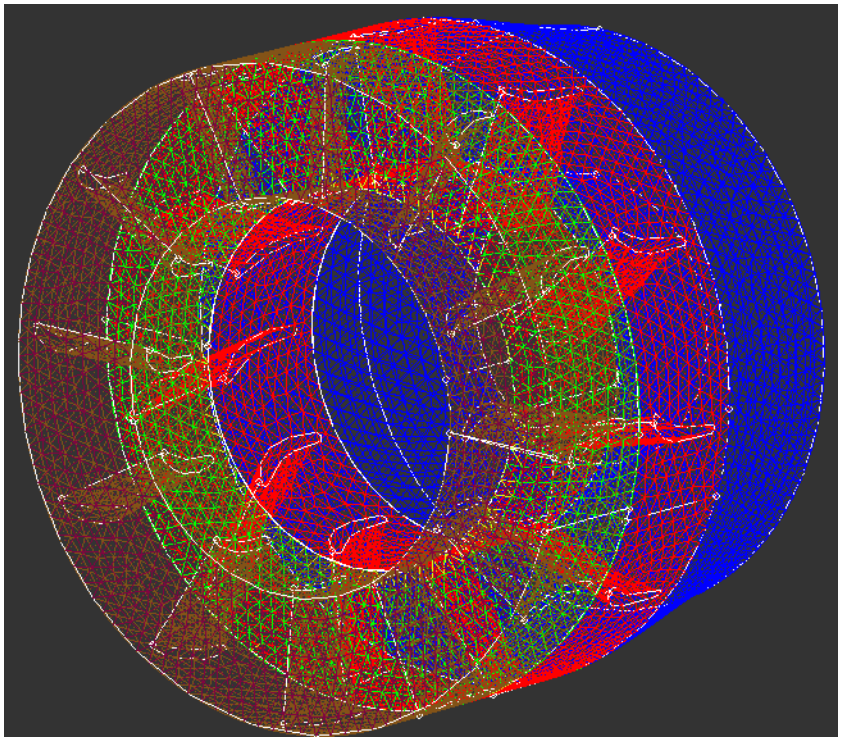


Figure 5-2 Full 0.5 inch mesh

The above mesh creation resulted in the inability at this time to set up boundary layers in an iterative environment. Because of higher shear and fluid viscosity close to any surface that a fluid passes over, a layer of smaller elements, a boundary layer, is used next to the surface. In practice, the next layers above the surface are progressively larger. Boundary layers are more time intensive to set up interactively, but provide more accurate results. When two surfaces require boundary layers that have a shared edge the mesh is computationally more difficult. HyperMesh does not provide a script to automate this process. Without boundary layers, the mesh can not be refined to the level required for a model of this type. To accomplish greater refinement, the HyperMesh script needs to be able to handle boundary layers. Figure 5-3 shows tetrahedral elements with a mesh size of 0.3 inch on a five inch blade results in a very poor leading and trailing surface, as

seen in Figure 5-4 through Figure 5-6. The future work for this condition should divide the blade face, which is currently treated as a single wrapped sheet, into multiple faces in order to obtain a more accurate surface approximation. A finer mesh would also allow a thinner blade to be analyzed. This thesis was focused on proving the design system by enabling the iterative process, not on finding an accurate solution.

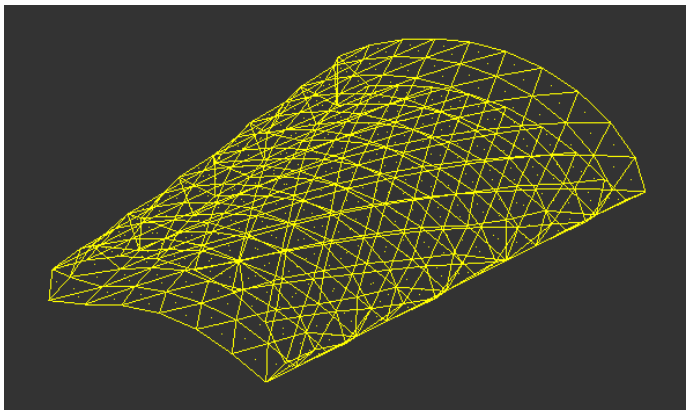


Figure 5-3 Isoview of a meshed blade

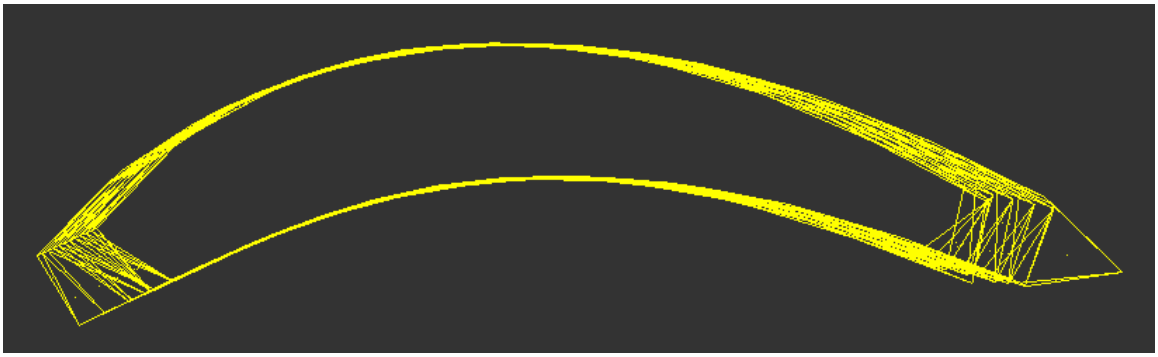


Figure 5-4 Blade profile view

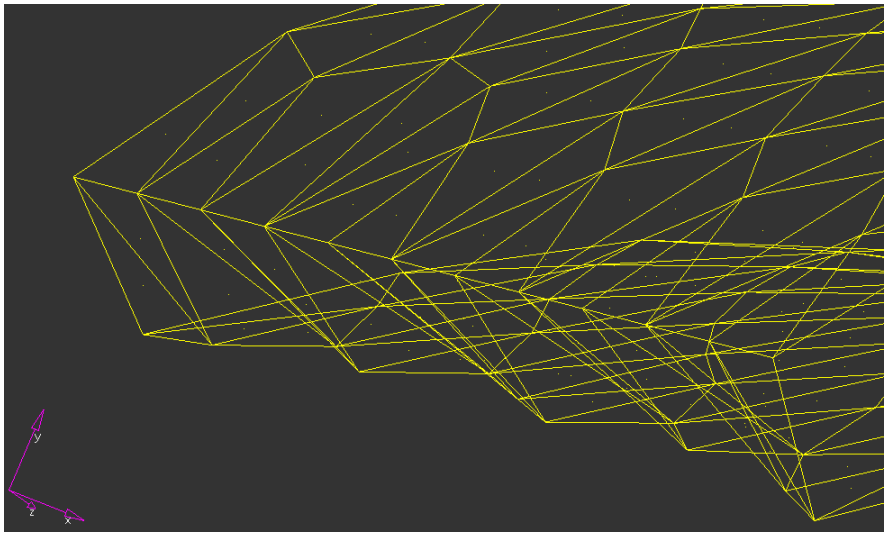


Figure 5-5 Blade leading edge with poor surface approximation

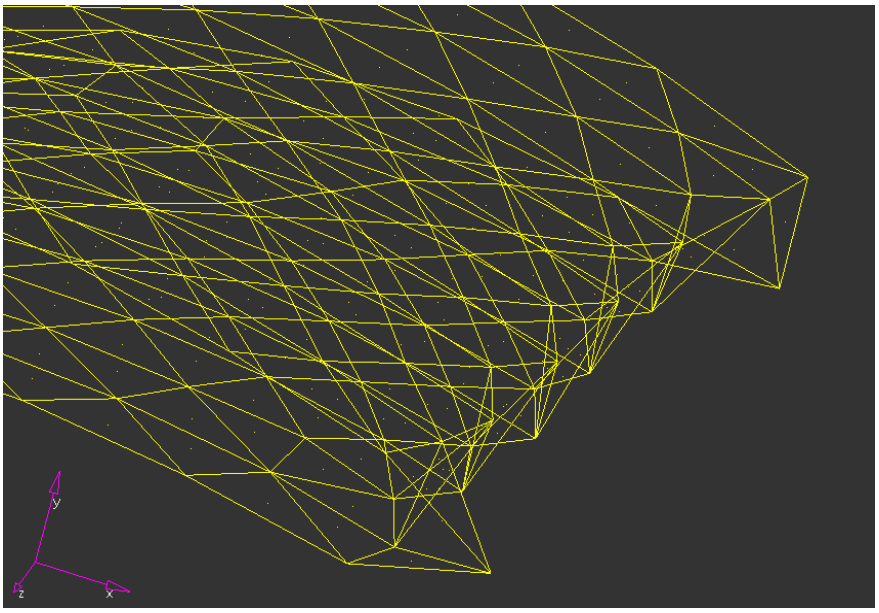


Figure 5-6 Blade trailing edge with poor surface approximation

5.4 Fluent Capabilities

Any number of outputs could have been selected in order to meet a design goal. In order to demonstrate the concept, only two were selected. The total pressure ratio across the rotor and the rotor/stator combination (a single stage) were used as the parameters to optimize. Fluent exported, to a text file, the total pressure of each of the inlet and outlet faces. The outputs varied significantly, depending on what size of mesh was used. Figure 5-7 shows how the results of a single run at an optimum changed with mesh size while Figure 5-8 shows the amount of error in the design when no changes are made to the model. Figure 5-7 shows that as the mesh was refined the stage pressure ratio increased by approximately 30% and is still following a linear pattern of improvement with no sign of tapering off to an horizontal asymptote. Grid independence was not achieved in this design primarily because of the limitations when equivalencing nodes and lack of a boundary layer.

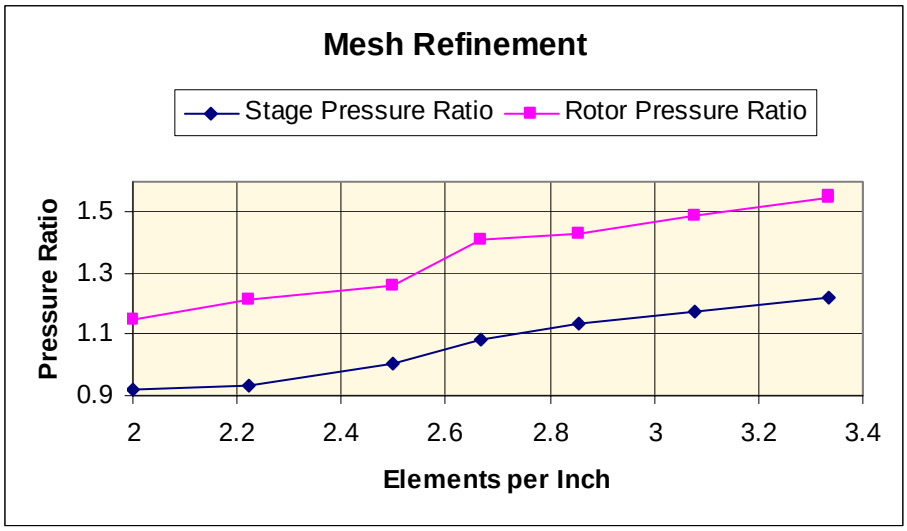


Figure 5-7 Mesh refinement of an optimum

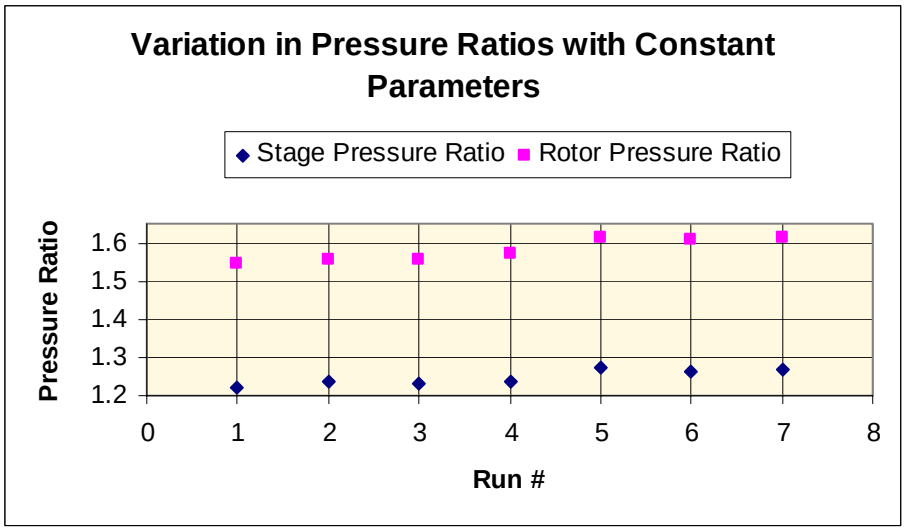


Figure 5-8 Results show that at the optimum value there is a 4% error

The input for the Fluent model was based on a J57 turbojet design, since it was the most similar in size to the test case built in NX4 for this thesis. The data was interpolated from Table B.4 of "Elements of Propulsion" (Mattingly, 2006, 799). Interpolating the data provided comparative data; however, the actual geometry of the model was not verified. The model length was used as a reference and then the inputs were interpolated from the approximate beginning and ending of the J57 high pressure compressor section. Interpolating from the J57 conditions provided approximate values for the last stage of an axial compressor of comparable size.

The last stage of an axial compressor should have a radial velocity component for the stage input but this was not included in order to simplify the model. Table 5-4 shows some of the inputs that were used for the model. Some of the additional conditions that were controlled through the journal file included setting the model as a first order unsteady problem, scaling the grid from metric to English units of measurement and

identifying each of the grid interfaces. None of the preceding conditions were adjusted by PODS, but the interfaces were retrieved from the database.

Table 5-4 Fluent model conditions

Fluent Conditions	Value
Max Rim Speed	1500 ft/s
Airflow	165 lbm/s
Temperature In	594 °F
Pressure In	144.4 psia
Angular Velocity	1956 rad/s

The following figures from Fluent show the best point of a DOE run with a 0.5 inch mesh size. The DOE ran a Latin Hypercube of 20 iterations. The best point of the DOE run resulted in a 12 blade rotor and an 11 blade stator, which is completely inaccurate considering that this produces a solidity (airfoil chord to airfoil spacing ratio) of approximately 0.15. Most stages are designed with a solidity of one or greater. Figure 5-9 shows profiles of total pressure, while Figure 5-10 shows the velocity path lines through the model. The streamlines and pressure profiles show that the setup in Fluent can produce an accurate result for the parameters that were provided for this model. For comparison, the J57 turbojet operates with an overall pressure ratio of 12 with 16 stages, meaning there is approximately a pressure ratio of 1.33 across each stage. Because this model does not function with the actual dimension of a stage in the J57, an analytical pressure ratio was found using the same input data supplied to Fluent. With a stage efficiency of 85 percent, the analytical result is 1.18 using this data. The stage pressure ratio for the best point using the DOE was approximately 1.13.

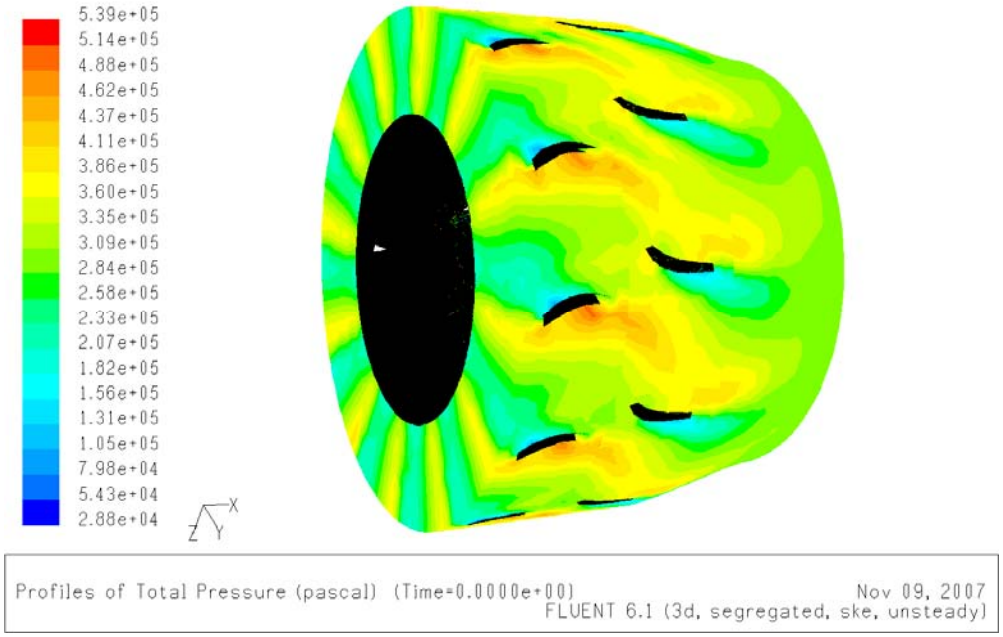


Figure 5-9 Pressure profiles on outer surfaces

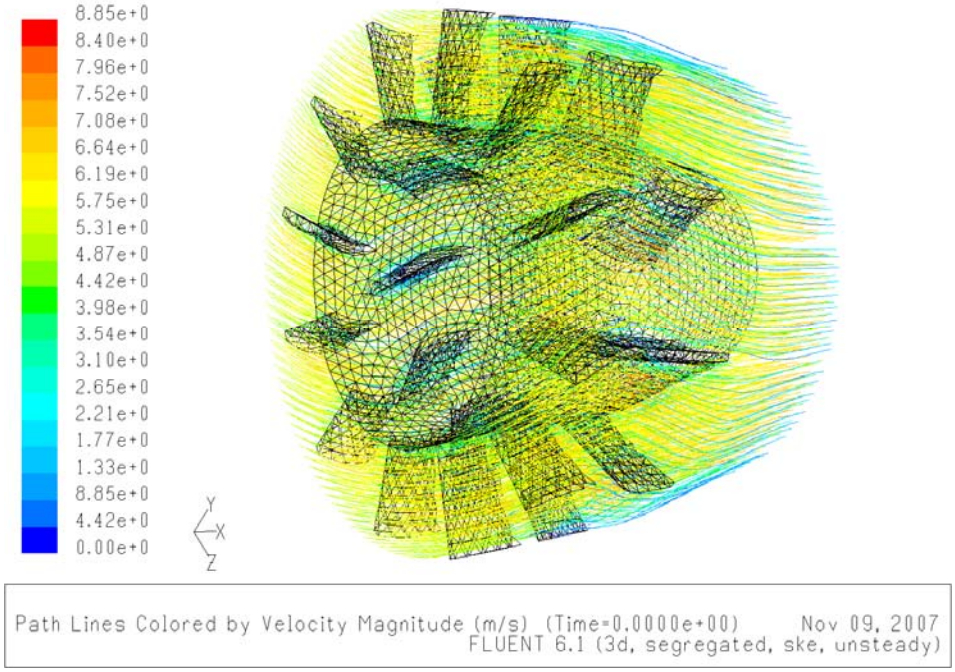


Figure 5-10 Velocity streaamlines

5.5 Design Study

iSIGHT-FD performed as expected with each of the programs that were written for PODS. Several different routines were used to test the robustness. Both the DOE and Optimization runs functioned equally well, for this level of accuracy. The final run of the optimization routine utilized the Pointer optimization method and resulted in a stage pressure ratio of 1.212. The Pointer method utilizes a combination of optimization techniques that includes a genetic algorithm, Nelder and Mead downhill simplex, sequential quadratic programming and a linear solver. The Pointer method was chosen because of its ability to solve a very complex surface knowing that the accuracy of the Fluent results was in question. Figure 5-11 shows the overall configuration. The ENX and FJW codes were directly accessed while the HyperMesh and Fluent were run with the command line calls.

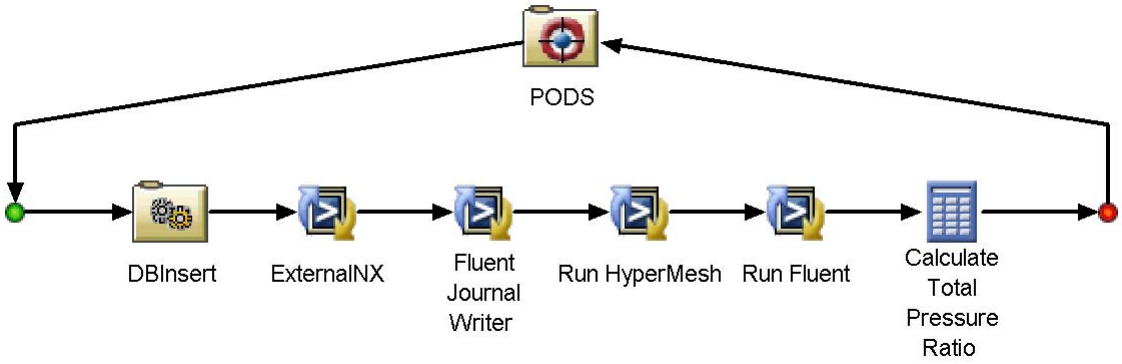


Figure 5-11 PODS workflow in iSIGHT-FD

The DBInsert task was used to encapsulate all of the insert commands. Figure 5-12 shows each table that had a row inserted for each run. The DBInsert program was

called individually for each table and iteration. The initial point chosen for this loop had a stage pressure ratio of 0.744 showing an increase of approximately 63% at the optimal design. This initial point was used for both the DOE and optimization runs. Even though this point would never be considered for a compressor the routines should still find the optimal point. The genetic algorithm will ensure that the search path does not stop in a local maximum and shows the flexibility of both the Pointer method and PODS.

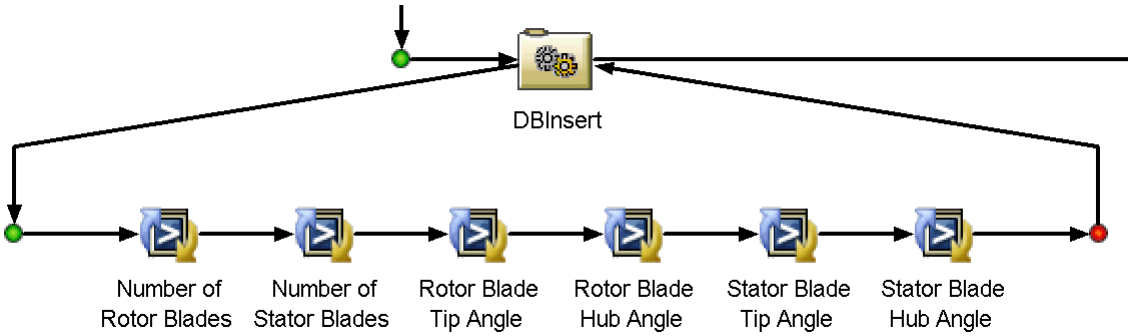


Figure 5-12 DBInsert subflow

A mesh of this fidelity does not produce reliable results, as seen in the surface plot in Figure 5-13. The figure shows that the results from each iteration have too much error to be able to accurately find the overall max or min. The history plot of Figure 5-14 shows the Pointer method search pattern. With such a coarse mesh, some maximum values were found but were not repeatable. A finer mesh is needed to create a smoother response surface. The optimal point found using the Pointer optimization routines had a stage pressure ratio of 1.21. Using the same analytical solution with 85% efficiency of 1.18, shows that the optimal point is fairly close.

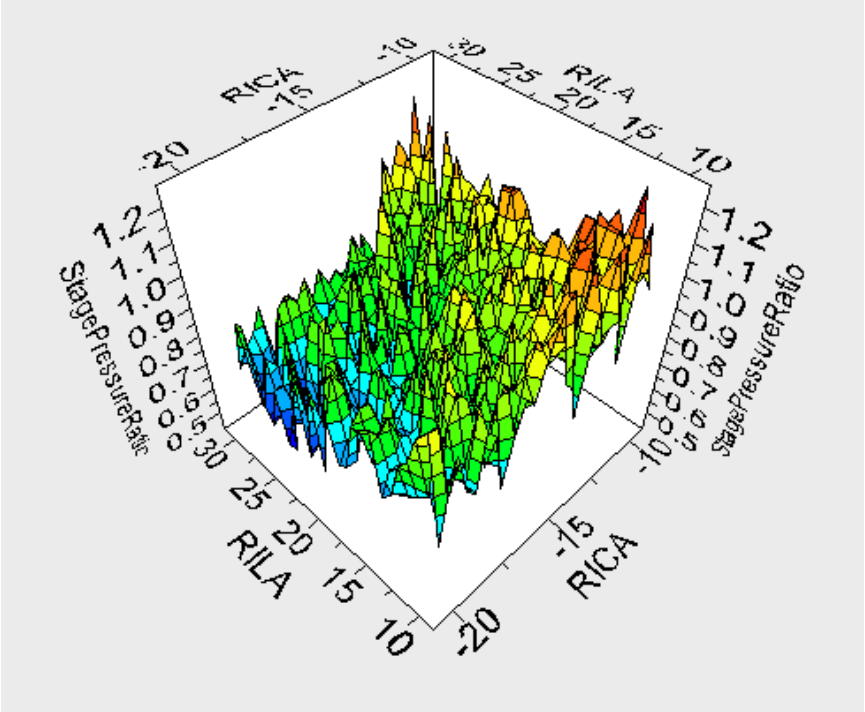


Figure 5-13 Rotor internal chord angle vs. lead angle

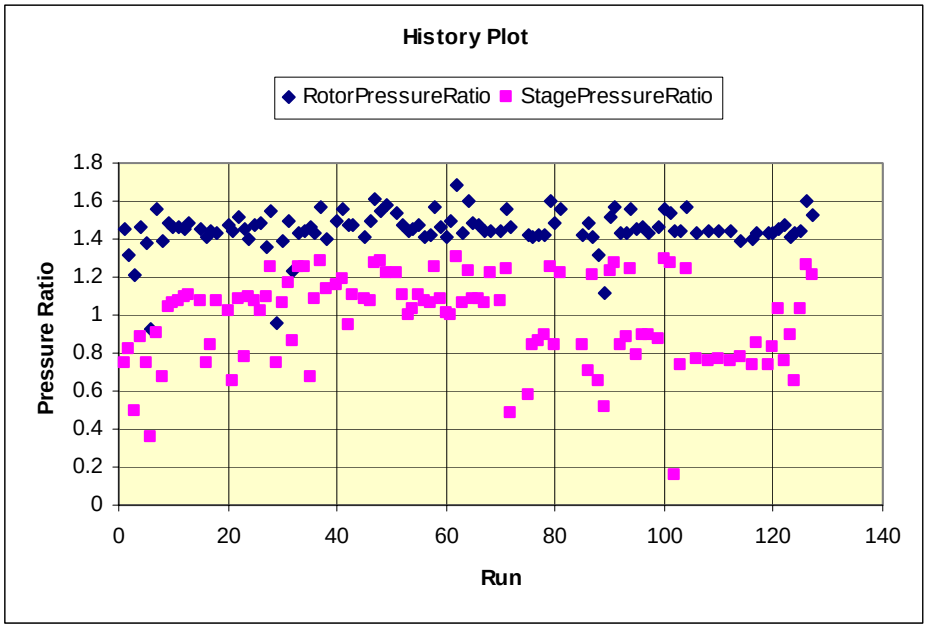


Figure 5-14 History plot of pressure ratios

The search pattern in Figure 5-14 is the result of the pointer method using several different algorithms in an attempt to avoid local max and min areas; however, since there is a potential error of approximately 4% in the pressure ratio, a local max may not be repeatable. Even though the only reliable way to optimize this solution is with a finer mesh that includes boundary layer conditions, the example problem demonstrates that complex analytical design may be optimized by integrating several major commercial programs.

6 Conclusions

With the completion of this thesis, industry now has a system to optimize a variable assembly of fluid domains that contain discrete features. Industry can now save time and money previously spent on the iterative process that occurs between different component groups. Preliminary design can find the optimal multi-component design at the start. The creation of this system included five programs, which comprised all of the PODS programs.

As a summary, the programs provide a system that uses a database to store all necessary data for the optimization of an assembly of fluid domains. PODS included DBInsert, INX, ENX, FJW and a TCL script. The PODS programs provided access to the database through C# functions, excluding the TCL script, while performing the necessary functions related to specific commercial, CAD/CAE programs. During the development process, the following three questions were answered:

1. What is the best practice for passing attributes/data between commercial programs?
2. Can a model with varying numbers of fluid domain interfaces be optimized while keeping arrays distinct from one domain to the next?

3. Can the process of updating the model, changing the number of features in an array and modifying the number of fluid domains be applied to optimization of legacy models?

Question one was addressed by storing and retrieving all of the data in a MySQL database. Previous research confronted the discrete feature problem with a CAD-centric model (King 2004). The limitation of this system was that all data was stored within the CAD domain and passed to CAE applications through a custom program that could only handle single domain models. This thesis provides a system that is Knowledge-centric. All of the non-geometric data is stored in a central database, so that the data is accessible to any peripheral application. In addition, most of the geometric data was also passed to each of the applications through the database. The PODS system captures all of the design intent in the database making access to that knowledge faster for all processes in industry.

While iSIGHT-FD handles all of the optimization routines, all data and program modifications were accomplished using custom code, since iSIGHT-FD cannot handle assembly or discrete feature optimization by itself. The built-in import/export functionality from each application was used to track Face IDs from one application to the next. While using the import/export functions to track face IDs proved the easiest for this set of applications, the process of passing face IDs would be more effective in the database for incorporating new applications. The advantage of this system is that all knowledge is accessible to all programs in a format that is faster and easier to search than typical text documents.

The second and third questions are addressed using the custom PODS codes. Using PODS the number of features in any array, in any domain, can increase or decrease with each iteration and the optimization workflow will complete. Additionally, the number of domains can increase or decrease with the same results. No other optimization system can claim this capability. By establishing loops in the INX, ENX and HyperMesh application, any legacy parametric model, with arrays and separate domains, can by analyzed in the same way. Legacy models still require setup of database tables as well as modifications to the Fluent inputs. This process streamlines the process of analyzing complex fluid domains and shortens the time to market.

Another conclusion of the Knowledge-centric solution is that the use of interactive models with programmatic interfaces between the model and the knowledge base is more efficient than creating the model programmatically. Interactive modeling reduces the development time to creation of parametric models or parameterization of legacy models. The programmatic interface makes assignment of BCs simpler and quicker than current techniques required for multi-fluid domains with discrete features. However, since PODS created a shell of the base model for the import/export functions, this process could still be improved by storing the face IDs for downstream applications in the database.

The programs were tested on assemblies ranging from one to three components without any difficulties. The Parametric Optimization Design System may be used for any multi-fluid domain problem. As processing power can handle more nodes and larger models, more domains can be optimized. The iterative methods required by separate component groups in industry will no longer be necessary. While a full engine has not yet

been analyzed with this system, the data structure, database service tools and software interfaces for a complete engine have been developed and successfully demonstrated. Optimization of full engine assemblies is achievable. PODS makes optimization of multi-fluid domains, which contain discrete features, possible and easier to setup for all of industry. With the completion of this thesis there is the potential to eliminate the process of passing inputs, outputs and constraints between individual component groups in order to find the optimal engine structure.

6.1 Further Work

There are many areas where the work in this thesis can be expanded upon. The following list outlines some of the critical areas where future research is needed:

1. Mesh algorithm improvement
2. Fluent setup
3. Face interference
4. Automatic relational database creation
5. Database storage of complete assemblies

The programs need to have certain algorithms built-in to the meshing process to be able to fully handle any model that is “dropped in.” One improvement to the meshing process would be similar to the INX program with application to HyperMesh. A selection GUI could be created that would set up all of the edge node conditions and create a default mesh. The GUI could contain other routines for dividing faces in order to achieve better surface approximations. Each iteration after would run through the same process as the ENX system by checking the node parameters of the base feature of the array and

copying them to the updated array. As meshing algorithms improve, the boundary layers can be included in the process as well. This is also dependant on a better understanding of the node equivalencing difficulties that were addressed in the body of this thesis.

To improve the Fluent setup, the range of possible inputs needs to be considered. An additional GUI should be created that supplies a list of all the possible fluid conditions and the choices that apply. This follows along similar lines as the INX and HyperMesh GUIs. Setup of all initial conditions can be stored in the database and passed from one iteration to the next. The development of a CFD GUI would be time consuming but would reduce the time needed to adjust code as the current system is built.

Another difficulty occurs when the INX or ENX program updates either the twist angle or the number of blades. As these faces change, they can start to overlap other blades, creating a part with blade interference and splitting of the inner or outer walls. Wall splitting prevents the meshing information from being assigned to all faces and will not import properly into HyperMesh. Future work could include routines to prevent interference conditions from occurring. One method to overcome this may be to add a program function that counts the number of faces in the model. If the number is not consistent with the expected changes the iteration is rejected. The advantage to creating an algorithm to control these limits would mean that the design space can be left wide open. Designs may be found this way that would never have been considered.

More work should be done to improve the overall table design. Additional research should be done to store and retrieve the parameters from a relational database scheme. Creation of a GUI, which enables users to select which focus parameters, or custom BC types, they want to include in the DB tables, and automatically generate them,

would greatly enhance this process. Use of attribute standardization would improve the storage of data for other MDO applications by establishing a working relational database. Custom BC types would facilitate many more model applications, which would also increase the potential for MDO.

Finally, if this research could be combined with attribute standardization and with a relational DB, not only could the attributes be stored, full assemblies could be stored within the DB instead of as a basic text file. A new standard, which uses a relational database to store all model information, could be used instead of the current export standards of STEP or IGES. Attributes, features, parts, assemblies, analysis inputs and results could all be saved-to and retrieved from a database. Direct modification of a model could happen without ever opening the CAD system during optimization, decreasing design time even further.

The work done in this thesis is a major step forward in design optimization applied to complex fluid/structure systems, wherein each physical system is simulated using major commercial codes with large discretized models. It is applicable to any multi-disciplinary domain and limited only by element limits.

As computation power and speed increase, there should be rapidly increasing numbers of similar applications emerge, with increasing complexity, resulting in improved system performance and decreased development time.

7 References

Altmeyer, J., S. Ohnsorge, and B. Shürmann.. ”Reuse of Design Objects in CAD Frameworks.” *Association for Computing Machinery* (1994): 754-761.

Anderl, R. and R. Mendgen. “Modelling with constraints: theoretical foundation and application” *Computer-Aided Design,* vol. 28, no. 3 (1996): 155-168.

Baizet, Y., F. Pourroy, and R. Allera. “Towards a Knowledge-based engineering system to support computational simulation activities at Renault Company.” In *Proceeding of the 10th ISPE International Conference on Concurrent engineering: the vision for the future generation in research and applications* vol. 1 (2003): 475-482.

Baker, T. “Attribution Standardization for Integrated Concurrent Engineering” M.S. Thesis, Brigham Young University, 2005.

Balachandran, M., and J.S., Gero. “A Knowledge-based approach to mathematical design modeling and optimization.” *Engineering Optimization* vol. 12 (1987): 91-115.

Borup, L., and A. Parkinson. “Comparison of four non-derivative optimization methods on two problems containing heuristic and analytic knowledge.” *Advances in Design Automation,* vol. 1 (1992): 137-143.

Bowland, N.W., and J.X. Sharma. “A PDM- and CAD-integrated assembly modeling environment for manufacturing planning.” *Journal of Materials Processing Technology* 138 (2003): 82-88.

Brown, D.R. “Knowledge-based engineering analysis” Ph. D. diss., Stanford University, 1988.

BYU/PACE Tutorial. Loss Through an Orifice: 2D Axisymmetric Flow.

Chern, J.H. “Knowledge-based engineering in concurrent engineering automation. *Application of artificial intelligence in engineering VII* vol. 1 (1992): 289-302.

.

Delap, D.C. “CAD-Based Creation and Optimization of a Gas Turbine Flowpath Module with Multiple Parameterizations” M.S. Thesis, Brigham Young University, 2003.

Deitel, H. M., and P. J. Deitel, ed. 2003. *C++ How to Program,* 4th ed. Pearson Education, Inc.

Ellis, R., and D. Gulick. *Calculus: One and Several Variables.* Saunders College Publishing, 1991.

Fan, I.S., G. Li, M. Lagos-Hernandez, P. Bermel-García, and M. Twelves. “A rule level knowledge management system for knowledge based engineering applications.” *Proceedings of the … ASME Design Engineering Technical Conferences* vol. 1 (2002): 813-821.

Fife, N. L. “Developing a Design Space Model Using a Multidisciplinary Design Optimization Schema in a Product Lifecycle Management System to Capture Knowledge for Reuse” M.S. Thesis, Brigham Young University 2005.

Fluent News, http://www.fluent.com/about/news/newsletters/00v9i2/s2.htm (Retrieved November, 2007).

Grönstedt, T. “Development of methods for analysis and optimization of complex jet engine systems” Ph.D. Thesis, Chalmers University of Technology, 2000.

Hejlsberg, A., S. Wiltamuth and P. Golde. *The C# Programming Language.* Addison – Wesley, 2004.

Hogge, D.G. “Integrating Commercial CAx Software to Perform Multidisciplinary Design Optimization” M.S. Thesis, Brigham Young University, 2002.

King, M.L. “A CAD-centric Approach to CFD Analysis with Discrete Features,” M.S. Thesis, Brigham Young University, 2004.

King, M. L., M.. J. Fisher and C. G. Jensen. “A CAD-centric Approach to CFD Analysis with Discrete Features.” *Computer-Aided Design and Applications.* Vol. 3, No.4 (2006) 279-288.

Layton, R., and J. Marra.. “Conceptual basis for a new approach to bladed-disk design.” *ASME Journal of Engineering for Gas Turbines and Power* (April 2000): 321-325.

Lindby, T., and J. L. T. Santos. “Shape design sensitivity analysis and optimisation with an existing associative CAD system.” *American Institute of Aeronautics and Astronautics*: (1994)1483-1490.

Liu, G., and J. He. “New research and concepts in turbo-jet engine design.” *Aircraft Engineering and Aerospace Technology*; vol. 69, Iss. 6 (1997): 527.

Lund, J. G. “The Storage of Parametric Data in Product Lifecycle Management Systems” M.S. Thesis, Brigham Young University, 2006.

Marra, J. “Use of knowledge-based engineering in compressor rotor design.” *Mechanical Engineering Magazine* (April 1997).

Mattingly, J., and H. Ohain. *Elements of Propulsion: Gas Turbines and Rockets.* American Institute of Aeronautics and Astronautics, 2006.

Patankar, S. *Numerical Heat Transfer and Fluid Flow*. Taylor & Francis, 1980.

Pinfold, M., and C. Chapman. “Using knowledge based engineering to automate the post-processing FEA results.” *International Journal of Computer Application in Technology*; vol. 21, iss. 3 (1997): 99-106.

Requicha, A. A. G. “Representations for Rigid Solids: Theory, Methods and Systems.” *ACM Computing Surveys*, 12(4) (December 1980):437-464.

Rohm III, T., S. Tucker, C. Jones, and G. Jensen. “Parametric design tools and applications.” In *Proceedings of DETC2000: ASME Design Automation Conference in Baltimore, Maryland* (10-13 September 2000): 657-664.

Sederberg, T. W. (January 19, 2007).*Computer Aided Geometric Design Course Note* http://tom.cs.byu.edu/~557/ (Retrieved November, 2007).

Schlülter, J., X.. Wu., H. Pitsch, S. Kim, and J. Alonso. “Integrated simulations of a compressor/combustor assembly of a gas turbine engine.” In *Proceedings of GT2005 ASME Turbo Expo: Power for Land, Sea and Air in Reno-Tahoe, Nevada.*(6-9 June 2005) .

Schuhmacher, G. “Optimizing Aircraft Structures.” *Concept to Reality*, Winter 2006.

Shapiro, V., and D. Vossler. “What is a parametric family of solids?” *Association of Computing Machinery* (1995): 43-54.

Ullman, L., ed. *Visual Quickstart Guide: My SQL,* 2nd ed. Peachpit Press, 2006.

Vogel, A. “A knowledge-based approach to automated flow field zoning for computational fluid dynamics.” Ph. D. diss., Stanford University, 1989.

Weiss, J. M and F. J. Kelecy. “Numerical Simulation of Steady-State Flow Through a Multi-Stage Turbine Using Unstructured Meshes.” In *Proceedings of the 3rd ASME/JSME Joint Fluids Engineering Conference in San Francisco, California, 18-23 July, 1999.*

Wikipedia contributors (2006). Multidisciplinary design optimization. *Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Multidisciplinary_design_optimization&oldid=45307534 (Retrieved April 8, 2006)

Zhongtu, L., W. Qifu, and C. Liping. "A knowledge-based approach for the task implementation in mechanical product design." *International Journal of Advanced Manufacturing Technology* 29 2006: 837-845.

Appendix A. Face Extraction Class

```
using System;
using System.Collections;
using System.Windows.Forms;

namespace FaceExtraction
{
   /// <summary>
   /// Extractor contains all of the functions that
   /// modify the NX assembly or part.    It allows the
   /// user to select faces and apply boundary conditions
   /// to those faces. The boundary conditions are
   /// assigned to the model faces as attributes and
   /// then extracted by this class.
   /// </summary>
   public class Extractor
   {
      public static ArrayList FaceArray;
      public static string BCD = "null";
      public Extractor()
      {
         //
         // TODO: Add constructor logic here
         //
      }
      public static void ExtractFaces()
      {
         NXOpen.Session.UndoMarkId myMark =
            NXInteraction.mySession.SetUndoMark(
            NXOpen.Session.MarkVisibility.Visible,"Extracting Faces");
         try
         {
            int j = 0;
            foreach(NXOpen.Body[] bA in NXInteraction.bArray)
            {
               NXOpen.PartLoadStatus partLoadStatus;
               NXInteraction.mySession.Parts.SetWorkComponent(
                  NXInteraction.componentArray[j], out partLoadStatus);
               NXInteraction.workPart =
                  NXInteraction.mySession.Parts.Work;
               foreach(NXOpen.Body b in bA)
               {
                  NXOpen.Features.Feature feature1 =
                     NXInteraction.workPart.Features.FindObject(
                     b.JournalIdentifier);
```

```
            NXOpen.Face [] faces;
            faces = b.GetFaces();
            NXOpen.Tag[] myTag = new NXOpen.Tag[faces.Length];
            for(int i = 0; i < faces.Length; i++)
            {
               NXInteraction.myUFSession.Modl.ExtractFace(
                  faces[i].Tag,0,out myTag[i]);
               NXOpen.Body myObject
                  =  (NXOpen.Body)NXOpen.Utilities
                     .NXObjectManager.Get(myTag[i]);
               NXOpen.Features.BodyFeature myBody
                  = (NXOpen.Features.BodyFeature)NXInteraction.
                     workPart.Features.FindObject(
                     myObject.JournalIdentifier);
               myObject.SetAttribute(
                  "Boundary_Condition",faces[i].
                  GetStringAttribute("BOUNDARY_CONDITION"));
               myBody.SetName("EXTRACTED");
               myBody.SetAttribute("EXTRACT","CHILD");
               myBody.SetAttribute("Boundary_Condition",faces[i].
                  GetStringAttribute("BOUNDARY_CONDITION"));
               ApplyColor(myObject);
            }
         }
         j++;
      }
      ModTransmitter.ArrayBuilder();
   }
   catch
   {
      NXInteraction.mySession.UndoToMark(
         myMark,"Extracting Faces");
      System.Windows.Forms.MessageBox.Show("All faces must have
         boundary condition assigned!","Error",
         MessageBoxButtons.OK,MessageBoxIcon.Error);
   }
}
public static void DeleteExtractedFaces()
{
   int i = 0;
   foreach(NXOpen.Body[] bA in NXInteraction.bArray)
   {
      NXOpen.PartLoadStatus partLoadStatus;
      NXInteraction.mySession.Parts.SetWorkComponent(
         NXInteraction.componentArray[i], out partLoadStatus);
      NXInteraction.workPart =
         NXInteraction.mySession.Parts.Work;
      foreach(NXOpen.Body b in bA)
      {
         NXOpen.Features.Feature feature1 =
            NXInteraction.workPart.Features.FindObject(
            b.JournalIdentifier);
         if(feature1.Name.Equals("EXTRACTED"))
         {
            NXOpen.Session.UndoMarkId myMark =
               NXInteraction.mySession.SetUndoMark(
               NXOpen.Session.MarkVisibility.Visible,
```

```
                "Delete Extracted Faces");
            try
            {
                NXOpen.NXObject Objects  =
                    NXInteraction.workPart.Bodies.FindObject(
                    b.JournalIdentifier);
                NXInteraction.mySession.
                    UpdateManager.AddToDeleteList(Objects);
                NXInteraction.mySession.
                    UpdateManager.DoUpdate(myMark);
            }
            catch(NXOpen.NXException e)
            {
                NXInteraction.mySession.UndoToMark(
                    myMark,"Delete Extracted Faces");
                NXInteraction.mySession.
                    ListingWindow.WriteLine(e.ToString());
                NXInteraction.mySession.
                    ListingWindow.WriteLine(e.Message);
            }
         }
      }
      i++;
   }
   NXOpen.PartLoadStatus pls;
   NXInteraction.mySession.Parts.SetWorkComponent(
      NXInteraction.rootComp,out pls);
   NXInteraction.workPart = NXInteraction.mySession.Parts.Work;
   ModTransmitter.ArrayBuilder();
}

public static void AssignAttribute()
{
   if(FaceArray != null)
   {
      foreach(NXOpen.Face s in FaceArray)
      {
         s.SetAttribute("Boundary_Condition",BCD);
         ApplyColor(s);
      }
      FaceArray = null;
   }
}

public static void DeleteHMModifications()
{
   foreach(NXOpen.Body[] bA in NXInteraction.bArray)
      foreach(NXOpen.Body b in bA)
      {
         NXOpen.Face [] faces = b.GetFaces();
         foreach(NXOpen.Face f in faces)
            f.DeleteAllAttributesByType(
         NXOpen.NXObject.AttributeType.String);
      }
}
private static void ApplyColor(NXOpen.Face s)
{
```

```
    //To get the colors by ID number open the color pallete in UG
    int colorIndex = 87;
    if(BCD == "inlet")
       colorIndex = 186;//Red
    if(BCD == "inlet interface")
       colorIndex = 78;//Red
    if(BCD == "outlet")
       colorIndex = 213;//Dark Faded Blue
    if(BCD == "outlet interface")
       colorIndex = 103;//Dark Faded Blue
    if(BCD == "wall")
       colorIndex = 125;//Dark Dull Orange
    NXOpen.DisplayModification displayMod =
       NXInteraction.mySession.
       DisplayManager.NewDisplayModification();
    displayMod.ApplyToAllFaces = false;
    displayMod.NewColor = colorIndex;
    NXOpen.DisplayableObject[] objArray =
       new NXOpen.DisplayableObject[1];
    objArray[0] = s;
    displayMod.Apply(objArray);
}
private static void ApplyColor(NXOpen.Body b)
{
    try
    {
       string color = b.GetStringAttribute("BOUNDARY_CONDITION");
    //To get the colors by ID number open the color pallete in UG
       int colorIndex = 87;
       if(color == "inlet")
          colorIndex = 186;//Red
       if(color == "inlet interface")
          colorIndex = 78;//Orange Orange Yellow
       if(color == "outlet")
          colorIndex = 213;//Dark Faded Blue
       if(color == "outlet interface")
          colorIndex = 103;//Azure Azure Cyan
       if(color == "wall")
          colorIndex = 125;//Dark Dull Orange
       NXOpen.DisplayModification displayMod =
          NXInteraction.mySession.
          DisplayManager.NewDisplayModification();
       displayMod.ApplyToAllFaces = false;
       displayMod.NewColor = colorIndex;
       NXOpen.DisplayableObject[] objArray =
          new NXOpen.DisplayableObject[1];
       objArray[0] = b;
       displayMod.Apply(objArray);
       Console.WriteLine(b.Color);
    }
    catch(NXOpen.NXException e2)
    {
       MessageBox.Show(e2.Message,"Error",
          MessageBoxButtons.OK,MessageBoxIcon.Error);
    }
}
```

```
public static void RestoreDefaultColor()
{
   try
   {
      foreach(NXOpen.Body[] bA in NXInteraction.bArray)
         foreach(NXOpen.Body b in bA)
         {
            NXOpen.Face [] faces = b.GetFaces();
            NXOpen.DisplayModification displayMod =
               NXInteraction.mySession.
               DisplayManager.NewDisplayModification();
            displayMod.ApplyToAllFaces = false;
            displayMod.NewColor = 87;
            NXOpen.DisplayableObject[] objArray =
               new NXOpen.DisplayableObject[faces.Length];
            objArray = faces;
            displayMod.Apply(objArray);
         }
   }
   catch(NXOpen.NXException e2)
   {
      MessageBox.Show(e2.Message,"Error",
         MessageBoxButtons.OK,MessageBoxIcon.Error);
   }
}

public static void AssignLayers()
{
   try
   {
      NXOpen.Body[] oneArray = new NXOpen.Body[1];
      for(int i = 0; i< NXInteraction.componentArray.Length; i++)
      {
         foreach(NXOpen.Body bo in NXInteraction.bArray[i])
         {
            Console.WriteLine(bo.Color);
            if(bo.Color != 87)
            {
               oneArray[0] = bo;
               string BC =
                  bo.GetStringAttribute("BOUNDARY_CONDITION");
               if(BC == "inlet")
               {
                  int m = 101;
                  NXInteraction.workPart.
                     Layers.MoveObjects(m,oneArray);
               }
               if(BC == "inlet interface")
               {
                  int m = 102;
                  NXInteraction.workPart.
                     Layers.MoveObjects(m,oneArray);
               }
               if(BC == "wall")
               {
                  int m = 103;
                  NXInteraction.workPart.
```

```
                    Layers.MoveObjects(m,oneArray);
                }
                if(BC == "outlet interface")
                {
                   int m = 104;
                   NXInteraction.workPart.
                      Layers.MoveObjects(m,oneArray);
                }
                if(BC == "outlet")
                {
                   int m = 105;
                   NXInteraction.workPart.
                      Layers.MoveObjects(m,oneArray);
                }
             }
          }
       }
       NXOpen.Session.UndoMarkId myMark =
          NXInteraction.mySession.SetUndoMark(
          NXOpen.Session.MarkVisibility.Visible,"Update Layers");
       try
       {
          NXInteraction.mySession.UpdateManager.DoUpdate(myMark);
       }
       catch(NXOpen.NXException e)
       {
          NXInteraction.mySession.UndoToMark(
             myMark,"Update Layers");
          NXInteraction.mySession.
             ListingWindow.WriteLine(e.ToString());
          NXInteraction.mySession.
             ListingWindow.WriteLine(e.Message);
       }
    }
    catch(NXOpen.NXException e3)
    {
       MessageBox.Show(e3.Message,"Error",
          MessageBoxButtons.OK,MessageBoxIcon.Error);
    }
}
private static void HighlightFace(
    NXOpen.Face face, bool highlight)
{
    if (face != null)
    {
       if (highlight)
          face.Highlight();
       else
          face.Unhighlight();
    }
}

public static void UnselectFaces()
{
    foreach(NXOpen.Body[] bA in NXInteraction.bArray)
       foreach(NXOpen.Body b in bA)
       {
```

```
                NXOpen.Face[] faces;
                faces = b.GetFaces();
                foreach(NXOpen.Face f in faces)
                {
                   HighlightFace(f, false);
                }
             }
          }

      }
   }
```

Appendix B. Attribute Modification Transmiter Class

```
using System;
using NXOpen.Utilities;
namespace FaceExtraction
{
   /// <summary>
   /// ModTransmitter contains the functions to build the arrays of
   body features that are used by the other classes for
   /// manipulating the part. It's primary function is to ensure
   /// that all attributes and faces have been transmitted from the
   /// base feature of a discete feature to all instances of that
   /// feature.
   /// </summary>
   public class ModTransmitter
   {
      public ModTransmitter()
      {
         // TODO: Add constructor logic here
      }
      public static void ArrayBuilder()
      {
         try
         {
            NXInteraction.componentArray = null;
            NXInteraction.rootComp = NXInteraction.mySession.Parts.
               Display.ComponentAssembly.RootComponent;
            if(NXInteraction.rootComp.GetChildren().Length != 0)
            {
               NXInteraction.componentArray =
                  NXInteraction.rootComp.GetChildren();
            }
            NXInteraction.bArray = new NXOpen.Body [
               NXInteraction.componentArray.Length][];
            for(int i = 0;
               i < NXInteraction.componentArray.Length; i++)
            {
               NXOpen.PartLoadStatus partLoadStatus;
               NXInteraction.mySession.Parts.SetWorkComponent(
                  NXInteraction.componentArray[i],out partLoadStatus);
               NXInteraction.workPart =
                  NXInteraction.mySession.Parts.Work;
               NXInteraction.bArray[i] =
                  NXInteraction.workPart.Bodies.ToArray();
            }
            NXOpen.PartLoadStatus pls;
```

```
            NXInteraction.mySession.Parts.SetWorkComponent(
               NXInteraction.rootComp,out pls);
            NXInteraction.workPart =
               NXInteraction.mySession.Parts.Work;
         }
         catch
         {
            NXInteraction.componentArray =
               new NXOpen.Assemblies.Component[1];
            NXInteraction.bArray = new NXOpen.Body [1][];
            NXInteraction.workPart =
               NXInteraction.mySession.Parts.Work;
            NXInteraction.bArray[0] =
               NXInteraction.workPart.Bodies.ToArray();
            NXInteraction.workPart =
               NXInteraction.mySession.Parts.Work;
         }
      }
      public static void Transmitter()
      {
         ArrayBuilder();
         foreach(NXOpen.Body[] bA in NXInteraction.bArray)
         {
            Extractor.FaceArray = new System.Collections.ArrayList();
            foreach(NXOpen.Body b in bA)
            {
               NXOpen.Face[] faces;
               faces = b.GetFaces();
               foreach(NXOpen.Face f in faces)
               {
                  NXOpen.Features.Feature feature1 =
                     NXInteraction.workPart.
                     Features.GetAssociatedFeature(f);
                  if(feature1.FeatureType == "INSTANCE")
                  {
                     Extractor.FaceArray.Add(f);
                     NXOpen.Features.Feature  [] child =
                        feature1.GetChildren();
                     if(child.Length != 0)
                     {
                        if(child[0].FeatureType == "CIRCULAR_ISET")
                        {
                           NXOpen.Face tempFace =
                              NXInteraction.featToFace(feature1);
                           Extractor.BCD =
                              tempFace.GetStringAttribute(
                              "BOUNDARY_CONDITION");
                        }
                     }
                  }
               }
               Extractor.AssignAttribute();
            }
         }
      }
   }
}
```

Appendix C. NX Interfacing Class

```
using System;
using System.IO;
using System.Collections;
using System.Runtime.InteropServices;
using System.Windows.Forms;
using NXOpen;
using NXOpen.UF;
using NXOpen.Utilities;

namespace FaceExtraction
{
   /// <summary>
   /// This class opens and closes the NX connection
   /// and stores all the instances of the NX classes necessary
   /// for manipulation. It also has a few member for manipulating
   /// the classes.
   /// </summary>
   public class NXInteraction
   {
      public static Session mySession;
      public static UFSession myUFSession;
      public static Part workPart;
      public static NXOpen.Features.Feature nullFeatures_Feature =
         null;
      public static NXOpen.Body [][] bArray;
      public static NXOpen.Assemblies.Component [] componentArray;
      public static ArrayList modelArrays;
      public static ArrayList [] modelArrayInstances;
      public static NXOpen.Assemblies.Component rootComp;

      public NXInteraction()
      {
         try
         {
            ModTransmitter.ArrayBuilder();
         }
         catch
         {
            MessageBox.Show("Empty part, model a fluid domain
               first!","Error",
               MessageBoxButtons.OK,MessageBoxIcon.Error);
         }
      }
      public static void Main(string[] args)
```

```
        {
            NXInteraction interact = new NXInteraction();
        }

        public static NXOpen.Features.Feature stringToFeat(string s)
        {
            NXOpen.Tag tempTag = (NXOpen.Tag)System.UInt32.Parse(s);
            NXOpen.Features.Feature tempFeat =
                (NXOpen.Features.Feature)NXObjectManager.Get(tempTag);
            return tempFeat;
        }

        public static NXOpen.Face featToFace(NXOpen.Features.Feature
            feat)
        {
            NXOpen.Features.BodyFeature bFeat =
                (NXOpen.Features.BodyFeature)feat;
            NXOpen.Face [] face = bFeat.GetFaces();
            return face[0];
        }

        public static NXOpen.Face stringToFace(string s)
        {
            NXOpen.Tag tempTag =
                (NXOpen.Tag)System.UInt32.Parse(s.ToString());
            NXOpen.Face face = (NXOpen.Face)NXObjectManager.Get(tempTag);
            return face;
        }
        public static int GetUnloadOption(string arg)
        {
// Return code to indicate this library can be
// unloaded by the user.
            return (int) Session.LibraryUnloadOption.Explicitly;
        }
    }
}
```

Appendix D. MySQL Export

```
using System;
using System.Text.RegularExpressions;
using MySQLDriverCS;
using NXOpen;
using NXOpen.UF;
using NXOpen.Utilities;
using FaceExtraction;

namespace ExternalNX
{
   /// <summary>
   /// Summary description for MySQLExport.
   /// </summary>
   public class MySQLExport
   {
      string attrString;
      MySQLConnection con;
      MySQLDataReader reader;
      public MySQLExport()
      {
         DBExport();
         //
         // TODO: Add constructor logic here
         //
      }
      public void DBExport()
      {
         try
         {
            string ip = "127.0.0.1";
            con = new MySQLConnection( new MySQLConnectionString(
               ip,
               "foilparams",
               "root",
               "root").AsString );
            con.Open();
            updateExpressions();
            con.Close();
         }
         catch(Exception ee)
         {
```

```
            Console.WriteLine( ee.ToString() );
        }
    }
    public int updateExpressions()
    {
        ModTransmitter.ArrayBuilder();
        //loop thru assembly parts
        int compCount = NXInteraction.componentArray.Length;
        if(compCount != 0)
        {
            foreach(NXOpen.Assemblies.Component comp in
                NXInteraction.componentArray)
            {
                NXOpen.PartLoadStatus partLoadStatus;
                NXInteraction.mySession.Parts.SetWorkComponent(comp,out
                    partLoadStatus);
                NXInteraction.workPart =
                    NXInteraction.mySession.Parts.Work;
                importTable();
            }
        }
        //close database connection
        return 0;
    }
    public void updateExpression(string exprName,string exprVal)
    {

        bool is_exp_in_part = true;
        NXInteraction.myUFSession.Modl.IsExpInPart
            (NXInteraction.workPart.Tag,exprName,out is_exp_in_part);
        if(is_exp_in_part)
        {
            NXInteraction.myUFSession.Modl.EditExp(exprName + "=" +
                exprVal);
        }
        else
        {
            NXInteraction.myUFSession.Modl.CreateExp(exprName + "=" +
                exprVal);
        }
    }
    private void getAttrString()
    {
        string attrName = "TABLE";
        attrString =
            NXInteraction.workPart.GetStringAttribute(attrName);
    }
    private int importTable()
    {
        getAttrString();
        int l = attrString.Length;
        if (l == 0)
            return 1;
        foreach(string token in Regex.Split(attrString,","))
        {
            string query   = "SELECT * FROM "
                        + token.Trim()
```

```
                    + " ORDER BY id DESC LIMIT 1";
            MySQLCommand cmd = new MySQLCommand(query,con);
            reader = cmd.ExecuteReaderEx();
            while(reader.Read())
            {
               for(int m=0; m < reader.FieldCount; m++)
               {
                  string name = reader.GetName(m);
                  string val = reader.GetString(m);
                  updateExpression(name,val);
               }
            }
            reader.Close();
         }
         return 0;
      }
   }
}
```

Appendix E. ENX Application

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Runtime.InteropServices;
using NXOpen;
using NXOpen.UF;
using NXOpen.Utilities;
using FaceExtraction;


namespace ExternalNX
{
   /// <summary>
   /// Summary description for ExternalNXMain.
   /// </summary>
   public class ExternalNXMain
   {
      [DllImport("MySQLExport.dll")]
      public static extern void myExport();
      NXOpen.Session.UndoMarkId cancelMark;
      private static NXInteraction Inter;
      private static ExternalNXMain NXMain;
      public ExternalNXMain()
      {
         cancelMark = NXInteraction.mySession.SetUndoMark
            (NXOpen.Session.MarkVisibility.Visible,"Undo all
               modifications");
         Extractor.DeleteExtractedFaces();
      }
      public static void Main(string[] args)
      {
         NXInteraction.mySession = Session.GetSession();
         NXInteraction.myUFSession = UFSession.GetUFSession();
         NXOpen.PartLoadStatus loadStatus;
         NXInteraction.mySession.Parts.OpenDisplay(args[0],out
            loadStatus);
         NXInteraction.workPart = NXInteraction.mySession.Parts.Work;
         NXInteraction.mySession.ListingWindow.Open();
         Inter = new NXInteraction();
         NXMain = new ExternalNXMain();
         NXInteraction.mySession.ListingWindow.WriteLine("Hello");
         MySQLExport MSE = new MySQLExport();
         ModTransmitter.Transmitter();
```

```
            Extractor.ExtractFaces();
            Extractor.AssignLayers();
            NXOpen.PartSaveStatus PSS;
            bool modified;
            NXInteraction.mySession.Parts.SaveAll(out modified,out PSS);
        }
    }
}
```

Appendix F. Fluent Journal Writer

```
using System;
using System.IO;
using System.Collections;
using MySQLDriverCS;

namespace FluentJOUWriter
{
   /// <summary>
   /// Summary description for Writer.
   /// </summary>
   public class Writer
   {
      MySQLConnection con;
      MySQLDataReader reader;
      TextWriter outfile;
      int rowCount;
      public static ArrayList AssemOrder;
      public static ArrayList CompAngVel;
      public Writer()
      {
         DBConnect();
         WriteJournal();
      }
      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main(string[] args)
      {
         Writer write = new Writer();
         System.IO.File.Delete("c:\\a#\\Thesis\\engine.nas");
         System.IO.File.Delete(
            "c:\\a#\\Thesis\\fluentFiles\\fluentOut.txt");
      }
      public void WriteJournal()
      {
         con.Open();
         DBExtract();
         outfile = new StreamWriter("c:\\a#\\Thesis\\engine.jou");
         outfile.WriteLine("file set-batch-options yes yes no");
         outfile.WriteLine("file import nastran
            c:\\a#\\Thesis\\engine.nas");
         outfile.WriteLine("grid check");
         //scale inches into meters
```

```
outfile.WriteLine("grid scale 0.0254 0.0254 0.0254");
outfile.WriteLine("define models energy yes no no no yes");
outfile.WriteLine("define models unsteady-1st-order? yes");
//pressure in pascal
outfile.WriteLine("define operating-conditions operating-
   pressure 995603");
outfile.WriteLine("define boundary-conditions zone-type
   inflow"
               +AssemOrder[0]
               +" mass-flow-inlet");
outfile.WriteLine("define boundary-conditions mass-flow-inlet
   , , 75.75 , 585 , 995603 ,,,,,,,,,");
int i;
for(i = 0; i < rowCount; i++)
   if((string)CompAngVel[i]!="0")
      outfile.WriteLine(
         "define boundary-conditions fluid fluid"
         +AssemOrder[i]
         +", , , no , yes , , ,"
         +CompAngVel[i]
         +", , , 1 0 0 , ,");
outfile.WriteLine(
   "define boundary-conditions zone-type outflow"
   +AssemOrder[0]
   +" interface"  );
for(i = 1; i < rowCount-1; i++)
{
   outfile.WriteLine(
      "define boundary-conditions zone-type
      outflow"
      +AssemOrder[i]
      +" interface");
   outfile.WriteLine(
      "define boundary-conditions zone-type inflow"
      +AssemOrder[i]
      +" interface"  );
}
outfile.WriteLine(
   "define boundary-conditions zone-type inflow"
   +AssemOrder[i]
   +" interface"  );
for(i = 0; i < rowCount-1; i++)
{
   outfile.WriteLine(
      "define grid-interfaces create interface"
      +i
      +" outflow"
      +AssemOrder[i]
      +" inflow"
      +AssemOrder[i+1]
      +", ,");
}
outfile.WriteLine(
   "solve monitors residual monitor? , , , , ,");
outfile.WriteLine("solve monitors residual convergence
   criteria 0.01 , , , ,");
outfile.WriteLine("solve set under-relaxation pressure 0.1");
```

```
      outfile.WriteLine("solve set under-relaxation mom 0.1");
      outfile.WriteLine(
         "solve set under-relaxation temperature 0.3");
      outfile.WriteLine(
         "solve initialize set-defaults pressure 890000");
      outfile.WriteLine(
         "solve initialize set-defaults x-velocity 550");
      outfile.WriteLine(
         "solve initialize set-defaults temperature 560");
      outfile.WriteLine("define models viscous ke-standard yes");
      outfile.WriteLine("solve initialize initialize-flow");
      outfile.WriteLine("solve iterate 500");
      outfile.WriteLine("file start-transcript
         c:\\a#\\thesis\\fluentfiles\\fluentout.txt yes");
      string output = "report surface-int inflow" + AssemOrder[0];
      for(i = 0; i < rowCount; i++)
      {
         output = output + " outflow" + AssemOrder[i];
      }
      output = output + " , total-pressure";
      outfile.WriteLine(output);
      outfile.WriteLine("file stop-transcript");
      outfile.WriteLine("exit yes");
      con.Close();
      outfile.Close();
   }
   public void DBConnect()
   {
      try
      {
         con = new MySQLConnection( new MySQLConnectionString(
            "127.0.0.1",
            "foilparams",
            "root",
            "root").AsString );
      }
      catch(Exception ee)
      {
         Console.WriteLine( ee.ToString() );
      }

   }
   public void DBExtract()
   {
      rowCount = 0;
      string query = "SELECT module FROM assydata";
      MySQLCommand cmd = new MySQLCommand(query,con);
      reader = cmd.ExecuteReaderEx();
      AssemOrder = new ArrayList();
      while(reader.Read())
      {
         AssemOrder.Add(reader.GetString(0));
         rowCount++;
      }
      reader.Close();
      query = "SELECT angvel FROM assydata";
      cmd = new MySQLCommand(query,con);
```

```
            reader = cmd.ExecuteReaderEx();
            CompAngVel = new ArrayList();
            while(reader.Read())
            {
               CompAngVel.Add(reader.GetString(0));
            }
            reader.Close();
         }
      }
   }
```

Appendix G. DBInsert

```
using System;
using MySQLDriverCS;
namespace DBInsert
{
   /// <summary>
   /// Summary description for Class1.
   /// </summary>
   public class Insert
   {
      MySQLConnection con;
      string InsertInto;
      public Insert(string[] args)
      {
         foreach(string a in args)
            InsertInto = InsertInto+" "+a;
         DBConnect();
         con.Open();
         MySQLCommand cmd = new MySQLCommand(InsertInto,con);
         cmd.ExecuteNonQuery();
         con.Close();
      }
      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main(string[] args)
      {
         Insert insert = new Insert(args);
      }
      public void DBConnect()
      {
         try
         {
            con = new MySQLConnection( new MySQLConnectionString(
               "127.0.0.1", "foilparams", "root", "root").AsString );
         }
         catch(Exception ee)
         {
            Console.WriteLine( ee.ToString() );
         }
      }
   }
}
```

Appendix H. HyperMesh TCL Script

```
#----------include mysqltcl package----------
proc loadmysqltcl { dir } {
    set oldcwd [pwd]
    cd $dir
    load libmysqltcl[info sharedlibextension]
    cd $oldcwd
}
set dir "C:\\A#\\Thesis\\mysqltcl-3.03"
loadmysqltcl $dir
package ifneeded mysqltcl 3.03 [list loadmysqltcl $dir]
set outfile [open "hmerror.txt" w]
puts $outfile "opened: 1"
#-------import engine and equivalence egdes-------------
*setactivepage 6
*createstringarray 16 "SELECTIONS 0 1 0 0 0 1 0 0 0" "" "BEGIN_PARTS"
   "END_PARTS" "" "BEGIN_LAYERINFO" "DISABLE  " "ENABLE  101-105"
   "END_LAYERINFO" "" "COMPONENT_NAME custom <Layer><Part Name>" ""
   "COMPONENT_ATTRIBUTES <UG User Defined Attributes>" ""
   "MODEL_ATTRIBUTES <UG User Defined Attributes>" ""
*feinputwithdata {#ug\ug} {C:/A#/Thesis/engine.prt} 1 0 -0.01 1 1 1 12
*createmark surfaces 1 "all"
*selfstitchcombine 1 18 0.01 0.01

#------create surface mesh-------------
*setedgedensitylink 0
*elementorder 1

#----------open database and copy mesh size---------
set ip "127.0.0.1"
set db [::mysql::connect -host $ip -user "root" -password "root" -db
   "foilparams"]
set query "SELECT meshsize FROM assydata"
set dbSelect [::mysql::sel $db $query -list]
set mlist [lindex $dbSelect 0]
set compIds [ hm_complist id ];
*createmark surfaces 1 "all";
set meshSize [lindex $mlist 0]
*defaultremeshsurf 1 $meshSize 0 0 2 1 1 1 1 0 0 0 0
*removetempcleanupfile

set err 0

if {$err == 0} {
```

```
   #----------access database and copy interface mesh---------
   set query "SELECT id,module FROM assydata"
   set dbSelect [::mysql::sel $db $query -list]
   set modList [lindex [lindex $dbSelect 0] 1]
   for {set i 1} {$i < [llength $dbSelect]} {incr i} {
      lappend modList [lindex [lindex $dbSelect $i] 1]
   }
   if {[llength $modList] > 1} {
      for {set i 0} {$i < [expr [llength $modList] - 1]} {incr i} {
         set modout [lindex $modList $i]
         set modin [lindex $modList [expr $i + 1]]
         *createmark elems 1 "by comps" "102_$modin"
         *deletemark elems 1
   #     *morphupdateparameter "handlesize" 0.163802208
   #     *morphupdateparameter "handletolerance" 0.00655208833
         *createmark elems 1 "by comps" "104_$modout"
         *copymark elems 1 "102_$modin"
      }
   }

   set err 1
}
::mysql::close $db
if {$err == 1} {
   #-------equivalence each module seperatly---------
   set equivSize [expr {0.3 - $meshSize * 0.5}]
   set compNames [hm_complist name]
   foreach mod $modList {
      *createmark components 1
      foreach cn $compNames {
         if {[string first $mod $cn] > 0} {
            *appendmark components 1 $cn
         }
      }
      *equivalence components 1 $equivSize 1 0 0
   }
   set err 2
}

if {$err == 2} {
   foreach mod $modList {
      *collectorcreateonly components "fluid$mod" "" 24 ;# 24 == dark
   blue
      *createmark components 1
      *createmark components 2
      foreach cn $compNames {
         set tempString [string range $cn 4 end]
         if {![string compare $mod $tempString]} {
            *appendmark components 1 $cn
         }
      }
      *tetramesh components 1 components 2 1.2 0.75 97
   }
   set err 3
}
if {$err == 3} {
   foreach comp $compNames {
```

```
      set index1 [string first "101_" $comp]
      set index2 [string first "102_" $comp]
      set index3 [string first "103_" $comp]
      set index4 [string first "104_" $comp]
      set index5 [string first "105_" $comp]
      if {$index1==0} {
         set tempString [string range $comp 4 end]
         *renamecollector components $comp "inflow$tempString"
      }
      if {$index2==0} {
         set tempString [string range $comp 4 end]
         *renamecollector components $comp "inflow$tempString"
      }
      if {$index3==0} {
         set tempString [string range $comp 4 end]
         *renamecollector components $comp "wall$tempString"
      }
      if {$index4==0} {
         set tempString [string range $comp 4 end]
         *renamecollector components $comp "outflow$tempString"
      }
      if {$index5==0} {
         set tempString [string range $comp 4 end]
         *renamecollector components $comp "outflow$tempString"
      }
   }
   *feoutput "C:/Altair/hw8.0sr1/templates/feoutput/cfd/general"
   "C:/A#/Thesis/engine.nas" 1 0 1
   set err 4
}
puts $outfile "Success: $err"
close $outfile
```