2005-06-30

Attribution Standardization for Integrated Concurrent Engineering

Tyson J. Baker
Brigham Young University - Provo

ATTRIBUTION STANDARDIZATION FOR INTEGRATED CONCURRENT ENGINEERING

By

Tyson Baker

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 2005

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Tyson Baker

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

_____	_____
Date	C. Greg Jensen, Chair
_____	_____
Date	W. Edward Red
_____	_____
Date	Jonathan D. Blotter

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Tyson Baker 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 material 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

ATTRIBUTION STANDARDIZATION FOR INTEGRATED CONCURRENT ENGINEERING

Tyson Baker

Department of Mechanical Engineering

Master of Science

Product design is a creative process, often subject to rapid and numerous design change requirements. To facilitate geometric redesign iterations, Parametric Computer-Aided Design (CAD) systems were introduced. To manage the numerous product design iterations produced by parametric CAD systems, Product Data Management (PDM) systems were developed to capture, document, and manage each product revision. PDM has proved effective thus far at managing design history. However, PDM is built upon database management systems (DBMS), which have the capability of doing far more than simply managing product revision history.

Product data consists not only of the physical geometry used to describe it, but also of a host of non-geometric data. This non-geometric data is referred to as attributes.

Examples of attributes include material properties, boundary conditions, finite element mesh information, manufacturing operations, assembly operations, cost, etc. Downstream Computer-Aided Engineering (CAE) applications apply attributes to (preprocess) the geometry to perform their respective operations. These attributes are not permanently associated with the geometry and may have to be recreated each time the geometry changes. Preprocessing for highly complex CAE analyses can sometimes require weeks of effort. An attribution method is presented which addresses the creation, storage, and management issues facing attributes in the CAD and CAE environments.

The research conducted explores the use of database management systems for defining, instantiating, and managing attributes in the CAD environment. Downstream CAE applications may then retrieve the attributes from the DBMS to automate preprocessing. The attribution system results in standardized attribute definitions, which forms the basis for communicating attributes universally among different downstream CAE applications.

ACKNOWLEDGMENTS

I express sincere gratitude to all those who contributed to this thesis. First and foremost, to my wife, Emily, who encouraged and supported me. To Dr. Jensen, for his mentoring, insight, and enthusiasm. To my friends in the ParaCAD research group. To Jon Lund, for helping me with MySQL. To Luke Fife, for help with TCL. To Matt King and Brady Larson, for their pearls of programming wisdom. To Chris Dye, for being my personal bike consultant. And finally, to the Thursday night ParaCAD basketball league, for all the great memories.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Global competition and escalating market efficiency are driving engineering businesses to adopt design philosophies which reduce product design cycle time. One such design philosophy is "Concurrent Engineering." Concurrent Engineering (CE) is "the systematic approach to the simultaneous, integrated design of products and their related processes, such as manufacturing, testing and support" (Gallagher et al., 1998). Conceptually, CE is intended to shorten the design cycle by allowing engineering processes to run in parallel, rather than in series.

The product data model is central to CE and can be defined as the sum of all the information needed to define a product, consisting of both geometric and non-geometric information. In order for CE to be possible, the product data model needs to be established at the beginning of the design cycle. Geometric and limited non-geometric information are defined with Computer-Aided Design (CAD) tools. Geometry defined in CAD can then be exported to downstream engineering applications, where additional non-geometric information, called attributes, are instantiated and attached to the geometry before other engineering tasks are performed. The process of mapping attributes onto geometry is often referred to as preprocessing. Preprocessing is required for Computer-Aided Engineering (CAE) applications such as Finite Element Analysis (FEA), Computational Fluid Dynamics (CFD), Computer-Aided Manufacturing (CAM),

Computer-Aided Process Planning (CAPP), Rapid Prototyping (RP), etc. This research develops an approach to facilitate CE through CAD-centric attribute application (preprocessing).

1.1 Problem Statement

Considerable effort has been and is still being expended by the major CAD vendors to develop software capable of producing parametric, feature-based geometry. The objective of parametric modeling is to create geometry that can be easily morphed to a variety of shapes and sizes without having to start over each time a design change is recommended. Thus, parametric solid modelers are capable of creating reusable designs. However, in order to get the CAD model into downstream CAE applications, the geometry typically gets translated into a neutral file format that the CAE application can process, such as IGES, STEP, DXF, etc. These neutral file formats describe the geometry in terms of vertices, lines, areas, and volumes, and some contain sections where additional information about these objects can be written, such as the parametric relationships that exist among the objects. However, the current generation of CAE applications does not have the modeling capabilities necessary to control the geometric objects based on the parametric rules written in those sections. Therefore, engineers currently enjoy parametric solid modeling tools, but remain attached to non-parametric CAE tools. This means that when a design change is recommended, it is a relatively simple task to open the existing CAD file and update geometry accordingly, but the CAE preprocessing must be performed anew, which can be a very time consuming process. If the preprocessing for CAE applications could be performed in the CAD environment, the prospects for CAE automation and multi-disciplinary optimization (MDO) become more plausible.

The CAD environment is an ideal place to perform CAE preprocessing as it takes place early in the design cycle, has powerful geometry manipulation capability, and contains much of the original designer intent. If attributes could be associated with parametric CAD geometry, then CAE preprocessing essentially becomes parametric as well. For example, if the boundary conditions, meshing information, and material properties required by a FEA program could be mapped onto parametric geometric objects in the CAD environment, then every time the geometry gets passed into the FEA application the associated attributes would persist with the geometry. Thus, CAE preprocessing would only need to be performed once per parametric product geometry definition. The result of this is that by the time a product is imported into a CAE application, a solution can be obtained with a single operation. In addition, because the attributes necessary for all desired CAE applications can be defined upstream in CAD, a host of other independent CAE applications could function simultaneously, which is the goal of CE.

In order for the attributes to be recognized by downstream CAE applications, standards for attribute naming and management must be created. In addition, to accommodate for changes in technology, the library of attribute definitions must be customizable, allowing users to create new attribute definitions dynamically. These two ideologies, standardization and customization, are contradictory by nature. For example, once a user defines a new attribute, it is not automatically recognized by the downstream application or even by other users, and thus, the standard set of recognizable attributes has changed.

1.2 Thesis Objective

The objective of this research is to establish a method for applying attributes in the CAD environment in a standardized yet customizable manner, enabling automation of downstream CAE applications. The following questions are addressed in this research:

1. Can a CAD-centric attribution tool be developed that achieves a level of attribute definition standardization, but also allows users to dynamically create new attribute definitions?
2. Can the attribute information be stored and managed externally from the CAD file in such a manner that allows for seamless integration with both CAD and CAE applications?
3. Can this process be demonstrated on commercially available CAD and CAE applications?

1.3 Delimitation of the Problem

Due to the ever-expanding suite of CAE tools, this research does not attempt to create an attribute reader tool that will work for every CAE application available. Such a task should be left to downstream application developers. However, to demonstrate the principle, an attribute reader for a single CAE application, ANSYS, is developed. It would be possible to create an attribution tool that works independent of choice of CAD software. However, the scope of this research is to explore methods of attributing information in a customizable fashion and storing the information in an efficient database, not to create a CAD-independent attribution tool. Thus, the attribution tool will be created for use with a single CAD program, the UGS incorporated graphics program, Unigraphics.

This will provide a road map for CAD-centric attribution that can then be applied on other CAD systems.

Much research has been performed investigating methods of handling CAD object name changes under topology changing operations. Entity names are a type of product attribute, so the management of such names is related to the context of this research. Marcheix and Pierra (2002) present a survey of the existing approaches on persistent naming in parametric design and propose two criteria for persistent naming approaches. Such work is extensive and quickly diverges from the focus of this research. For this work, a simplified approach is adopted in that certain attributes will be lost after topology changes and therefore must be interactively reapplied by the user. To learn more about other techniques for handling the persistent naming problem, readers are encouraged to review Marcheix's work.

It is logical to assume that if one can create and modify attributes stored in an external database via the CAD environment, the same functionality could be developed for downstream CAE applications. That is, to create a bi-directional link allowing attributes to be created and modified from either the CAD environment or the downstream CAE application environment. By design, attributes are naturally created and maintained within downstream applications. However, this research will not be concerned with creating a bi-directional attribute link between the downstream application and the external database, although such a capability would inevitably result from an extension of this work. The reason for the omission of such a link is that the focus of this work is to explore methods to automate downstream CAE applications. Creating and modifying attributes in downstream applications impedes the automation process. If one were to

imagine all the product information flowing from the CAD environment to downstream applications, then using downstream applications to create attributes could be compared to swimming upstream.

CHAPTER 2: LITERATURE REVIEW

Reviewed herein are research publications that are most relevant to this research. The questions posed in chapter one lead to a review of the following topics:

- Concurrent engineering (Section 2.1)
- CAD Systems (Section 2.2)
- Computer Aided Engineering (Section 2.3)
- API Programming (Section 2.4)

2.1 Concurrent Engineering

Concurrent engineering has received considerable research attention over the past two decades as industry strives to shorten design cycles. As researchers have sought to understand and extend concurrent engineering, it has presently come to be thought of as an umbrella spanning all of the processes related to a particular product, beginning with initial concept generation, to detailed design and analysis, process planning, manufacturing, and so on. Concurrent engineering is not a new concept. It gained popularity in industry briefly after World War II, but, due to technology limitations, was later replaced by a departmentalization scheme, often referred to as the "over the wall" approach (Prasad 1996). This created a sequential design scheme that lengthened design time.

Gallagher et al. (1997) discuss the importance of establishing a complete product data model in order to integrate the processes of CE. They use the Pro/ENGINEER application programming interface (API) to extract feature data from the CAD database and use that information in a cost analysis program. While their work focused on integrating CE processes, their scope was limited to operating only upon the geometric entities created by CAD without concern for preprocessing for downstream applications. The research presented herein builds upon Gallagher's methodology of defining the complete product data model within the CAD environment by extending the ability of the CAD program to define the non-geometric information required by downstream CAE applications.

Jiang et al. (2002) present an integrated concurrent engineering approach to the design of complex components. As a case study, they attempt to integrate the solid modeling of a scroll compressor with associated downstream applications such as FEA, CAM, and CAPP. Their work relies heavily on the Pro/ENGINEER suite of integrated software such as Pro/MECHANICA, Pro/MANUFACTURE/, Pro/CMM, etc. to automate downstream processes. There are inherent advantages and disadvantages to using a set of integrated software from a particular vendor. Obviously, the advantages are seamless integration with downstream applications and conservation of product data model information. The disadvantages are that the designer must use the downstream applications provided by the vendor, and thus lose the ability to choose a preferred downstream application. This research has the same objective as Jiang's, that is, to automate downstream applications; however it seeks a more robust approach that is not limited to a particular suite of interrelated software.

Ma and Tong (2003) present a modeling technique based on what they call "associative features." They use the CAD API to create a knowledge-oriented tool. As a case study, they illustrate their techniques on the design of plastic injection molds. They propose that CE becomes realistic through use of such modeling techniques as "associative features." This research follows a similar approach to realizing CE, which is, using the CAD API to automate redundant operations; however the objective of this research focuses on CAD-centric attribution whereas Ma and Tong limited their scope to defining new types of CAD features.

Bailey et al. (1999) describe an environment based on the concept of an Intelligent Master Model (IMM) to foster concurrent engineering. They describe work done at General Electric Aircraft Engines where changes in one discipline of engine design are communicated to the IMM, which in turn, communicates those changes to the rest of the associated disciplines. For example, if the engine designer changed the flow rate through the engine in one program, that change would go through the IMM and the engine geometry would automatically update to compensate for the change. Bailey's work spoke in general terms about how the IMM can be used to reduce design cycles, however they did not demonstrate or prove their methods in their publication. Hoffman et al. (1998) built upon Bailey's concept of the IMM and developed an architecture to manage a product master model. In their view, the CAD model and all downstream applications are clients of the master model, which is essentially a database containing all of the product information. In their work, all the attributes associated with the product data model were defined in the respective downstream application where they had scope. This research

differs in that all the desired attributes are assigned initially in the CAD environment, and stored in an external database for application in downstream applications.

Concurrent engineering is a broad paradigm, allowing researchers an array of possible methods to further its application in industry. While research in concurrent engineering is being conducted in many different fields, the research presented in this section focuses on the methods which are the most relevant to the methods used in this thesis, that of a CAD-centric approach to concurrent engineering. The motivation for establishing a CAD-centric approach to concurrent engineering is due to the ability for CAD systems to not only store geometric information, but to allow that information to be passed and interact with other applications. Custom applications designed for use within the CAD environment allow the CAD software to be extended from simply a geometry definition application to a tool driving parallel processing downstream CAE applications.

2.2 CAD Systems

While CAD began and was initially known as Computer-Aided Drafting in the 1960's, it rapidly evolved into a powerful engineering tool (Sutherland 1963). Advances in computer hardware and software induced an evolution of CAD from 2D electronic drafting tools, to wireframe modeling systems, to surface modelers, and finally to the current generation of solid modelers. Today's high-level CAD packages are capable of producing parametric, feature-based solid geometry. A review of CAD systems presents the basis upon which the attribution tool developed in this research is added.

Wireframe modelers depict 3D geometry via creation of a series of vertices and connecting lines in 3D coordinates. They were primarily used as 3D visualization tools and generated initial enthusiasm for the possibilities of CAD. Wireframe modelers do not

contain any mathematical information about the surfaces of the model, nor can they adequately describe the interior of solids. Such inadequacies led to the next logical step in CAD evolution, surface and solid modeling systems (Lee 1999).

Surface modeling systems built upon wireframe modelers by including mathematical information about the surfaces created by the lines and points. Surface modelers generate surfaces by interpolation of points or interpolation of curves through sweeping operations. In addition, information about the connectivity of one surface to another could also be maintained by surface modelers. In this manner, complex surface geometry could be generated (McMahon and Browne 1998).

Solid modeling systems allow for creation of a closed body by recognizing the volume created by a series of connected points, lines, and surfaces. Mathematical information stored in the CAD database allows the modeler to recognize the volume represented by the inside of the solid as opposed to the volume outside. Today's CAD systems are combination solid/surface modelers. In the initial stages of development of solid modelers, two ideologies emerged for creating solids: Constructive Solid Geometry (CSG) and Boundary Representation (B-rep). Most of today's CAD systems are hybrid CSG/B-rep (Anderl and Mendgen 1996).

The idea behind CSG is to create complex geometry by performing Boolean operations on a series of simpler volumes, called primitives, such as blocks, cylinders, cones, spheres, etc. Figure 2.1 (King 2004) shows examples of Boolean operations typical of CSG.

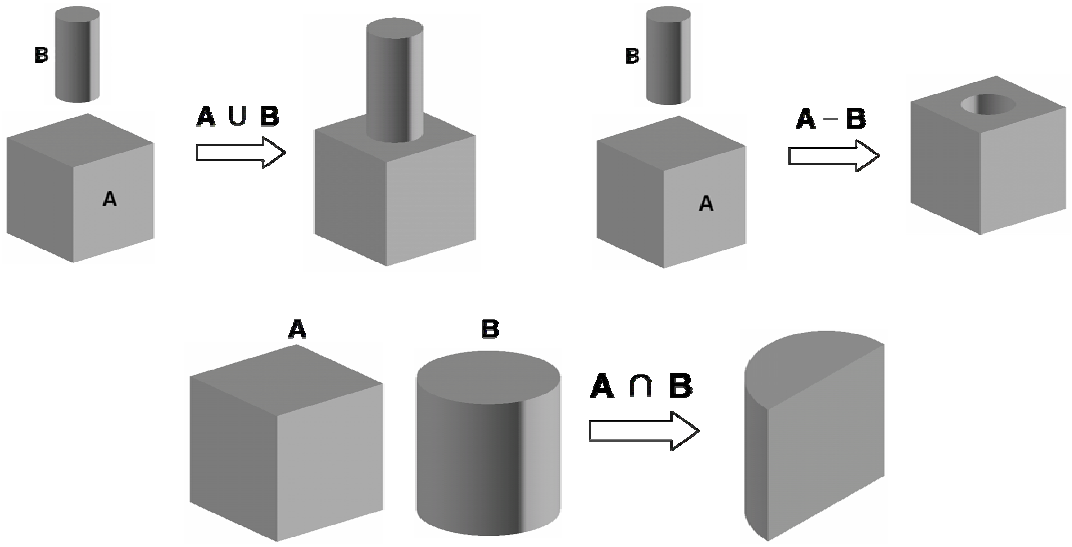


Figure 2.1 Boolean Operations for CSG Primitives

The primitives are created when the user inputs a required number of geometric parameters. The creation order and relationship between primitives and their respective Boolean operations is maintained in a CSG tree (Shapiro and Vossler 1995). Figure 2.2 (Wilson 2004) demonstrates a typical CSG tree.

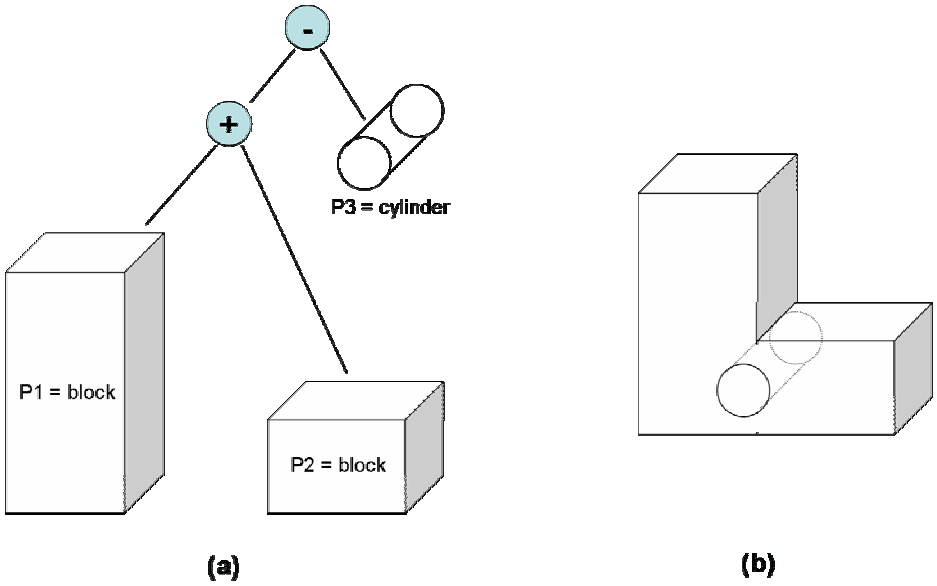


Figure 2.2 CSG Tree

B-rep systems operate by storing topological information about the solid, i.e. vertices, edges, faces (Lee 1999). By handling topology in such a manner, B-rep systems are able to perform more complex freeform surface operations than CSG systems. Figure 2.3 (Wilson 2004) shows a B-rep scheme for a simple solid.

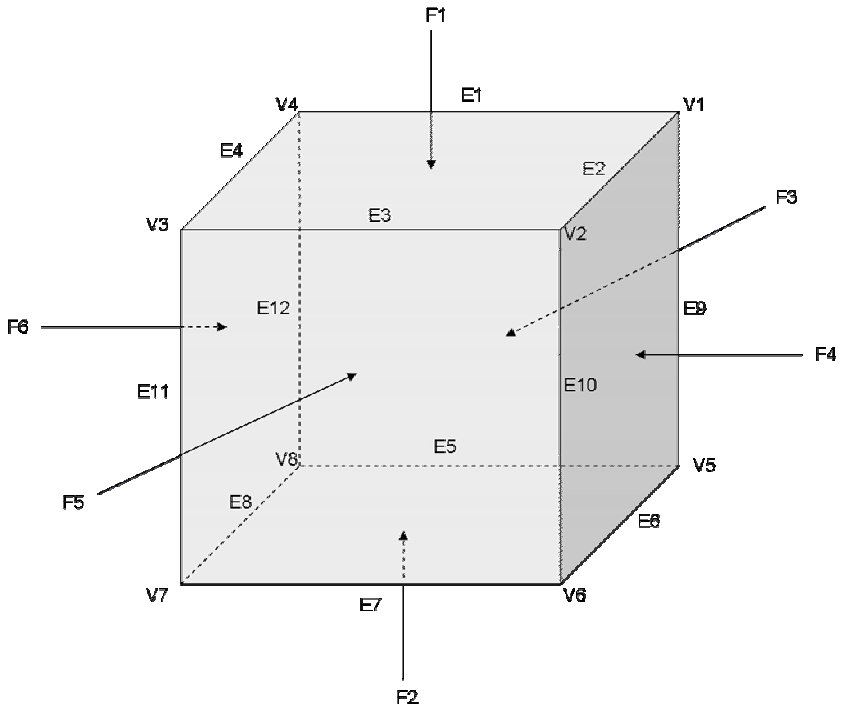


Figure 2.3 B-rep scheme for 3D volume

Many current commercial CAD vendors include elements of both CSG and B-rep modeling in their systems.

Parametric solid modeling was a paradigm first introduced by Parametric Technology Corporation's (PTC) Pro/ENGINEER in the 1980's and has since been adopted by all the high level CAD packages (Hoffman and Kim 2001). The parametric modeling paradigm uses geometric constraints, dimensions, and relationships to produce a system of equations. Parametric modelers use an iterative, sequential approach to arrive at

a solution that satisfies the applied geometric constraints (Ault 1999). Only uncoupled equations are allowed by parametric modelers and the entire set of equations must be satisfied before any subsequent operations may be performed. Variational modelers, on the other hand, follow a similar paradigm as parametric modelers, but can solve coupled equations and solve the set of equations simultaneously. Variational modelers may produce undesirable results, but allow for creation of any possible geometry. Some CAD programs are combination parametric/variational systems.

Feature-based modeling has gained prominence in commercial CAD software in concurrence with parametric modeling. Similar to the CSG tree, in feature-based modeling the part creation history is maintained in a feature tree. Features are named, typed geometric information structures (Shapiro & Vossler 1995). Examples of features include the sketch, boss, fillet, blend, revolve, extrude, slot, hole, etc. When a model update is necessary, the update begins with the first feature in the feature tree and progresses sequentially down to the last.

The review of CAD systems provided herein provides the basis on which an attribution tool will be added. The developed attribution tool communicates with the CAD system in order to gather and store information about geometric objects (vertices, lines, surfaces, volumes) and/or features of digital product geometry. An understanding of how CAD systems handle geometric objects and features is essential for the developed attribution tool to be able to communicate with the CAD environment, and associate attributes with CAD geometry. After a CAD-centric attribution foundation is established, the next step in the development of the developed attribution tool is the establishment of a connection between the attribution tool and downstream CAE applications.

2.3 Computer-Aided Engineering

Computer-Aided Engineering applications are a key component in product design, giving businesses the ability to test products without developing costly prototypes. Due to the complexity of many CAE applications, considerable effort is being expended to automate CAE, allowing engineers to analyze more designs in a shorter period of time. The lines between CAD and CAE applications are becoming increasingly blurred, as many CAD vendors now include CAE applications embedded within the overall package. CAE developers, in turn, are developing their systems to allow direct communication with CAD files from the major CAD packages, rather than converting the CAD files into a neutral file format. Such steps have greatly increased the ability of engineers to use CAE applications in preliminary design. However, given a glimpse of the power of CAE applications in preliminary design, researchers are now seeking to find new ways to allow CAE analysis to be a part of design optimization. Optimization algorithms may need to analyze thousands of design iterations before finding an optimal solution. For this reason, it is essential to automate CAE preprocessing, in addition to the automation of CAD-CAE integration that has already occurred between CAD and CAE developers.

CAE applications have become increasing widespread as computational resources have grown over the past three decades. CAE first gained popularity in the 1960's in structural engineering and has since expanded to include heat transfer, computational fluid dynamics (CFD), and vibration analysis. When properly applied, CAE applications can provide highly accurate predictions of a system's response to a particular input.

Most computer-aided analysis programs utilize the finite element method, or finite element analysis (FEA), to approximate a solution. The finite element method operates by

discretizing complex geometries with simple shapes, such as squares or triangles in the 2D case, or cubes and pyramids respectively for 3D geometry. The reason for discretization into simple shapes is that the complex equations for structural, thermal, fluid, and vibrational analysis can more easily be applied to simple geometry. Because these simple shapes can only approximate the original geometry, the solution to finite element problems is always an approximation. Thus, considerable effort is expended by FEA analysts to produce a high quality "mesh" of the original geometry, as it will affect the accuracy of the solution.

2.3.1 CAE Geometry Import File Formats

In order for CAE tools to perform their tasks, geometry must either be created or imported from a CAD model. Most CAE applications have modeling capabilities that allow them to create basic geometry; however, for more complex geometry it is more practical to model the geometry in CAD and export it to the CAE application in a geometry neutral file format. Some of the most popular neutral file formats include DXF, Parasolid, IGES, and STEP. When CAD geometry is imported using these neutral formats, the parametrics governing the rules and relationships in the model are lost. This research seeks to eliminate this problem with the development of an attribution tool.

2.3.2 CAD-CAE Integration

Generally speaking, the CAD to analysis step follows an "over-the-wall" approach. For example, after a CAD designer has finished modeling a part, he or she may pass the model "over-the-wall" to a structural, thermal, or aero analyst (or even to all three) who will create a finite element mesh of the model for analysis. After reviewing the analysis results, the analyst may suggest changes in the model geometry, thus

throwing the model back "over-the-wall" to the designer. Significant effort has been spent exploring integration methods for CAD and analysis software to reduce the time required for such an iterative process.

Automating the grid generation process for CAD-CAE integration is not a trivial task. Samareh (1999) recognizes grid generation for analysis as "…the most labor-intensive and time-consuming part of the computational aerosciences for analysis, design, and optimization." Research in CAD-CAE integration represents one of the most challenging and important aspects of the overall CE integration problem. This research includes integration of analysis tools with the product data model along with other types of downstream applications with the vision of achieving CE. Referring to the role of integrating analysis tools and CAD with respect to overall integration of the product data model, Arabshahi et al. (1993) state, "…it is clear that the analysis related tools will simply be part of a much broader based product information environment."

Gabbert and Wehner (1998) discuss CAD-FEA integration through managing the product data model in an object-oriented data structure and passing that data to FEA via the STEP (Standard for the Exchange of Product Model Data) exchange format. They attempted to partially define the product data model by using object-oriented data storage methods in connection with the STEP data exchange format. Agonafer et al. (1997) accomplished preprocessing for CFD in a commercial CAD program, CATIA. They assigned all of the necessary CFD attributes in the CAD environment and then imported the attribute information into the built-in CATIA CFD tool for automated meshing and analysis. Such a method enables automation of airsolid meshing and CFD analysis, but,

again, can only operate within the integrated set of CAD-analysis software of a particular vendor.

Samareh (1999) suggests that in an ideal design environment, a designer could use a parametric model to easily explore alternative designs and optimize the design based on a set of desired objectives and constraints. In order to integrate and streamline multi-disciplinary analysis and optimization of a parametric model, a fully automated, push-button grid generation tool must be developed. Samareh (1999) pointed out that for an automated MDO environment to function, a parametric CAD model must exist because the optimization routine will cause the model to morph into different shapes, which would exploit any weakness in the model's parameterization scheme causing the model to fail to regenerate. Hogge (2002) explored MDO of a parametrically defined turbine blade. For his work, he created a parametric turbine blade and automated the FEA and CFD analyses within an optimization algorithm. In order to perform these analyses in an optimization loop, he had to devise a method to automate the analyses and feed the results back to the optimization algorithm. His method for automating the CAE applications is of particular interest to this research. Although effective in its results, Hogge's automation method was very labor intensive and required highly technical knowledge about how the downstream applications communicated with the CAD model. This research builds upon Hogge's work in that it facilitates the MDO process by creating a more intelligent link between the CAD tool and downstream applications via CAD-centric attribution.

2.3.3 Use of CAD-centric Attributes for CAE Automation

Researchers have recognized the need to apply non-geometric information to the CAD model to automate CAE applications. The ability to apply attributes in the CAD

environment provides the most promising solution to CE integration. For his work in CAD-FEA automation, Shepard (1988) defined the physical, non-geometric information as attributes and declared the necessity to define attributes with respect to the product geometry. Along the same lines, Arabshahi et al. (1991) was able to apply attributes in the CAD environment through a process he called "attribution." Later, Arashahi et al. (1993) stated, "the ability to apply attributes, such as material properties, analysis type, loads, etc, to the solid model is one of the most fundamental steps along the road to integration of CAD and FEA." They developed an attribution tool that would map FEA specific attributes onto the solid model. Around the same time, other researchers were investigating the implications of being able to apply attributes to the CAD model. Stroud (1993) categorized the different types of information, both geometric and non-geometric, that can be handled in solid modeling. He did not present a strategy for handling such information, but his objective was to call attention to application developers to implement strategies for handling non-geometric information. Subrahmanyam et al. (1995) presented a conceptual framework for attribute-based mechanisms and outlined a basic taxonomy of attributes. They also discuss attribute behavior under geometric transformations, shape-changing and topology-changing operations. They describe attributes as, "ideal candidates to store information relevant to a particular phase in the life-cycle of a product. For example: design, analysis, assembly, process planning, etc." In more recent efforts, Shepard (2000) addressed strategies for handling attributes when entities in the solid model are created, deleted, or modified. O'Bara et al. (2002) implemented an object-oriented, CAD-centric attribute management system for CAE automation. He suggests that in order for CAE applications to reach their potential, an environment that supports

geometry-based specification of analysis attributes must exist. His attribute management tool allowed the end user to associate geometric objects with attribute objects, which are C++ classes representing a specific type of attribute information structure. The attributes associated with the geometry were then stored or retrieved in terms of file I/O. O'Bara's work is the most relevant to this research, however, important differences exist that require additional research. The attribute types defined by O'Bara were hard-coded into the C++ workspace, and thus the library of attribute types could not be easily edited, while this research aims to make the attribute language user-customizable.

This research builds upon the efforts made by the aforementioned researchers by creating a CAD-centric attribution tool to foster CE. However, the philosophy of how the attributes are to be handled is fundamentally different. Previous efforts in CAD-CAE integration developed attribution tools that catered to the information necessary for a single CAE discipline, such as structural analysis, for example. In order for the parallel-process paradigm of CE to operate, an attribution tool that can apply attributes to any discipline, not just FEA or CFD, is demonstrated in this research, creating a universal attribution tool for all disciplines and establishing a standard attribute language that can be interpreted by downstream CAE applications.

2.4 API Programming

CAD and CAE developers, recognizing that users would want to create custom applications for their programs, built their applications such that users can access a library of programming functions which can be used to control the application from a separate software program. An Application Programming Interface, or API, is the library of functions allowing users to programmatically control of the functionality of a particular

software program. The development of API's is especially important to the attribution tool created in this research. The API's of the CAD and CAE applications allow the attribution tool to act as a bridge between the CAE and CAD applications. Significant research has been performed involving the CAD API.

Hogge (2002) classified API's into three categories: macros, program-specific languages, and high-level programming language with program specific functions. For example, the Unigraphics API, called UG/Open, is written in C, thus falling under the third category. ANSYS, on the other hand, has created its own program-specific API language, called the ANSYS Parametric Design Language (APDL), and falls under the second API classification. Development of a CAD-centric attribution tool requires the use of Unigraphic's API. In addition, use of the ANSYS API is necessary to interpret the attributes applied in CAD for this research. Delap (2003) cites some of the features of an effective CAD API from Hoschek and Dankwort (1994):

- Access to geometric elements in the CAD data structure
- Support of program control mechanisms to effectively support non-linear design steps
- Trace function for debugging, since the programs are getting large in industrial applications
- Method for error handling in order to recover from user errors when using the parametric model
- Possibility to request user input in order to program the interaction dialog for using the model to generate variants
- Write and read data from file, e.g. to read predefined sets of dimension values

API programming can replace interactive operations performed in the software's graphical user interface (GUI). A well-designed API will allow access to all the same functionality found in the GUI. API programs can be called interactively from within the software, or externally for batch mode operation.

Significant research has been performed exploring the use of programmatic CAD API's in their work. Ardelan (2000) used the SolidWorks API to develop a design "wizard" to guide the user through the steps for spacecraft component design. The users were prompted to enter design information into a series of GUI's that had design knowledge built into them. The user input was then used to size predefined components and add them automatically to an assembly. Tucker (2000) performed a study to explore the translation of API programs between different CAD systems. His method would read in a program in the API of one CAD system and output a program in the API of another.

There are inherent disadvantages to developing routines in API's. First, a CAD application developed in an API requires a considerable time investment. Also, API development is not intuitive and requires training and experience on the part of the developer. Rohm (2001) implemented a method that would eliminate these disadvantages by development of a system that would interpret user interactions in a graphical user interface and produce the API code to perform the same function.

Shaw and Rangel (2002) integrated CAD with CAM/CAPP using a commercial CAD API (I-DEAS). They employed user-defined attributes created with the I-DEAS API to help in feature recognition for generation of NC-toolpath generation. The API requested model data and machine features, and would then generate toolpaths from that information.

Delap (2003) used the Unigraphics API to create a custom CAD application that would generate a 2D gas turbine engine flowpath geometry based on text input from a thermodynamic cycle analysis software program. He was then able to perform optimization of the flowpath by running his custom application in batch mode.

Recent efforts have been made exploring the ability to create CAD API utilities that operate independent of choice of CAD software. Astle (2003) integrated a library of CAD-independent curve and surface functions with his custom flank-milling program. His work demonstrated the ability to separate CAD API dependent operations with data processing functions. In a similar fashion, Wilson (2004) used the Unigraphics API to create a custom rapid-prototyping slicing application. Wilson built upon Astle's notion of separation of data processing and CAD API functions. Although he demonstrated his software only in Unigraphics, the program architecture was developed so that the data processing algorithms could be reused for different CAD programs.

Without the existence of API's, none of the aforementioned research would have been possible. This research uses the CAD API extensively to develop a method for performing CAE preprocessing in the CAD environment. After the CAE preprocessing has been performed a single time within the CAD environment, it becomes possible to perform design optimization by iterating on geometric variations and quickly performing CAE analysis for each iteration. The developed attribution tool is intended to provide a road-map for performing CAD-centric CAE attribution for any CAD and CAE package with its respective API.

CHAPTER 3: BACKGROUND

An overview of the background for the specific topics used in this research is presented in this chapter. The following topics are addressed:

- Mathematics of computer-aided geometric design
- The finite element method
- Object-oriented programming
- Database Management Systems

An understanding of the mathematics and algorithms used in the topics presented in this chapter will assist the reader in understanding the methods used to achieve the research objectives. However, if the reader already posses sufficient understanding of the aforementioned topics, he or she is invited to proceed to chapter four, wherein the research method will be described.

3.1 Mathematics of Computer-Aided Geometric Design

Reviewed herein are some of the underlying principles relative to solid geometry and freeform surfaces. An understanding of the principles and methods used in the creation of solids and freeform surfaces is necessary when developing CAD API programs.

3.1.1 Solid Modeling

A solid model is a mathematically defined as a point set S in 3D Euclidian space (R^3). Referring to the interior and boundary of the set by iS and bS, respectively, the solid S can be defined as:

$$S = iS \cup bS \qquad \text{Equation 3.1}$$

Given the compliment of S, cS, then

$$W = iS \cup bS \cup cS \qquad \text{Equation 3.2}$$

where W includes all possible points in R^3. Figure 3.1 graphically depicts this relationship (Zeid 2005).

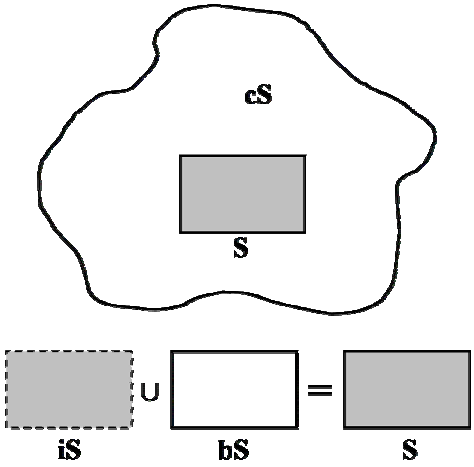


Figure 3.1 Mathematical definition of a solid

Mathematically, solids should possess the following properties:

- Rigidity. This means that the solid's shape is invariant and independent of the model's location or orientation in Euclidean space.
- Homogeneous three-dimensionality. Boundaries of solids must maintain contact with the interior. No isolated or dangling boundaries should exist.
- Finiteness and finite describability. This implies that the size of the solid is not infinite, and that a limited amount of information can describe the solid.
- Closure under rigid motion and regularized Boolean operations.
- Boundary determinism. The solid's boundary must contain the solid and determine the interior of the solid distinctively (Zeid 2005).

Because solid modelers contain information about the solid volume, CAE applications are able to discretize the volume into finite elements. There is an inherent relationship between the volume of a solid and attributes applied to that volume. For example, when applying a mesh element size it is important to know the volume of the solid. Thus, intelligent attributes can be applied to solids based on its mathematically defined physical parameters.

3.1.2 Bezier Curves

The Bezier curve is central to freeform curves and surfaces. Bezier curves are composed of at least two control points. The linear segments joining the control points form the control polygon, which is not visible when the curve is plotted. Bezier curves are classified according to the degree of the curve, $n = m–1$, where n is the degree and m is the number of control points. Figure 3.2 shows the control points and control polygon

of a degree three Bezier curve. If the control points of a Bezier curve are known, that curve can be exactly duplicated.

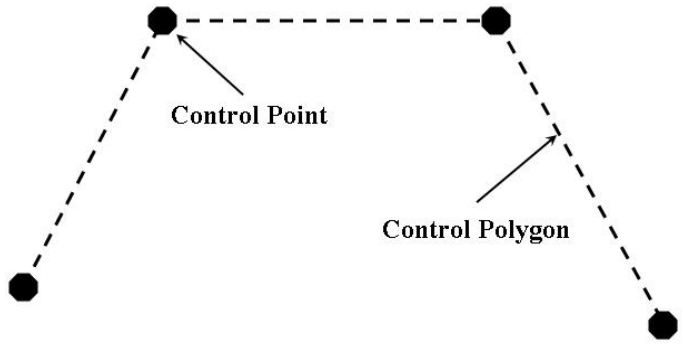


Figure 3.2 Control polygon for Bezier curve

Figure 3.3 shows examples of degree n = 1, 2, 3, and 4 Bezier curves respectively, along with their control points and control polygons.

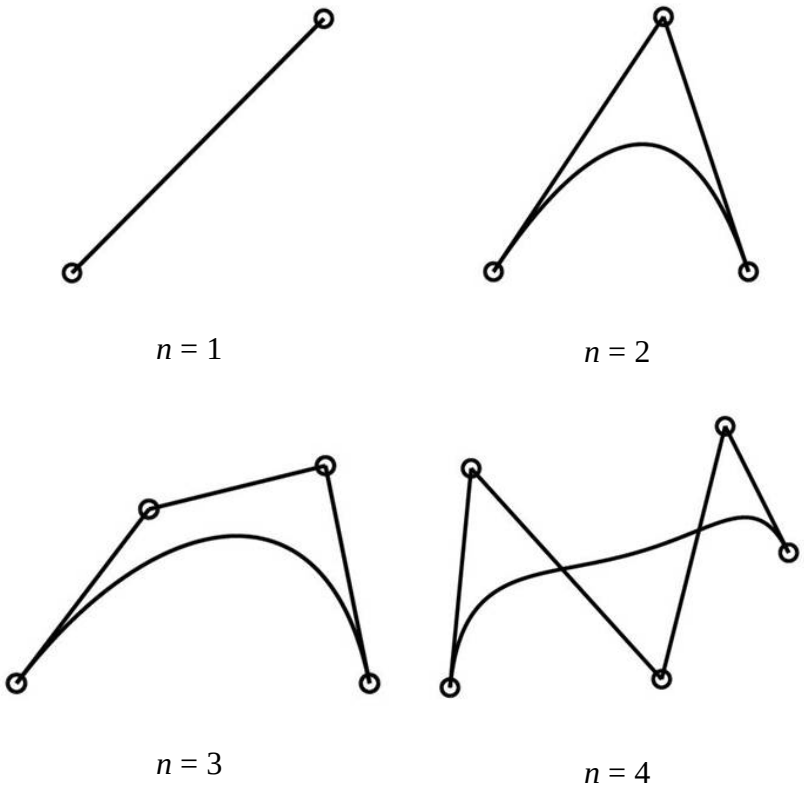


Figure 3.3 Degrees 1-4 Bezier curves

The general equation for a Bezier curve is defined as:

$$\overline{P}(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \overline{P}_i$$

where *n* is the curve degree, *t* is the location along the curve between 0 and 1, $\binom{n}{i} = \frac{n!}{i!(n-1)!}$, $\overline{P}(t)$ is a system of mapping functions from parameter space to cartesian space (X, Y, Z), $\vec{P}_i$ is an *n* x *m* matrix of control point coordinates, and $\binom{n}{i}(1-t)^{n-i} t^i$ forms what are known as Bernstein polynomials. Expansions of the Bezier curve equation for degree one, two, and three are:

$$\vec{P}(t) = (1-t)P_o + tP_1$$

$$\vec{P}(t) = (1-t)^2 P_o + 2t(1-t)P_1 + t^2 P_2$$

$$\vec{P}(t) = (1-t)^3 P_o + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

Among connecting Bezier curves, continuity constraints may exist. Two types of continuities can exist between Bezier curves, geometric (G^i) and parametric (C^i). Sederberg (2002) explains that curves which are parametrically continuous are also geometrically continuous to the same degree, but the reverse is not always true. Table 1 summarizes the meanings of each type of continuity (Delap 2003).

Table 3.1 Definition of Bezier continuities

Geometric Continuity		Parametric Continuity	
Degree	Description	Degree	Description
G^0	Curves share a common endpoint	C^0	Curves share a common endpoint
G^1	Curves are tangent	C^1	First derivative is identical
G^2	G^1 and have same curvature at endpoint	C^2	Second derivative is identical
G^n	Curves are G^n if the equations can be modified to be C^n	C^n	nth derivative is identical

Bezier curves form the building blocks for B-splines and Bezier surfaces. The integration of the mathematics of Bezier curves into modern CAD systems has greatly increased the ability of engineers to mathematically define and manipulate freeform surfaces. When applying certain attributes to freeform curves or surfaces, an understanding of Bezier curves and surfaces is helpful in knowing how to apply them. For example, a FEA mesh attribute could be created for Bezier curves that apply nodes at certain intervals along the curve. Because the governing equations for the curve are known, the designer can better decide how large the interval needs to be.

3.2 The Finite Element Method

One of the objectives of this research is to automate CAE applications via CAD-centric preprocessing. Thus, a brief overview of the finite element method is appropriate. Often times it is impossible to produce a solution to complex problems using analytical methods. For such problems, numerical techniques can be used. The finite element method operates by dividing a body or structure into smaller elements of finite dimensions called “finite elements.” The collection of the finite elements is often referred to as the

"mesh," or "grid." The original structure is then considered as a collection of these elements connected at a finite number of joints or "nodes." Figure 3.4 (King 2004) shows a 2D mesh of an area, and Figure 3.5 shows some of the commonly used elements in 3D FEA.

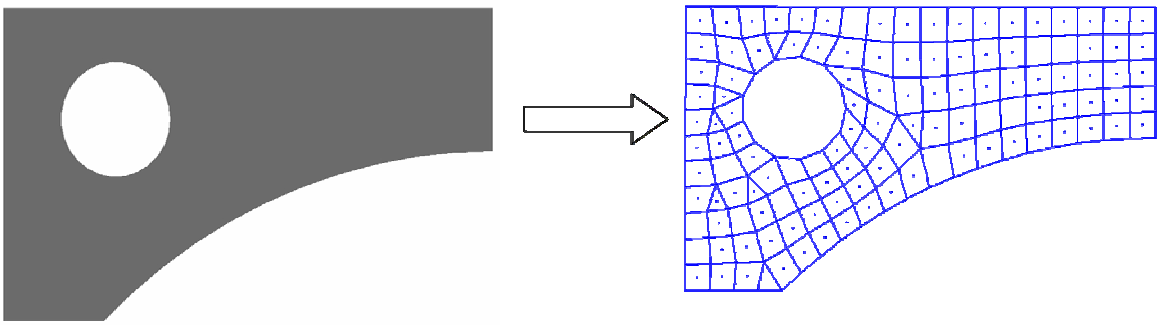


Figure 3.4 Mesh for 2-dimensional area

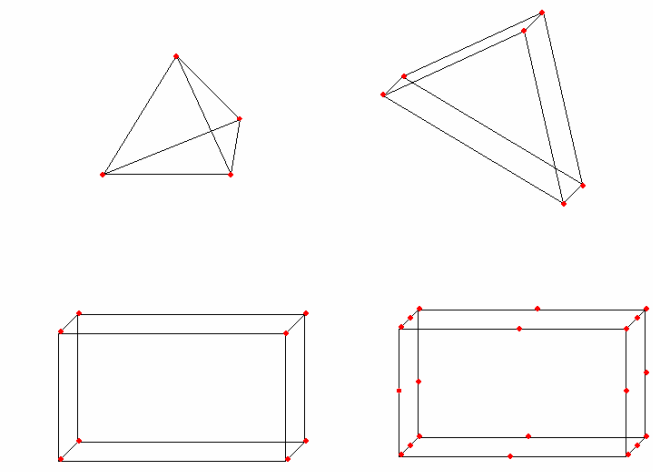


Figure 3.5 Common 3D FEA elements: 4, 6, 8, and 20 node elements

The equations of equilibrium for the entire structure or body can be obtained by combining the equilibrium equation at each element such that the continuity is ensured at each node. After the necessary boundary conditions are imposed, the equations of

equilibrium can be solved to obtain the desired variables throughout the body, such as stress, strain, flow velocity, or temperature distribution, depending upon the application.

3.2.1 Mesh Generation

Mesh generation for FEA presents a unique challenge to preprocessing. In order to get the most accurate solution, the mesh must be fine enough to adequately represent the geometry of interest. However, time constraints and computational resources limit the number of elements that can be used in the mesh. For example, consider the simple case of a hole in a plate as shown in Figure 3.6.

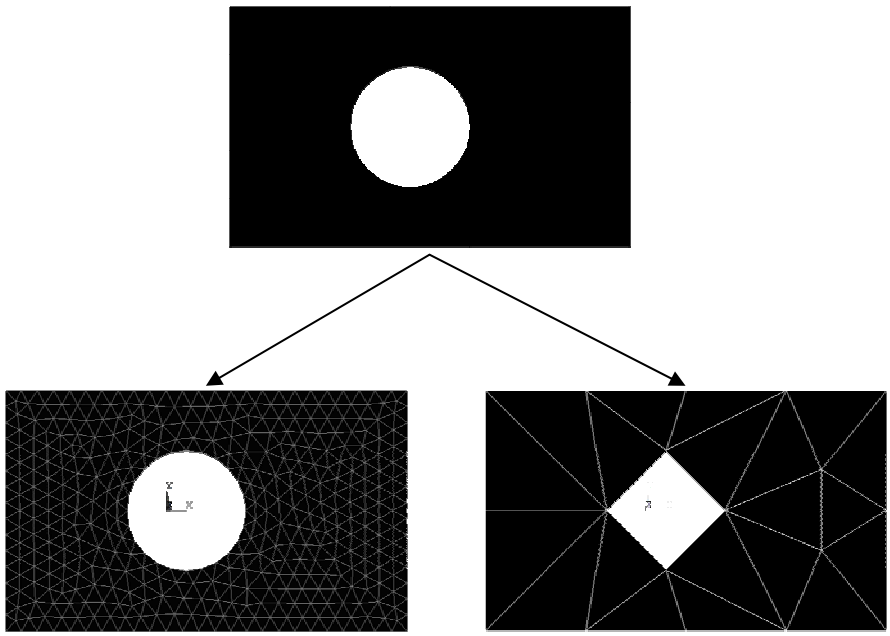


Figure 3.6 Effect of mesh density

If the mesh is too coarse, the hole in the plate will not be accurately represented, appearing as a polygon. With a finer grid, the hole is more accurately represented, but the element count has greatly increased, making the solution more computationally expensive.

Mesh generation algorithms can be divided into two groups: structured and unstructured. A structured mesh is one whose interior nodes have an equal number of adjacent elements. Unstructured mesh generation relaxes the node valence requirements, allowing any number of elements to meet at a single node.

Producing a mesh that is highly representative of the original geometry, yet computationally economical is an area where CAD-centric attributes are highly useful. Very specific information about geometric objects applied by the attribution tool could be used to guide the meshing software. After the geometry has been meshed, the next step is to define the boundary conditions and solve the problem according to the numerical approximations for the governing equations. The attribution tool in this research was designed to be able to apply attributes relative to structural FEA, thermal FEA, and CFD.

3.2.2 Numerical Analysis

The finite element method is able to solve problems for discretized geometry by approximating the governing differential equations as a set of linear algebraic equations. King (2004) summarizes Balling (2001) in giving a brief explanation of how these equations are developed. To demonstrate how the finite element method works, consider a thermal analysis for a solid slab of uniform thickness, t, shown in Figure 3.7.

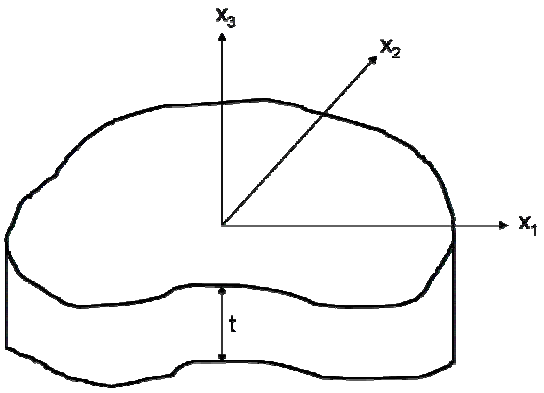


Figure 3.7 Thermal analysis, uniform thickness

To make the problem 2D, assume that all the heat flow is in the x_1 and x_2 directions. The governing differential equation for heat transfer in this example becomes:

$$k\nabla^2 T = kT,_{11} + kT,_{22} = -Q \qquad \text{Equation 3.3}$$

where T is temperature function, k is the coefficient of thermal conductivity, and *Q* is the heat radiation per unit volume. The conductive heat fluxes q_1 and q_2 in the x_1 and x_2 direction are derived from Fourier's Law of heat conduction:

$$q_1 = -kT,_1 ; \quad q_2 = -kT,_2 \qquad \text{Equation 3.4}$$

After the domain is discretized, to produce the finite the finite element form of the governing equation, we can use

$$T(x_1, x_2) \approx \underline{N}^{\mathrm{T}} \underline{U} \qquad \text{Equation 3.5}$$

where $\underline{N}$ is the vector of shape functions and $\underline{U}$ is the vector of approximations to the nodal temperatures. To construct the weighted integral, substitute equation 3.5 into equation 3.3 and integrate over the volume of the domain to obtain equation 3.6.

$$t \int_{area} \underline{N}\left(k\underline{N}^{\mathrm{T}},_{11} \underline{U} + k\underline{N}^{\mathrm{T}},_{22} \underline{U} + Q\right) dA = \underline{0} \qquad \text{Equation 3.6}$$

Applying Green's theorem (3.7) to the first two parts of equation 3.6 and invoking the finite element approximation (3.5) we obtain equation 3.8, which is the set of linear algebraic expressions:

$$\int_{area} f\, g_{,i}\, dA = \oint_{perimeter} f\, g\, n_i\, dP - \int_{area} f_{,i}\, g\, dA$$

Equation 3.7

$$t \oint_{perimeter} \underline{N}\left(q_1 n_1 + q_2 n_2\right) dP + t \int_{area} \left(\underline{N}_{,1}\, k\, \underline{N}^{\mathrm{T}}{}_{,1} + \underline{N}_{,2}\, k\, \underline{N}^{\mathrm{T}}{}_{,2}\right) dA\, \underline{U} = t \int_{area} \underline{N}\, Q\, dA$$

Equation 3.8

From equation 3.8 we define the system stiffness matrix $\underline{\underline{K}}$ and force vector $\underline{F}$:

$$\underline{\underline{K}} = t \int_{area} \left(\underline{N}_{,1}\, k\, \underline{N}^{\mathrm{T}}{}_{,1} + \underline{N}_{,2}\, k\, \underline{N}^{\mathrm{T}}{}_{,2}\right) dA = t \int_{area} \left[\underline{N}_{,1} \quad \underline{N}_{,2}\right] k \begin{bmatrix} \underline{N}^{\mathrm{T}}{}_{,1} \\ \underline{N}^{\mathrm{T}}{}_{,2} \end{bmatrix} dA$$
$$= t \int_{area} \left(\underline{N} \nabla^{\mathrm{T}}\right) k \left(\nabla \underline{N}^{\mathrm{T}}\right) dA$$

Equation 3.9

$$\underline{F} = t \int_{area} \underline{N} Q dA - t \oint_{perimeter} \underline{N}\left(q_1 n_1 + q_2 n_2\right) dP = t \int_{area} \underline{N} Q dA - t \oint_{perimeter} \underline{N} q_{out}\, dP$$

Equation 3.10

Equation 3.11 is obtained after substituting equations 3.9 and 3.10 into equation 3.8,

$$\underline{\underline{K}}\, \underline{U} = \underline{F}$$

Equation 3.11

Next the shape functions, N^T, are specified for an element. Shape functions are unique to the type of element used. Consider a triangular element as shown in Figure 3.8.

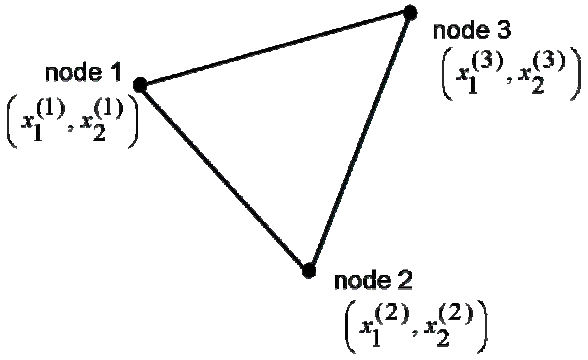


Figure 3.8 2D triangular element

The shape functions in terms of nodal coordinates are:

$$\underline{N} = \begin{bmatrix} \frac{1}{2A}\left(x_1^{(2)}x_2^{(3)} - x_1^{(3)}x_2^{(2)}\right) & \frac{1}{2A}\left(x_2^{(2)} - x_2^{(3)}\right) & \frac{1}{2A}\left(x_1^{(3)} - x_1^{(2)}\right) \\ \frac{1}{2A}\left(x_1^{(3)}x_2^{(1)} - x_1^{(1)}x_2^{(3)}\right) & \frac{1}{2A}\left(x_2^{(3)} - x_2^{(1)}\right) & \frac{1}{2A}\left(x_1^{(1)} - x_1^{(3)}\right) \\ \frac{1}{2A}\left(x_1^{(1)}x_2^{(2)} - x_1^{(2)}x_2^{(1)}\right) & \frac{1}{2A}\left(x_2^{(1)} - x_2^{(2)}\right) & \frac{1}{2A}\left(x_1^{(2)} - x_1^{(1)}\right) \end{bmatrix} \begin{Bmatrix} 1 \\ x_1 \\ x_2 \end{Bmatrix} = \underline{\underline{C}}\,\underline{P}$$

Equation 3.12

where the superscript denotes the node number for the element and A is the area of the triangle. Once these shape functions are determined it is possible to evaluate the element integrals given by equations (3.9) and (3.11). At this point the element stiffness matrices (3.9) and force vectors (3.10) can be solved after providing the boundary conditions. Finally, the system of equations are partitioned and solved. Once the equations are solved, the element results can be calculated. In this case, the FEA was for heat transfer, but the derived equations could also be applied to structural deflection or torsion, as well as to irrotational fluid flow. For example, with irrotational flow the temperature term is replaced by a potential function ϕ, in which $\phi_{,i} = v_i$, where v_i is the x_i component of velocity. Additionally, heat flux, q, is equal to the velocity term, and q_{out} is the equivalent

of v_{in}. Certain attributes applied and managed by the developed attribution tool represent boundary conditions used for finite difference equations.

3.3 Object-oriented Programming

The attribution tool created for this research was created using C++ and takes advantage of the object-oriented nature of that programming language. A brief overview of object-oriented programming basics will assist the reader in understanding the nature and development of the work done in this research. Object-oriented programming (OOP) is based on the idea of using "objects." An object is a bundle of related functions and variables. An object may be an instance of a structure, a class, or some other type of data structure. The goal of OOP is to allow portability of large sections of code to other software programs.

3.3.1 C++ Classes

The C++ class is central to the idea of OOP. A class is essentially a blueprint that defines the variables and methods common to all objects of a certain kind. The variables belonging to a class (member variables) can be declared to be standard C variables, such as integer, float, double, etc., or they can be instances of other classes. An example of how to create a class is shown below. The name of the class is ExampleClass, and no member variables or member functions have been added to the class.

```
class ExampleClass
        {
        public:
                /*public member variables and functions go here*/

        private:
                /*private member variables here*/

        protected:
                /*protected members here*/
        };
```

An instance of ExampleClass could then be made by declaring a variable as type ExampleClass. In this manner, all of the functions and variables defined by ExampleClass become available to the newly declared instance.

To better understand the motivation for using classes, a simple real world analogy is appropriate. For example, a specific bicycle is one of many types of bicycles in the world. It is inferred then that this bicycle object is an instance of the class of objects known as bicycles. Bicycles have some variables (the current gear) and behavior (braking and changing gears) in common. However, each bicycle's variables are independent and may be different from that of other bicycles. When designing bicycles, the manufacturers take advantage of the fact that bicycles share common behavior and variables, thus using a single blueprint for the basic shared characteristics. Classes, then, are software blueprints for objects that share similar behaviors and variables.

3.3.2 Class Inheritance

Object-oriented systems have the ability to define classes in terms of other classes. A class may inherit behavior from a parent class. This characteristic is referred to as inheritance. Going back to the bicycle example from section 3.3.1, mountain bikes, road bikes, and BMX bikes are all different kinds of bicycles. In object-oriented terminology, these different bicycles are all subclasses of the parent class, bicycles.

Each subclass inherits variable and function declarations from the parent class. However, subclasses are not limited to the variables and functions defined by the parent class. Subclasses can add to the variables and functions inherited from the parent class. Subclasses can also override inherited functions and provide specialized implementations of those functions. Inheritance is not limited to one "generation." The inheritance tree, or class hierarchy, can extend as far as needed, allowing member functions and variables to be passed down many levels. The farther down in the class hierarchy a class appears, the more specialized its behavior becomes. The ability to pass functionality to other classes from a parent class allows programmers to reuse the code in the base class. These methods and techniques will be used in the development and coding of the attribution tool presented in this thesis.

3.4 Database Management Systems

Database management systems (DBMS) are becoming increasingly important as engineering businesses are storing and retrieving more and more information in a digital format. Access to a database is often referred to as a *query*, thus *query languages* have been created expressly for this purpose. Many query languages have been created for DBMSs. Among those languages, one in particular has become dominant, called *Structured Query Language*, or SQL. Currently, SQL has become the standard database query language and is supported by most DBMSs.

In the 1970's DBMSs organized data into a hierarchy, and such systems were called *hierarchical* systems. Hierarchical systems allowed fast access to stored data; however, they could not easily perform simple ad hoc queries and DBMS users demanded a simpler way to work with data. This demand gave rise to another approach called

relational systems. E.F. Codd developed the relational model in 1970. Relational systems represent data as simply the contents of one or more tables. Examples of the more popular relational DBMSs include IBM's DB2, Microsoft's SQL Server, Microsoft's Microsoft Access, and Oracle Corporation's Oracle DBMS. SQL was developed for use with relational systems; therefore, most relational systems can be accessed via SQL (Chappell 2002).

Relational databases store data in the form of tables. A table is a collection of rows and columns. Rows represent records in the table and columns represent fields. Relational systems allow the user to access any combination of rows and columns through the querying capability of SQL. Figure 3.9 shows an example of a table found in relational systems.

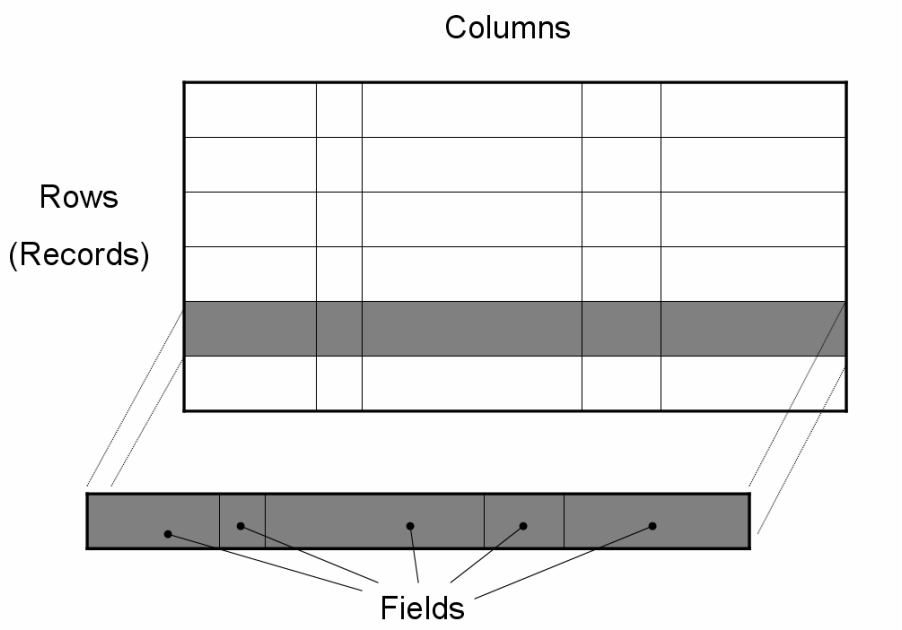


Figure 3.9 Relational database table

The attribution tool created for this research relies heavily on the capabilities of relational DBMSs and the querying capabilities of SQL. There are several advantages to

using a DBMS over file I/O. DBMSs are fast; query languages allow for rapid sorting and presentation of information. They are centralized; all users have access to the same database over a network, rather than each user having individual copies of data files. They are easy to maintain, DBMS administrators can rapidly sort through information and eliminate old data, if necessary. More will be discussed on how and DBMSs are used for this research in Chapter four.

CHAPTER 4: METHOD

This chapter describes the method that was initially architected and proposed in the prospectus of this thesis. The methods used to achieve the research objectives posed in Chapter one are discussed herein. For convenience, the thesis objectives are repeated below.

- Implement a method for CAD-centric attribution that establishes a standardized attribute definition, yet, allows for user-customization.
- Develop a process for managing applied attributes in a manner that allows for seamless data exchange between both CAD and CAE applications.
- Demonstrate that the attribution tool can be used to automate the preprocessing of downstream CAE applications.

The general approach used to develop a CAD-centric attribution tool is discussed in this chapter and can be applied for any CAD system, programming language, database application, or CAE application. However, for this research the attribution tool was created for use within a specific CAD system, the UGS graphics system, developed using the C++ and TCL programming languages, designed to interact with the MySQL DBMS, and demonstrated on ANSYS, a commonly used CAE application. A discussion of how this tool was developed for use with the aforementioned applications will follow in Chapter five and will parallel the discussions of this chapter.

The goals of this research were accomplished via the following steps:

- Creation of DBMS table structure for storage of attribute library definitions, as well as storage of object attributes applied in CAD.
- Development of an object-oriented data structure that invokes the CAD and database API's.
- Development of an attribute translator to extract attributes from the DBMS and implement them in CAE preprocessing.
- Demonstration of preprocessing automation in CAE applications.

The attribution tool developed in this research is a collection of software programs including a series of DBMS tables, an object-oriented data structure, a graphical user interface, and an attribute translator program. Figure 4.1 shows how the different components of the attribution tool are related.

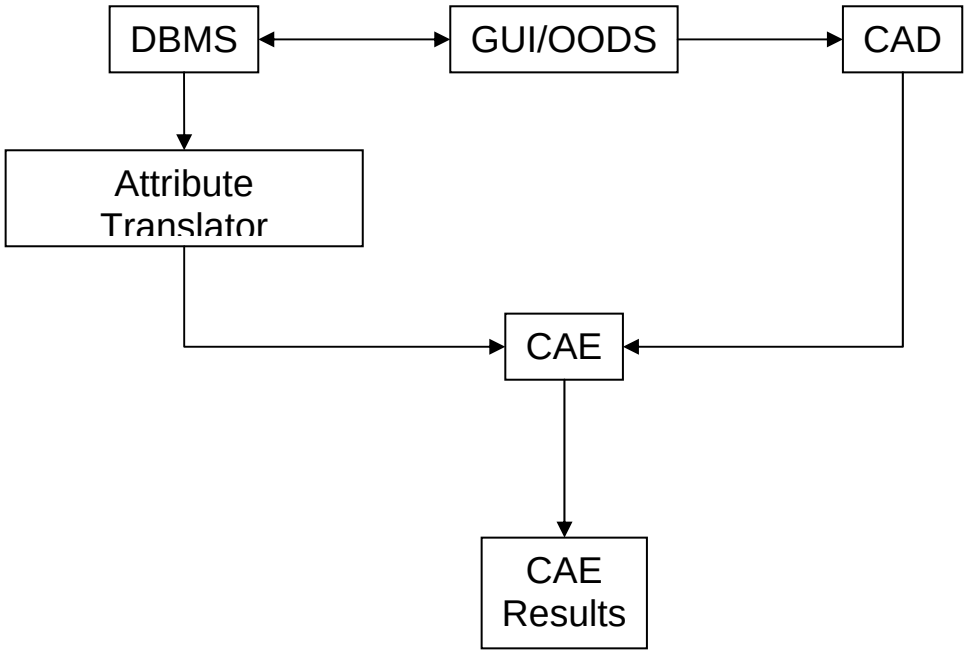


Figure 4.1 Components of CAD-centric attribution tool

The developed attribution tool acts as a bridge between the CAD and CAE environments. The attribution tool does not fundamentally alter the functionality of the CAD or CAE systems, but rather makes CAE preprocessing accessible from the CAD environment, and stores that information as attributes in an external database. The attributes applied during the CAD-centric preprocessing then become part of the product definition data. CAE attributes have previously enjoyed only temporary existence within the CAE application. In many cases, attribute creation is duplicated because different CAE applications require the same attributes to be applied, but the attributes cannot be universally communicated between the different applications. Each separate CAE application applies its particular set of attributes to the product geometry, and those attributes live only within the originator CAE application. The paradigm shift in this research is that all the CAE attributes becomes consolidated into a central repository, where they can be universally passed around as needed to downstream applications. This paradigm eliminates attribute duplication and allows the information to persist with the product throughout its lifecycle. The approach used to create the components of the attribution tool is discussed in this chapter.

4.1 Attribute Storage and Management

When applying attributes in the CAD environment, it is important to determine the best method to store and manage them. Several possibilities exist for attribute storage. Many CAD programs allow attributes to be created and stored within the CAD database. An alternative attribute storage method is file input/output (I/O). Using a file I/O method, attributes would be written to and read from a text file. The final alternative explored in

this research, and the one ultimately chosen, was to use a DBMS for attribute storage and management.

Storing the attributes in an external database has several advantages over file I/O and CAD databases. DBMSs are more flexible in the information they can hold. Where traditional CAD attributes are created using a number of pre-defined fields, a DBMS table can be created allowing the user to enter more information about the attribute. This feature becomes important when developing a method that allows for a standardized attribute language. Typically, CAD attributes do not contain enough information to effectively communicate with downstream CAE applications. Chapter five discusses how the method used in this research for storing additional attribute information in a DBMS enables a standardized attribute language to be developed.

Another advantage of using a DBMS is that they are more centralized than file I/O or the CAD database. When attributes are stored in a DBMS table, that information exists in a single location on a server that multiple users can access over a network. Using a file I/O approach is cumbersome because a new text file containing attribute information would have to be created for every CAD part. Using file I/O for attribute management, engineers would have to keep track of all the associated attribute text files in addition to the CAD files, instantly doubling the number of files to be concerned with. For groups of engineers working together on the same part, everyone on the team would have to have a copy of the associated attribute text file, as well.

DBMSs are more convenient and computationally efficient than file I/O and CAD databases. Database query languages have been developed for DBMSs that are optimized for speed and efficiency. One problem encountered with file I/O is that it is very difficult

to insert or delete lines in the middle of the text file. To delete a line from the middle of a text file, an algorithm must be created that splits the file in half before and after the deleted line, then pastes them back together. This process is an inefficient use of computational resources. Using simple DBMS queries, a user can easily insert, delete, or edit records anywhere in a table.

DBMSs are more user-friendly than file I/O and CAD databases. An engineering analyst who is unfamiliar with a CAD program could open a DBMS table containing attributes and quickly sort through the records to find a desired attribute. By contrast, attributes stored in the CAD database require the user to enter the CAD program and manually search for the attributes, which can be difficult to find a desired attribute when dealing with large, complex models. Using the file I/O method, the engineer would have to sort through a text file to find a particular attribute, which can be tedious for large text files. Simple DBMS queries can quickly present to the user the desired information, rather than manually searching for it.

Aside from storing attributes applied to geometric objects, a DBMS can be used to manage a library of attribute definitions. Attribute definitions stored in a DBMS table provide a solution to the research objective of developing a standardized attribute language. Furthermore, storing attribute definitions in a DBMS rather than hard-coded in a software application enables users to easily access the attribute definitions, making them highly user customizable. The development of the DBMS attribute architecture discussed in chapter five demonstrates how the method for attribute storage used in this research makes it possible for an attribution tool to establish a standardized, yet user-customizable attribute language.

4.2 An Object-Oriented Attribute Data Structure

An object-oriented data structure was established to enable the DBMS attribute architecture to interface with CAD geometry. The object-oriented data structure acts as a "bridge" between the DBMS attribute architecture and the CAD geometry. Interaction between the DBMS and the CAD system is possible via access to their respective APIs within the data structure. The data structure was developed using an object-oriented programming language because instances of attributes are treated as programming objects, each instance containing similar functionality and variables (see section 3.3 for OOP background). The object-oriented data structure has several important responsibilities.

- Dynamically extracts attribute definitions from the DBMS and puts them in the GUI menus.
- When new attribute definitions are created using the GUI, it inserts the definitions into the DBMS and updates the attribute definition fields in the GUI.
- Assigns unique names to geometric CAD objects whereupon attributes have been applied and stores the name both in the CAD database and in the attribute record in the DBMS.
- Keeps track of all the attributes applied to a CAD part and ensures they are properly stored in the DBMS.
- Controls the GUI functionality.

The object-oriented data structure and the DBMS have a bi-directional link, as shown in Figure 4.1. Attributes and attribute definitions can be created, edited, or deleted from either the GUI or the DBMS. The object-oriented data structure, through a series of

intuitive GUIs, allows the user to apply attribute definitions directly to the CAD geometry, whereas working from the DBMS does not. In other words, a user cannot apply an attribute definition to a geometric object directly from the DBMS unless that object already has a unique name applied to it.

4.3 A Graphical User Interface for CAD-centric Attribution

The attribution tool developed for this research is intended to be a general purpose tool capable of applying any conceivable attribute definition to geometric objects in the CAD environment. Thus, it was necessary to create a GUI that was sufficiently robust in its design as to accomplish this objective. The GUI is activated via a user function called from the CAD program. In order to be a generic attribution tool that can communicate with any downstream application, the GUI for the attribution tool should have the following qualities:

- Include pull down menus containing the attribute definitions that are dynamically linked with the attribute definitions stored in the DBMS.
- Allow the user to define new attribute definitions and store them in the Attribute Library table in the DBMS.
- Allow the user to select objects from the CAD model on which to apply attributes. If an object has no name, the GUI should force the user to apply a unique name to the object, storing the name in both the CAD file and DBMS.
- When an attribute is applied, the attribute definition, value, and object name should be stored in the Object Attribute table in the DBMS.

- The GUI should display the attributes that have been applied to the CAD geometry. This information should be dynamically linked to the data in the Object Attribute table of the DBMS.
- The GUI should allow the user to edit or delete applied attributes. This allows for a bi-directional link between the Object Attribute table in the DMBS and the GUI/object-oriented data structure.

Developing a GUI in this manner ensures that the DBMS is always the central repository for attribute data, rather than hard-coding attribute definitions into the object-oriented data structure. After attribution has been completed in the CAD environment, the next step is to use the attributes stored in the DBMS to automate CAE applications. In order to do this, intermediate applications must be developed to interpret attribute definitions and translate them into application specific commands. These intermediate applications are referred to as attribute translators.

4.4 CAE Attribute Translation

After a product's attributes are applied in CAD and stored in the DBMS, a common ground can be reached for communicating what the attribute definitions mean to CAE applications. As part of this thesis, research was performed investigating the feasibility of creating programs that could read in all the attributes pertaining to a CAD file in the DBMS and automate the CAE preprocessing. Such programs are referred to as "attribute translators". The objective of creating attribute translators is to eliminate CAE preprocessing and facilitate optimization processes involving CAE applications. Aside from CAE preprocessing automation, another benefit of developing attribute translators is that many of the steps required for complex preprocessing procedures are contained in the

code and invoked when a certain set of attributes are detected, thus reducing the difficulty and/or tedium of CAE preprocessing.

Prior to the work done in this thesis with attribute translators, a common method used to automate CAE preprocessing for optimization purposes was to access the commands recorded by the CAE application and then instruct the application to run the saved list of commands for every design iteration. In such instances, in order to include CAE applications in optimization loops, the user must perform at least one CAE analysis manually in order for all the actions to be recorded properly. This method enjoys a level of popularity due to its simplicity; however, it has several disadvantages. First, it is CAD part specific, and must be performed anew for every different part. Also, the attributes applied by the CAE application are not available to other CAE applications, reducing the feasibility of concurrent engineering. Using attribute translation programs, CAD geometry and associated attributes can be shared among various CAE applications for multi-disciplinary optimization. Attribute translators can be developed in any programming language; the only requirement for choosing a programming language is that it must be able to access the APIs of both the CAE program and the DBMS. Scripting languages, such as Perl, TCL, Python, etc., are especially convenient in this regard.

There are inherent disadvantages to using attribute translators. First, attribute translators are CAE application specific; each CAE application must have its own unique translator. Next, attribute translators are high maintenance. Every time a user creates a new attribute definition in the Attribute Library table of the DBMS, the translator must be updated to recognize the new definition. In addition, new software releases of CAE applications can introduce changes in the CAE API functions, which require changes in

the attribute translator functions. Finally, the attribution tool as a whole requires significant development time. If only a handful of analyses are required for a design, it is more effective to simply perform the preprocessing in the CAE applications. The power of attribute translators coupled with the attribute library contained in the DBMS is most effectively realized during optimization procedures, where hundreds or thousands of design/analysis iterations occur.

4.5 A Proof Case

To test the feasibility of the developed attribution tool for optimization, a series of different geometries were created to test the robustness of the tool. For each geometry, attributes were applied in the CAD system for subsequent analysis in ANSYS. The design variables in optimization problems often involve topological and non-topological geometric variation. Therefore, a series of dimensional iterations, similar to what would be experienced during an optimization routine, was performed for each geometry to test the response of the attribution tool in a downstream CAE application. Some iterations involved non-topology changing operations, such as altering the values of defining dimensions. Changes in topology were also performed on the different geometries, such as the creation or deletion of lines, faces, and/or vertices. In order for optimization loops to work well, the attribution tool must be very robust when dealing with non-topology changing operations. Topology changing operations are less likely to be included in optimization because they often produce undesirable results. Also, topology changing operations result can encounter the persistent naming problem, which was delimited from this thesis in chapter one. However, changes in topology can be handled by the attribution tool to a certain degree, as will be demonstrated in chapter six.

CHAPTER 5: DEVELOPMENT

The method described in chapter four was implemented for use on the following set of commercially available CAD, DBMS, and CAE software: Unigraphics, MySQL, and ANSYS, respectively. The object-oriented data structure was developed using C++ and the attribute translator was created using the TCL/Tk scripting language. This chapter discusses the development of the DBMS attribute architecture, the object-oriented data structure/GUI and attribute translator in conjunction with the aforementioned set of software programs.

5.1 DBMS Attribute Architecture Development

The DBMS attribute architecture was created using a free, SQL based DBMS called MySQL, and consists of three different tables used to store attributes and attribute definitions. These tables are the *Attribute Library* table, the *Part* table, and the *Object Attribute* table, respectively.

5.1.1 Attribute Library Table

There are two purposes for creating an Attribute Library table in a DBMS. The first is to establish a standardized set of attribute definitions that attribute translators can interpret. The purpose for creating attribute translators is discussed in section 4.4. The second purpose of the Attribute Library table is to provide a place to store additional

information about an attribute. Many CAD systems have the built-in functionality that allows attributes to be applied to geometric objects. CAD attributes consist of a title, value, and data type applied to a geometric object, which get stored in the CAD data file. However, in order for downstream CAE applications to use attributes, more information is required than is typically given by a CAD attribute. The Attribute Library table builds upon the CAD attribute definition, allowing the necessary additional attribute information to be passed to the CAE application. The columns of the Attribute Library table are *Domain*, *Sub Domain*, *Title*, *Type*, and *ID*. Once the user fills in these fields, an attribute definition is considered fully defined. Every row in the Attribute Library table constitutes an attribute definition.

The purpose for inclusion of the domain and sub domain columns is to help the attribute translator group sets of similar attributes. These fields act as flags for the translator, instructing it how to sort the attributes. These two fields are the most significant source of additional information not included by the traditional CAD attribute. The title and type fields are common to both the CAD attribute and the Attribute Library table. The title field acts as a keyword instructing the downstream CAE application how the attribute is to be applied. The type field tells the CAE application what type of data the attribute contains, whether it is a real number, string, integer, etc. The ID field is an auto-incrementing integer that is automatically created by the database for each new attribute definition. Each record, or attribute definition, in this table has its own unique identification number. The purpose for this will be explained in the section 5.1.3.

It is important to note the fields that have been included in the Attribute Library table as well as those which have been notably excluded. The Attribute Library table

excludes the *value* and *object name* fields, whereas CAD systems include them in their attribute definitions. The purpose for their omission is to make attribute definitions independent of value or geometric object. This is a significant paradigm shift from the attribution method used by CAD systems. Before this research, it was impossible to communicate attributes to CAE applications in a standardized manner because the attributes were defined by their value and geometric object, making every attribute unique. It is via an Attribute Library table that is value and geometric object independent that one of the most significant contributions of this research is possible, that of attribute standardization.

5.1.2 Part Table

The Part table is the simplest of the three tables. It contains only two columns, or fields: *Part Name* and *ID*. The Part Name field simply contains the name of the file where the CAD geometry resides. The ID field is an auto-incrementing integer that provides a unique, numeric identification for each CAD file that has had attributes applied to it. The purpose for the creation of this table is for convenience in development of the object-oriented data structure and attribute translator.

5.1.3 Object Attribute Table

The Object Attribute table contains instances of attribute definitions from the Attribute Library table for specified values and/or geometric objects. The relationship between the Attribute Library table and the Object Attribute table is analogous to class inheritance in object-oriented programming. Each entry in the Object Attribute table inherits all the information of an Attribute Library record, but also includes additional

information pertaining to the attribute definition, including the object name, value, and CAD part name. The fields of the Object Attribute table are as follows: *Object Name*, *Value*, *Library ID*, *Part ID*, and *ID*.

The *Object Name* field allows the user to define the geometric object whereupon an attribute definition is applied. In order for an attribute definition to be applied to a geometric object, a unique name for that object must exist. However, it is acceptable to apply an attribute without assigning it to a geometric object, leaving the field blank. Such attributes are referred to as "part attributes," where the attribute pertains to the CAD part as a whole. Examples of part attributes are units of measure, or tolerances.

The *Value* field allows the user to enter a value for the applied attribute. It is possible that a particular attribute definition may not require any value; for such cases the *Title* field of the Attribute Library table and the *Object Name* field from the Object Attribute table may provide enough information for the CAE application. For example, certain FEA mesh generation algorithms require definition of a "seed face" in order to sweep a mesh through a volume. No value is required for this attribute, only the information telling the meshing program which face is the "seed face." Alternatively, some attributes may require more than a single value. An example of this occurs when a force is applied at a point and the x, y, and z components of that force need to be stored as values. To accommodate for such situations, an attribute definition for each different value required must be created in the Attribute Library table. Figure 5.1 shows how the attribute definitions for an attribute requiring multiple values would appear in the Attribute Library table.

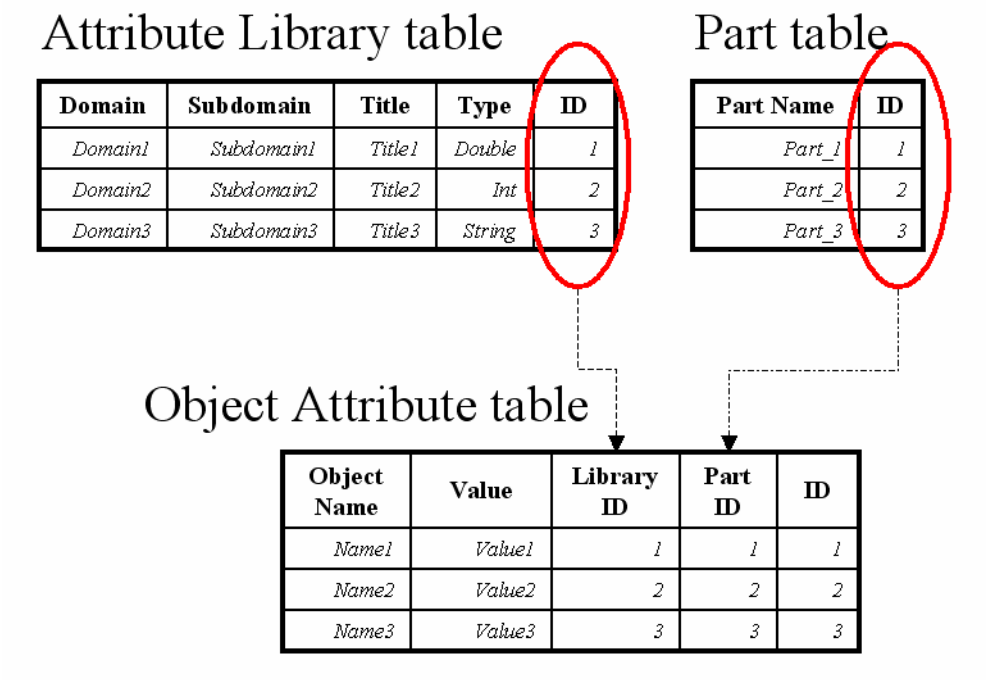
Table 5.1 Attribute definition for multi-component attributes

Domain	Subdomain	Title	Type	ID
…	…	…	…	…
…	…	…	…	…
Structural	Loads	Point Load, X	Double	15
Structural	Loads	Point Load, Y	Double	16
Structural	Loads	Point Load, Z	Double	17
…	…	…	…	…
…	…	…	…	…

The limitation of only being able to apply one value per attribute is certainly one of the weaknesses of the developed attribution tool. However, the efficiency and speed at which DBMSs operate compensates for the lack of elegance of this solution. The flexibility of the *Value* and *Object Name* fields of the Object Attribute table enable the attribution tool developed for this research to be a universal attribution tool, applicable to any conceivable attribute definition, value, and/or geometric object.

The *Library ID* and *Part ID* fields contain integers referring to the ID field from the Attribute Library table and Part table, respectively. The purpose for their inclusion in this table is to allow the Object Attribute table to "inherit" information from the Attribute Library table and the Part table. Figure 5.2 shows how the Object Attribute table inherits information from the other two tables via their respective *ID* fields.

Table 5.2 Inheritance in DBMS tables



By including the ID from the Attribute Library table and the Part table, the Object Attribute table can easily access the information in the records of the respective tables based on the ID value through SQL queries. It is appropriate, at this point, to make a distinction between *attributes* and *attribute definitions* as discussed in this thesis. Attribute definitions are the records of the Attribute Library table, while attributes are the records of the Object Attribute table. The relationship between attribute definitions and attributes is analogous to object oriented programming and is unique to this research.

5.1.4 Standardization and Customization

As previously mentioned, an objective of this research is to develop an CAD-centric attribution method that established a standardized attribute format, yet also allowed users to dynamically edit, delete, or create new attribute definitions as necessary.

The DBMS attribute architecture developed in this research achieves attribute standardization because the attribute definitions are created independent of geometric object or value. It could be said that an attribute "language" has been created in this research because of the standardized nature of attribute definitions in the Attribute Library table.

The DBMS attribute architecture constitutes one-third of an attribution tool developed for this research. The DBMS is central to the attribution tool and its architecture establishes the attribution philosophy of defining attributes independent of object name or value. The other two parts of this research are concerned with the application of that philosophy. The second part of the attribution tool is an object-oriented data structure, which deals with the interaction between the DBMS and CAD. The third part of the attribution tool is an attribute translator, whose purpose is to extract attributes from the DBMS and send them to CAE applications.

5.2 Object-oriented Data Structure Development

An object-oriented data structure was developed using the C/C++ programming language. Access to the Unigraphics API and MySQL API is granted to the object-oriented data structure via their respective header files. The object-oriented data structure relays attribute information between the DBMS and the CAD attribute GUI. During the initial stages of this research, it was planned to use the object-oriented data structure to use the Unigraphics and MySQL APIs to send commands to store attributes in both the CAD database and the DBMS. However, during the development of the object-oriented data structure, it became obvious that such redundancy in data storage would not be necessary and only the DBMS needed to contain all the attribute information. It was

discovered that only a single part of the attribute information must be stored in both the CAD database and the DBMS: the object name. When an attribute is applied to a geometric object, the object-oriented data structure prompts the user to assign a unique name to the CAD object if one does not already exist. That name is then stored both in the CAD file and in the DBMS. The purpose for storing the object name in the CAD file is such that after the file is exported into CAE, the attribute translator can know which geometric object to apply certain attributes to. The process of sorting attributes by object name after geometry import to CAE will be discussed in section three of this chapter. Another reason why the object-oriented data structure does not store attributes in the CAD database is because the traditional CAD attribute definitions are not compatible with the attribution method developed in this research. Thus, it became clear that the Unigraphics API functions for programmatically applying CAD attributes would not be used as extensively as was initially thought because this research does not deal with traditional CAD attributes and there was no need to store the attributes in the CAD database. Rather, the object-oriented data structure uses the Unigraphics API mainly to apply unique names to geometric objects, query the working CAD model for the file name, highlight geometric objects to facilitate object selection, and provide access to the GUI functions. To manage attribute creation and storage, the object-oriented data structure uses functionality afforded by the MySQL API.

The object-oriented data structure is initialized from the CAD-environment from a user-function, which brings up a GUI. During the initialization of the GUI, the object-oriented data structure queries the Attribute Library table in the DBMS and fills up a series of menu items in the GUI containing the fields for attribute definitions. In addition,

the data structure also queries the Object Attribute table in the DBMS, which fills up fields in the GUI displaying to the users the attributes that have already been applied to the specified CAD file. If no attributes have been applied, the fields will be empty. Pseudo code showing how the data structure interfaces with the DBMS and the GUI during initialization is shown below.

```
.
.
.
connect to DBMS server
get working part name from CAD, check to see if part name
     exists in Part table of DBMS
     if (part name does not exist in DBMS)
          insert record into Part Table containing CAD part
          name
     else
          do nothing
query Attribute Library Table for unique domain fields, fill
     up array in data structure containing domains
fill up domain pull-down menu in GUI with domain array from
     data structure
query Object Attribute Table for any attributes already
     existing for part name, creating an array of object
     names on whom attributes exist
check for consistency between object names in DBMS and CAD
     file
     if (object name exists in DBMS but not in CAD file)
          delete all records in Object Attribute Table
          containing that object name
     else
          do nothing
fill up fields in the GUI to display the existing attributes
     for that part
.
.
.
```

Attributes and attribute definitions existing in the DBMS are retrieved and stored as instances of attribute “objects” in the data structure during initialization. Class functions in the data structure are used to control how attributes are created and managed in conjunction with the DBMS. In the development of the object-oriented data structure,

two classes were created to handle the attribute information: the Library class and the Attribute class.

5.2.1 The Library Class

The Library class is the base class upon which the Attribute class inherits. The purpose of the Library class is to establish the state of an attribute (domain, sub domain, title, and type). The Library class in the object-oriented data structure parallels the Attribute Library tables in the DBMS in that it contains information related to attribute definitions, effectively serving as a library for storing attribute definitions. The Library class contains functions for extracting the attribute definitions from the DBMS, storing them in the data structure, and inserting them into the GUI. In addition, the Library class contains functions for creating new attribute definitions in the DBMS. The pseudo code for creating a new attribute definition is shown below.

```
.
.
.
open new attribute definition dialog
get user input for domain, sub domain, title, and type
store values as new attribute object definitions in data
     structure
use DBMS query to create new record in Attribute Library
     table
insert new attribute definition into domain, sub domain,
     title, type attribute definition fields
.
.
.
```

The ability to create new attribute definitions allows the attribution tool to fulfill the research objective of being user-customizable.

5.2.2 The Attribute Class

The Attribute class inherits from the Library class. An instance of an attribute in the data structure will have all the information and functionality of its base class (the Library class), in addition to new functionality and variables. The attribute class is the object-oriented data structure equivalent of the DBMS Object Attribute table in that it contains information about attributes, while the Library class contains information about attribute definitions. The new variables used by the Attribute class include the object name of the attribute, the value given the attribute, and the name of the CAD file. The Attribute class contains functions for inserting, deleting, and editing records in the Object Attribute table, as well as functions for applying names to geometric objects in Unigraphics.

Each time the user applies an attribute definition to an object and/or assigns a value to it, a new instance of the attribute class is created and stored in the data structure. At the same time, a new record is created in the DBMS containing the information and the field in the GUI is updated to include the newly applied attribute. The following pseudo code demonstrates how instances of applied attributes are created.

```
.
.
.
obtain values for domain, sub domain, title, and type from
     GUI fields
query DBMS and obtain ID of attribute defined by the domain,
     sub domain, title, and type
obtain object name and value from GUI fields
create new instance of attribute in data structure using
     attribute ID, value, object name, and part ID from user
     input
insert newly created attribute from data structure into
     Object Attribute table in DBMS
```

```
add attribute to GUI field displaying attribute list for
     current CAD part
.
.
.
```

5.2.3 The Graphical User Interface

The developed attribution tool's graphical user interface gives the user access to the functionality of the object-oriented data structure, the data in the DBMS, and the geometry in the CAD system. The GUI is an intuitive tool to walk the user through the process of applying attributes to CAD geometry. The developed GUI was created using Unigraphics' GUI development tool, UI Styler. A priority in the GUI development was to make it as general an attribution tool as possible. All of the specific attribute information must be independent of the GUI, and stored in the MySQL DBMS. A result of the generalized nature of the GUI is that it appears simple and intuitive. The complexity of the GUI lies in knowing what attributes exist in the DBMS and how they are intended to be used. When the user invokes the compiled object-oriented data structure from Unigraphics' user-function menu, the GUI is initialized and appears as a window in the Unigraphics. Figure 5.3 shows the GUI upon initialization.

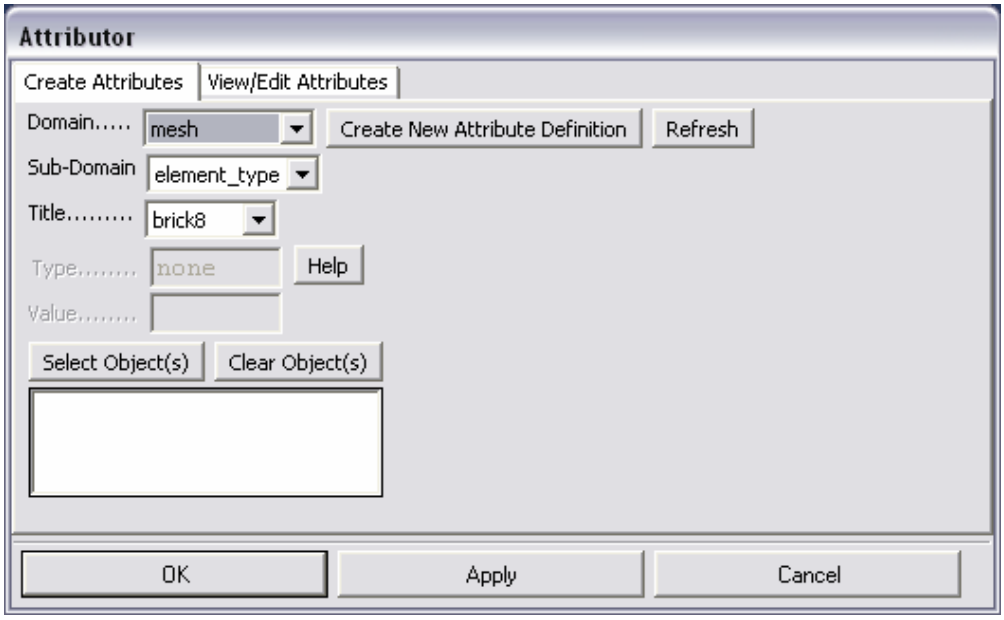


Figure 5.1 GUI for attribution tool

The items listed in the pull-down menus for the Domain, Sub-Domain, Title, and Type fields are dynamically linked to the MySQL Attribute Library table and allow the user to choose which attribute definition he or she wishes to apply. New attribute definitions can be created by clicking the 'Create New Attribute Definition' button, which brings up a new window as shown in figure 5.4 below.

Figure 5.2 GUI window to create a new attribute definition

The user simply enters the information needed to create an attribute definition and the Attribute Library table will be dynamically updated to include the new definition. The items in the Domain, Sub-Domain, Title, and Type fields are updated by clicking the 'Refresh' button.

If the attribute definition is of type double, integer, or string, the Value field becomes active and allows the user to enter in a value. When the user clicks on the 'Select Object(s)' button, the GUI hides itself and initializes another GUI window whose purpose is to help the user in geometric object selection. This window allows the user to filter geometric objects for easier selection. When all the desired geometric objects are selected, the user clicks 'OK' and is returned to the original GUI, whereupon a list of the geometric objects' names appear in the window below the 'Select Object(s)' button. If the user had selected an object that did not have a name applied to it yet, another window appears prompting the user to create a unique name to associate for that object. Clicking 'Apply' creates a record in the Object Attribute table in the DBMS containing the attribute definition and value for each geometric object selected. It will also create a new instance of an attribute in the object-oriented data structure and will display all the attribute instances under the 'View/Edit Attribute' tab, as shown in figure 5.5 below.

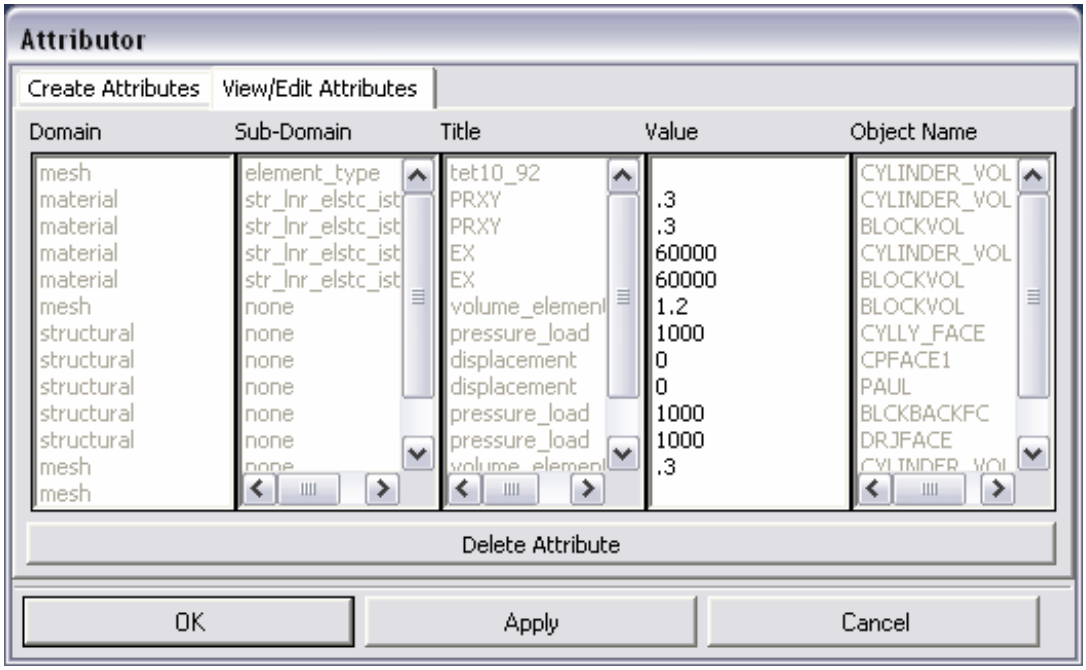


Figure 5.3 View and edit applied attributes

Double-clicking the values in the 'Value' column allows the user to edit the applied values. Updates in the values occur dynamically in the MySQL database after the edit. Attributes can also be dynamically deleted from the object-oriented data structure and the MySQL database by selecting on an attribute and clicking the 'Delete Attributes' button.

The GUI provides access to the CAD-centric attribution method developed herein. It is controlled by functions in the object-oriented data structure, and is dynamically linked to attributes and attribute definitions contained within the MySQL DBMS. It was designed to be used as an all purpose attribution interface, where all attribute knowledge is granted to the GUI by the DBMS. A complete listing of the code used to develop the GUI and the object-oriented data structure is found in Appendix A and B.

5.3 CAE Attribute Translator Development

The attribute translator constitutes the final part of the developed attribution tool. It can be thought of as the inverse of the object-oriented data structure. Whereas the object-oriented data structure creates instances of attribute objects, the attribute translator extracts the information created by the object-oriented data structure and sends it to CAE applications. Most CAE applications have APIs that allow the user to perform CAE operations programmatically. The attribute translator developed herein exploits this characteristic of modern CAE applications by developing translation algorithms that uses the DBMS API to extract all the attributes pertaining to CAD geometry from the DBMS and the CAE API to perform specific commands in the CAE application. The following pseudo-code demonstrates the algorithm for cycling through attributes in the DBMS and sending commands to the CAE application.

```
.
.
.
get CAD part name
get list of object names from Object Attribute table where
     part name = CAD part name
loop through list of object names
     get list of object attribute id's of attributes for a
          given object name
     loop through list of Object Attribute id's
          get domain_name, subdomain_name, title_name, and
          type of attribute id[j]'s library id
          if (domain == domain_name && subdomain ==
               subdomain_name && title == title_name)
                    get value of applied attribute
                    apply value to object[i]
.
.
.
```

This algorithm essentially cycles through all the attributes on an object name, one object name at a time, and sends commands to CAE based on a recognized set of domain, sub domain, and title names. Figure 5.6 graphically depicts this algorithm.

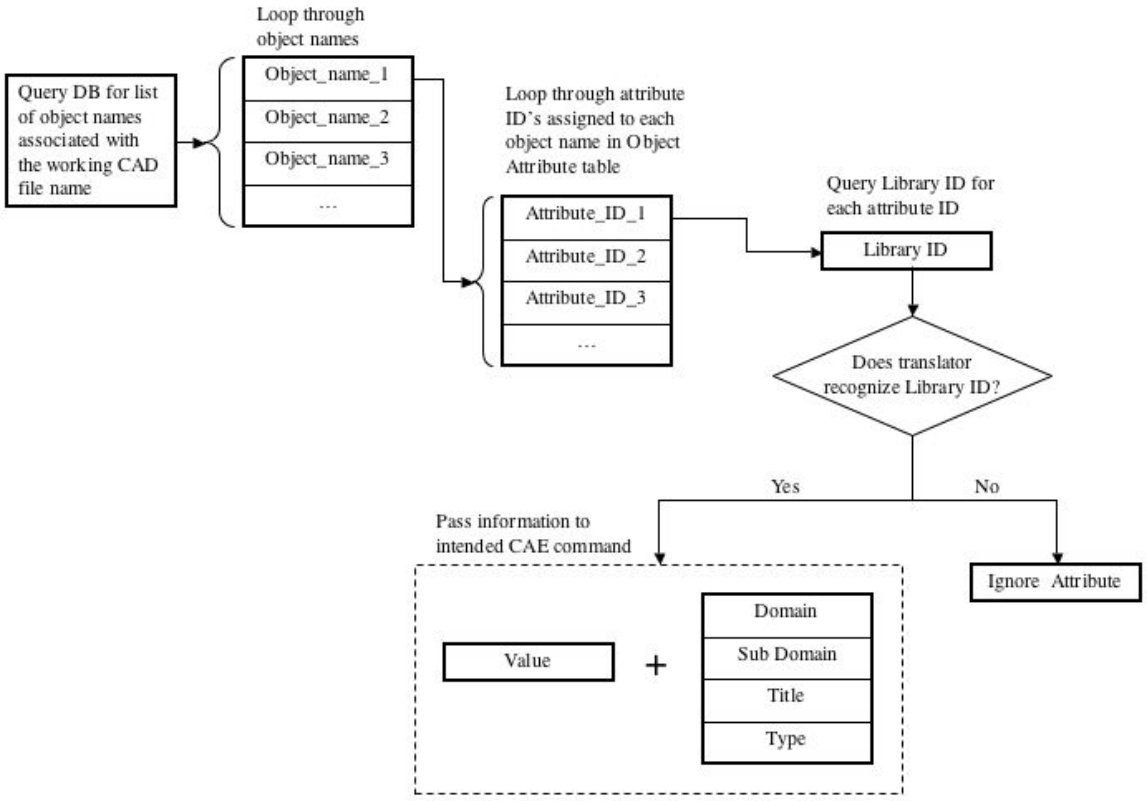


Figure 5.4 Attribute translator algorithm

In order for this algorithm to operate, the user must tell the attribute translator the name of the CAD file in the DBMS on which to operate. There are many possible methods to accomplish this. The method used in this research is to designate the initial job name for the CAE application the same as that of the CAD file. This allows for a simple command in the CAE API to ask the job name and compare it to names used in the Part table of the DBMS. Using that Part name, the translator sends a query to the DBMS to obtain the list of object names for that part. If a record exists where there is no object

name, it is recognized to be a part attribute by the translator, and a similar algorithm exists for handling them.

The algorithm described above follows the same attribute philosophy as developed in the DBMS attribute architecture. It treats object names and values separately from attribute definitions. In this manner, an attribute translator can communicate with the attribute definitions contained within the Attribute Library table. The algorithm described above may need to be performed multiple times in an attribute translator. For example, a structural FEA program may require that the geometry be completely meshed before any loads are applied to it. For such a case the algorithm would need to be performed once for the mesh attributes, then a second time for the structural load attributes. In this manner the attributes can be processed in the correct order as the CAE application demands.

The algorithm described in this section can be used to create translators for many different CAE applications. The complete TCL code for the developed attribute translator can be found in Appendix C. The attribute translator is the third and final piece of the attribution tool developed for this research. The concurrent engineering paradigm, which calls for parallel engineering processes, becomes possible when attribute translators have been developed for multiple CAE applications.

CHAPTER 6: RESULTS

The CAD-centric attribution tool developed in this research, consisting of the object-oriented data structure, GUI, external DBMS attribute architecture, and CAE attribute translator, was tested on a series of different geometric configurations. The different geometric configurations chosen present unique tests of the attribution tool's capabilities. The attribution tool was tested for each geometric case study under changes in parametric and topologic data, as well as changes in applied attribute data.

6.1 Results: Case Studies

Three case studies were chosen to examine the capabilities of the attribution tool. For each case study, a unique set of parametric geometry was created and a series of attributes were applied using the developed attribution tool. A structural FEA analysis was then automatically performed for the original geometry in ANSYS via the attribute translator program. Changes in the parameters, topology, and attribute data were then imposed and the analysis performed anew. The purpose for such iterations was to test the effectiveness of the attribution tool in an optimization routine. This section will discuss the geometry used in the case studies and why it was chosen, as well as the attributes applied to the geometry.

6.1.1 Case Study 1: Constructive Solid Geometry

The first case study involved geometry created by CSG primitives. In this case, the CSG primitive was a simple block. This geometry was chosen because CSG primitives maintain a strong presence in the current generation of solid modeling programs. Figure 6.1 shows the geometry used in this case study.

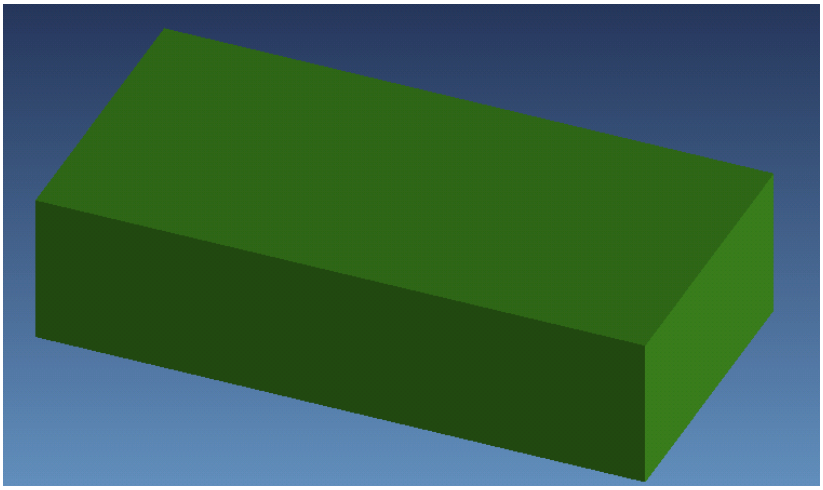


Figure 6.1 Case study 1 geometry

Attributes were applied to the block such as to subject one of the faces to be constrained with respect to the X, Y, and Z-axes, while the opposite face was assigned a pressure load. Figure 6.2 shows the GUI of the attribution tool running natively in UG and the attributes that were applied to set up the subsequent FEA analysis.

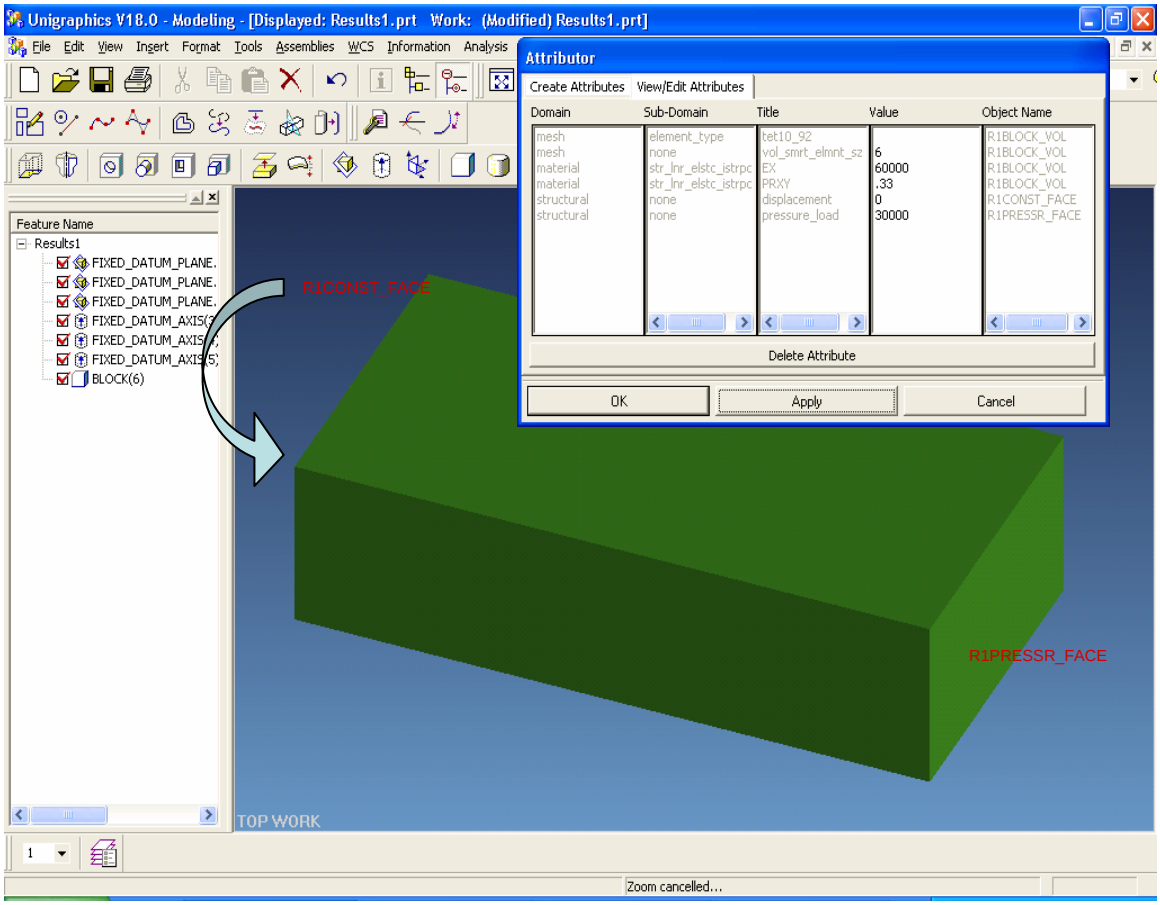


Figure 6.2 Attribute application for case study 1

Aside from the load and displacement constraints, additional attributes were also applied to the block. Material properties and FEA mesh information were mapped to the solid volume of the block. Table 6.1 summarizes the applied attributes for case study 1.

Table 6.1 Case study 1 attributes

Domain	Sub-Domain	Title	Value	Object Name
Mesh	Element type	Tet10_92		R1BLOCK_VOL
Mesh	None	Vol_smrt_elmnt_size	6	R1BLOCK_VOL
material	Str_lnr_elast_istrpc	EX	60000	R1BLOCK_VOL
material	Str_lnr_elast_istrpc	PRXY	.33	R1BLOCK_VOL
structural	None	Displacement	0	R1CNST_FACE
structural	None	Pressure_load	30000	R1PRESSR_FACE

The first attribute in Table 6.1 defines the type of FEA element used to mesh the block, while the second row tells ANSYS how to space the nodes along the solid edges. The third and fourth attributes in Table 6.1 define the Modulus of Elasticity and Poisson's Ratio of the volume, respectively. The fifth attribute constrains face R1BLOCK_VOL in the X, Y, and Z directions, while the final attribute applies a pressure load of 30 ksi to the face R1PRESSR_FACE. The object names and values were applied using the GUI, while the domain, sub-domain, and titles were predefined attribute definitions stored in the DBMS. The second and third case studies are similar in nature to the first, in that they constrain one face as fixed in coordinate space and apply a pressure load to another face. They also use the same meshing strategy and have similar material properties.

6.1.2 Case Study 2: B-Rep Geometry

The second case study examined the attribution tool's interaction with B-rep geometry. The B-rep geometry created for this case study consists of a 3rd degree Bezier curve revolved around an axis. This case study presents an opportunity to explore the manner in which the developed attribution tool interfaces with the CAD system and the

complex, mathematically defined curves and surfaces of B-rep geometry. Figure 6.3 illustrates the B-rep geometry.

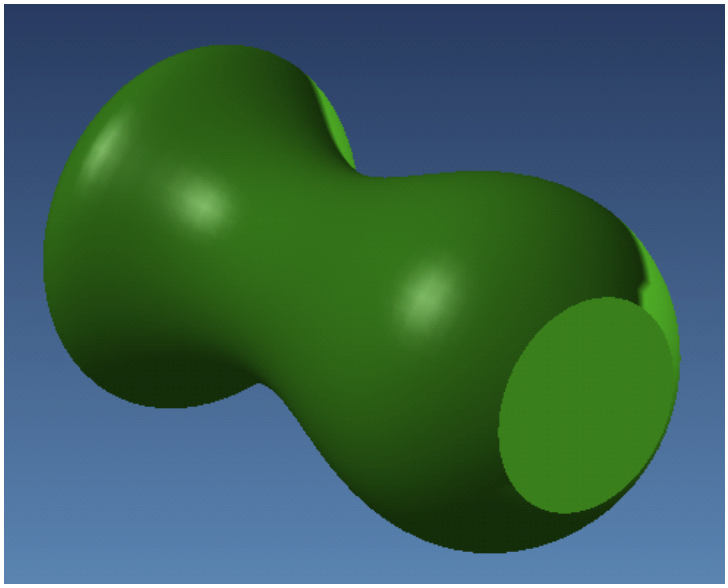


Figure 6.3 Case study 2 geometry

In a similar manner to case study 1, a pressure load attribute was mapped to the visible planar face in Figure 6.3 and a displacement constraint applied to the opposite planar face. The attributes for this case study are summarized in Table 6.2.

Table 6.2 Case study 2 attributes

Domain	Sub-Domain	Title	Value	Object Name
material	Str_lnr_elast_istrpc	EX	60000	BEZ_VOL
material	Str_lnr_elast_istrpc	PRXY	.33	BEZ_VOL
Mesh	None	Vol_smrt_elmnt_sz	9	BEZ_VOL
mesh	Element type	Element_type	tet10_92	BEZ_VOL
structural	None	Displacement	0	R2DISP_FACE
structural	None	Pressure_load	70000	R2P_FACE

6.1.3 Case Study 3: Discrete Features

The final geometry tested using the developed attribution tool contains discrete features. Discrete features are copies of a set of pre-existing features and they exist in an array defined by the user relative to the original set of defining features. The discrete feature used in this case is an array of box shaped bosses on a flat rectangular geometric primitive. Figure 6.4 shows the initial geometry used for this case study.

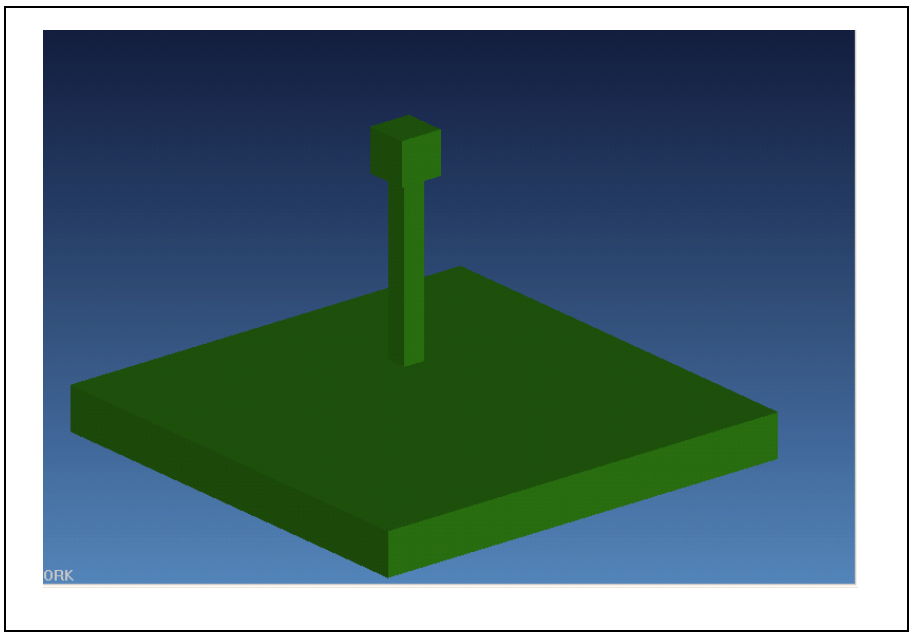


Figure 6.4 Case study 3 geometry

General FEA mesh and material related attributes were applied to the geometry. A structural load was applied to a face on the boss, and the bottom face of the base was given a displacement constraint of zero, meaning the face is fixed in the X,Y, and Z directions. The attributes applied for this initial geometry are summarized in Table 6.3.

Table 6.3 Case study 3 attributes

Domain	Sub-Domain	Title	Value	Object Name
material	Str_lnr_elast_istrpc	EX	60000	DF_VOL
material	Str_lnr_elast_istrpc	PRXY	.33	DF_VOL
mesh	None	Vol_smrt_elmnt_sz	9	DF_VOL
mesh	Element type	Element_type	tet10_92	DF_VOL
structural	None	Displacement	0	R3DISP_FACE
structural	None	Pressure_load	70000	R3P_FACE

6.2 Results: FEA Automation and Design Iterations

This section discusses the performance of the developed attribution tool for the aforementioned case studies. The test procedure for each case study consisted of the following process. First, a set of attributes were applied to solid geometry in Unigraphics and subsequently stored in MySQL. Next, an automated structural analysis was performed in ANSYS via the attribute translator macro. Then, to simulate iterations in an optimization routine, the parametric geometry was altered and reanalyzed for several different configurations. To judge the performance of the developed attribution tool, a series of questions were posed for the results of each case study. The questions are as follows:

1. Did the attributes map correctly between Unigraphics and MySQL?
2. Did the translator program correctly automate the structural analysis in ANSYS?
3. Were attributes lost after parametric changes in the geometry?
4. Were attributes lost after changes in topology?
5. Were changes in attribute values accurately passed to the structural analysis?

6. How long did the automated analysis require, including preprocessing, meshing, solution, and post-processing?

6.2.1 Results Case Study 1: CSG Primitive

Seven different configurations were tested using the CSG primitive block from case study 1. Configurations 1-4 included modifications to the parametric geometry, while the configurations 5 and 6 incorporated changes in topology. Configuration 7 is a special case and will be discussed separately. Figure 6.5 shows configurations 1-6 with the resultant automated mesh and post-processing results for each.

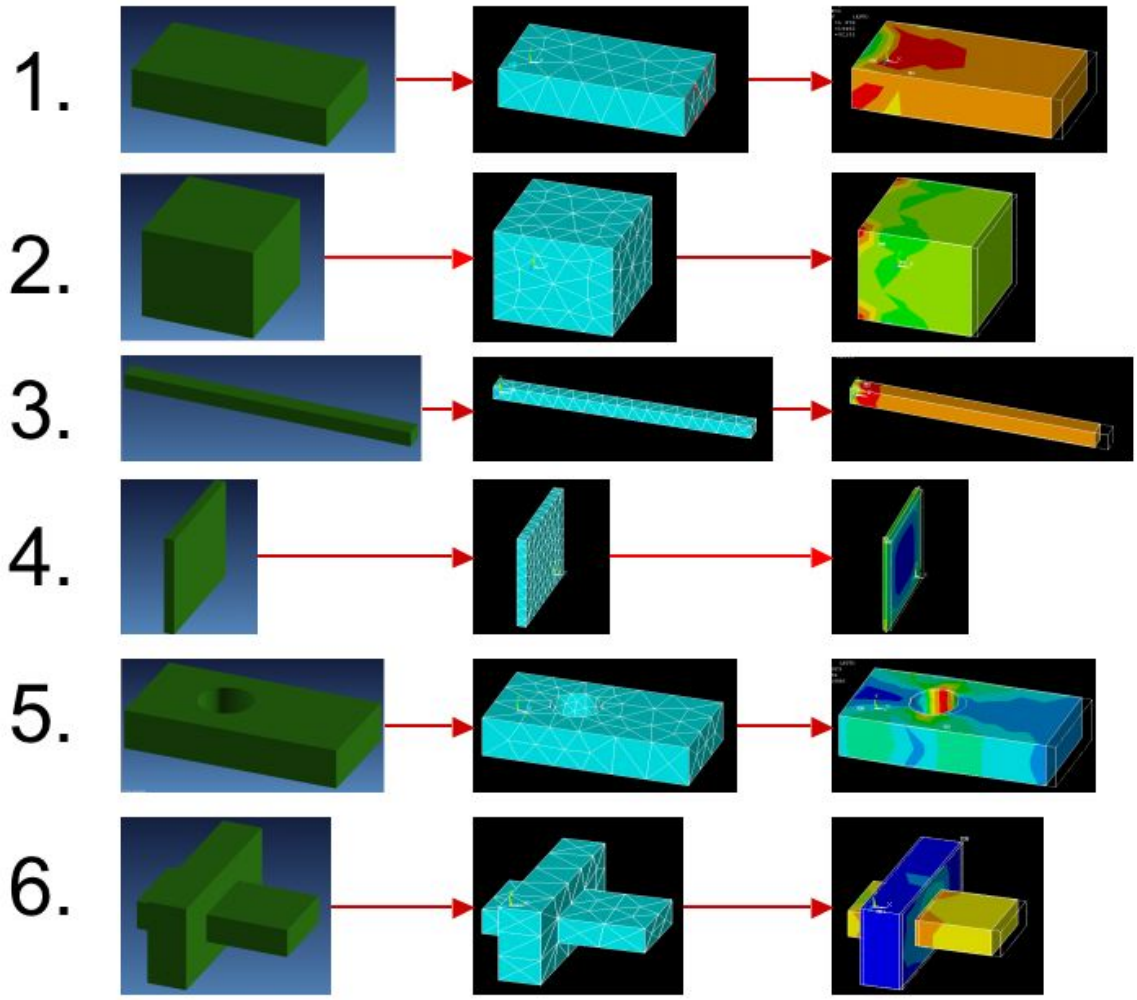


Figure 6.5 Case study 1 results

The series of questions used to determine the performance of the developed attribution tool were applied to each case individually, as well as to the results of all six collectively. The results for each configuration and are summarized in Table 6.4 below.

Table 6.4 Case study 1 performance characteristics

Did the attributes map correctly between Unigraphics and MySQL?	Yes
Did the translator program correctly automate the structural analysis in ANSYS?	Yes
Were attributes lost after parametric changes in the geometry?	No
Were attributes lost after changes in topology?	No
Were changes in attribute values accurately passed to the structural analysis?	Yes
How long did each automated analysis require, including preprocessing, meshing, solution, and post-processing?	~10 sec.

In this case study, the attribution tool performed predictably and accurately. Attributes applied via the GUI in UG mapped correctly to the attribute tables in MySQL. Changes in parametric geometry and topology had no affect upon the attributes stored in MySQL. The translator code executed to completion for each case study without error by importing the geometry, applying the pertinent boundary condition and mesh attributes, performing the solution, and displaying the post-processed results. For configuration 3, the force applied to the pressure face was modified in the GUI in UG, and the value was correctly passed through the MySQL attribute tables into ANSYS. The time required to perform the automated analysis was very low due to the simple geometry used and the relatively coarse mesh.

It should be noted that, for configurations five and six, there were no topological changes to two faces upon whom boundary conditions were applied. Such a scenario pertains to the class of problems known as “the persistent naming problem,” as mentioned in Chapter 1. The persistent naming problem was delimited in this research, but, for the

sake of completeness, configuration 7 included topological changes to faces whereupon attributes were applied. The results for configuration 7 are displayed in Figure 6.6.

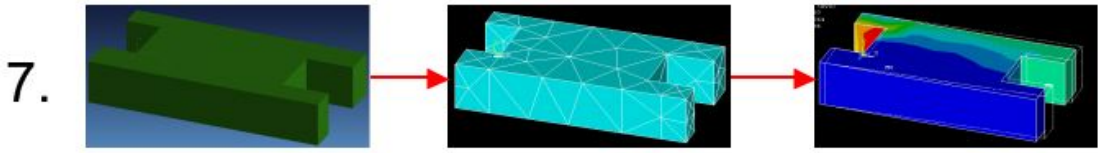


Figure 6.6 Results case study 1, configuration 7

The results for this configuration were less predictable than the previous six configurations. A slot feature on each end of the block divided the single faces of the pressure face and constraint face. When the pressure face and the constrained face were split into three faces by a slot feature, Unigraphics was forced to create a new name for the two new faces, while assigning one of the three faces the name previously assigned to the single face. The translator code still executed to completion without error for this configuration, but the user had no control over which of the three faces should inherit the attribute associated with the original entity name. In addition, this problem is highly CAD system dependent. For a given change in topology, different CAD programs will assign new names and transfer existing names to topological entities in different manners. Therefore, for configurations involving the persistent naming problem, caution should be exercised such that the user ensures the correct topological entities inherit the legacy attributes as the user intends. Specifically, for problems involving the persistent naming problem, the user must open the GUI in the CAD environment and manually inspect that the attributes are applied properly. Therefore, optimization procedures using the

developed attribution tool are not recommended for geometry involving the persistent naming problem

6.2.2 Results Case Study 2: B-Rep Geometry

For this case study, 3 different geometric configurations were tested. Configuration 1 includes the base geometry described in section 6.2.1. For configuration 2, topology changes were introduced via a cylindrical boss cut out from the pressure face. Note that, once again, the persistent naming problem arises in this research for configuration 2. Parametric changes were introduced to the base geometry for configuration 3. The shape of the B-rep surface was modified via changes in the location of the control points used to define the Bezier curve. In addition to modifying the geometry for configuration 3, it also underwent another test in which the value of its mesh density attribute was modified. In complex analyses, mesh optimization is often just as important as geometric optimization. Therefore, analysts will spend considerable resources modifying the mesh attributes without changing the geometry at all. Thus arises the need to verify that the developed attribution tool can be used to modify attribute values in optimization loops. The results for each of the 3 configurations tested for case study 2 are graphically depicted in Figure 6.7.

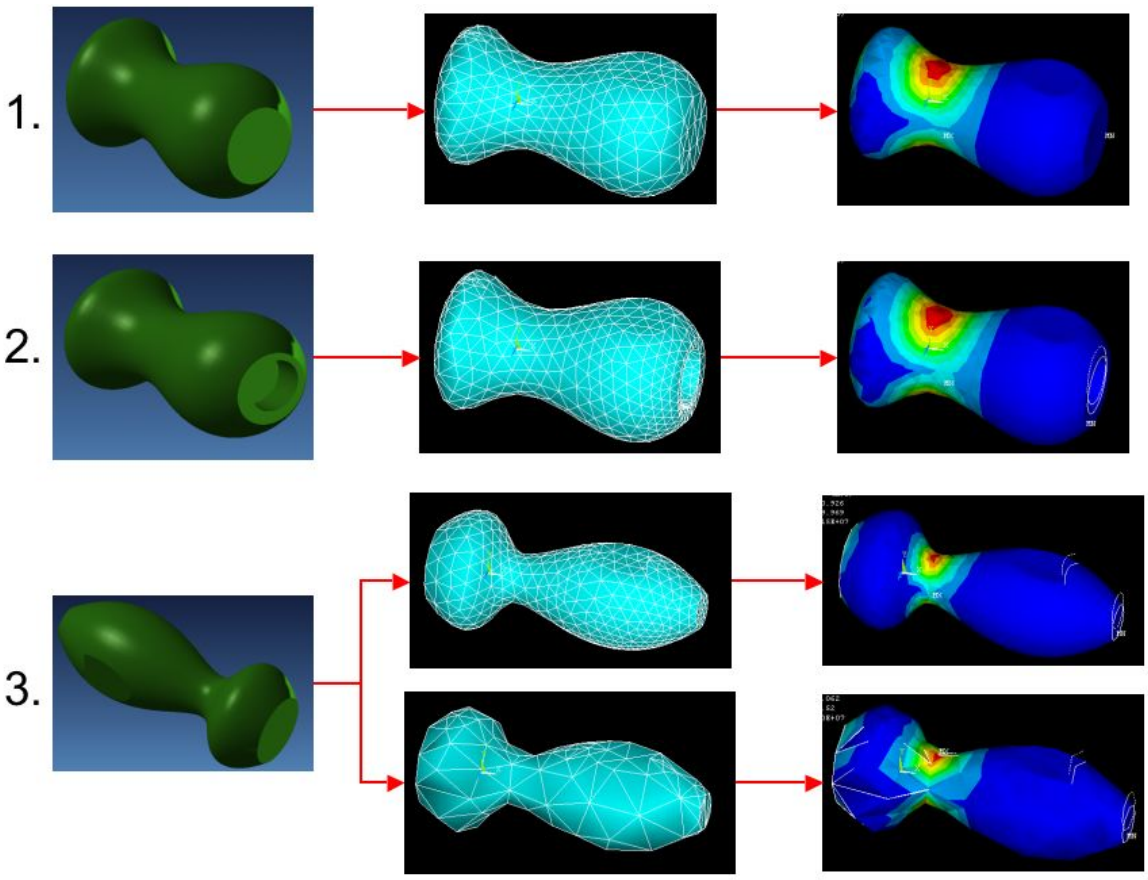


Figure 6.7 Results case study 2

The attribution tool performed as expected for each of the configurations tested. For configuration 2, in similar fashion to case study 1 configuration 7, it was expected that the persistent naming problem would force the CAD system to automatically assign attribute names without user input. For configuration 3, the attribution tool executed without error for both changes in the parametric geometry and changes in the mesh density attribute value. The overall performance of the attribution tool for case study 2 was evaluated by the aforementioned series of questions and the results are summarized in Table 6.5 below.

Table 6.5 Case study 2 performance characteristics

Did the attributes map correctly between Unigraphics and MySQL?	Yes
Did the translator program correctly automate the structural analysis in ANSYS?	Yes
Were attributes lost after parametric changes in the geometry?	No
Were attributes lost after changes in topology?	No
Were changes in attribute values accurately passed to the structural analysis?	Yes
How long did each automated analysis require, including preprocessing, meshing, solution, and post-processing?	~10 sec.

For this case study, all of the attributes applied via the GUI in UG were correctly stored in the MySQL attribute tables. The attribute translator program executed to completion for each configuration and no attributes were lost or incorrectly applied after changes in parametric geometry or topology. Attribute values modified in UG were accurately modified in the MySQL tables and passed into ANSYS. The time required to automatically perform the preprocessing, analysis, and post-processing was the same as that of the configurations used in case study 1. This is not surprising because the number of attributes applied and the number of elements used in the mesh were similar for each case.

6.2.3 Results Case Study 3: Discrete Feature Geometry

For this case study, two configurations were used to test the attribution tool's ability to automate procedures where discrete features are involved. Often times, when a user defines discrete features in CAD, he or she will want the same attributes applied to the parent discrete feature applied to the children discrete features. For this case study,

attributes were applied to the parent discrete feature, and then the discrete feature was copied relative to itself in a 2 x 2 rectangular array. The results of are graphically depicted in Figure 6.8 below.

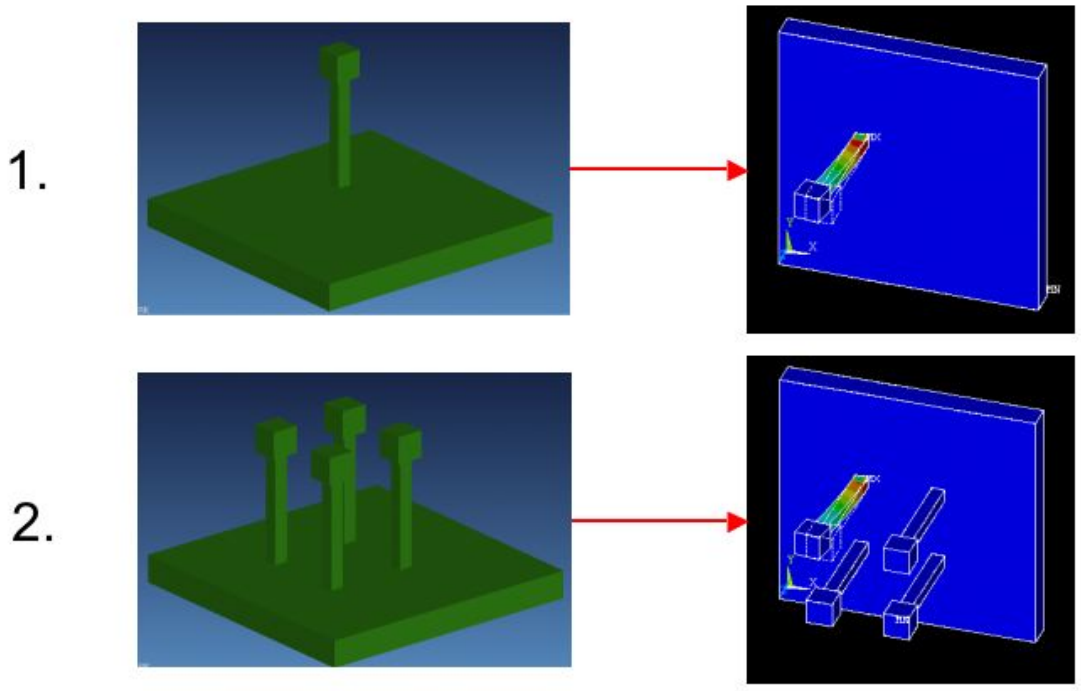


Figure 6.8 Results case study 3

As can be observed from Figure 6.8, the attributes applied to the parent discrete feature were not applied to the children. In order for the attributes to be copied to the children, functionality would have to be added to the GUI and object-oriented data structure that would query the CAD database for discrete features. If discrete features were detected having attributes applied to them, the GUI would then query the prompt the user to decide whether or not to copy the attributes of the parents. At the time this research was conducted, research was concurrently being conducted by King to explore methods of automatically applying attributes for discrete features from parent to child. This research

is complementary to King's work with discrete feature mapping and is complementary in nature. The addition of King's work to this work would certainly make the developed attribution tool more powerful when dealing with situations where there are large numbers of discrete features. Conversely, this research would complement King's work by bringing order through standardization to the attributes used in his work. For more information on discrete feature attribute mapping, interested readers are encouraged to review King (2004).

While attributes were not automatically copied from parent to child discrete features, the attribution tool developed for this research still accomplished its objective of automating structural analysis using attributes stored in MySQL. Table 6.7 summarizes the results of the questions posed to judge the performance of the attribution tool for this case study.

Table 6.6 Case study 3 performance characteristics

Did the attributes map correctly between Unigraphics and MySQL?	Yes
Did the translator program correctly automate the structural analysis in ANSYS?	Yes
Were attributes lost after parametric changes in the geometry?	No
Were attributes lost after changes in topology?	No
Were changes in attribute values accurately passed to the structural analysis?	Yes
How long did each automated analysis require, including preprocessing, meshing, solution, and post-processing?	~10 sec.

In spite of the attribution tool's inability to capture design intent regarding discrete features, it otherwise performed as expected, in similar fashion to case studies 1 and 2.

CHAPTER 7: CONCLUSIONS AND RECOMMENDATIONS

The objective of this research was to develop a method for applying and managing product attributes in the CAD environment in a standardized, customizable manner. Specifically, an entire CAD-centric attribution system was conceptually laid-out, including a hierarchal set of DBMS attribute tables, a code structure to manage attribute information between the DBMS tables and the CAD model, and a code structure to pass, translate, and automate attribute information from the DBMS to the CAE application.

The developed attribution tool interfaces directly with Unigraphics and is capable of the following operations:

- Application of attribute definition to specified CAD object and assign value if required
- Creation of new attribute definitions from the CAD environment and subsequent storage in DBMS attribute definition table
- Attribute extraction from DBMS to CAE application
- Attribute translation into CAE application commands
- Automation of CAE processing for all attributes pertaining to respective CAE application

Results for CAE automation of three geometric case studies in an optimization environment were collected. The completeness and validity of the CAD-centric attribute

application and CAE automation for several design iterations were assessed and reported for all tests performed. Conclusions were drawn concerning the strengths and weaknesses of the attribution tool for CAE automation in an optimization environment.

It was found that product attribute definitions can be stored in an external DBMS and applied directly to geometric entities in the CAD environment. The storage of attributes in an external DBMS is the first step to the creation of a language to communicate product attributes universally among CAE applications. With standardized attributes applied to parametric CAD geometry, the CAE applications themselves become parametric in nature. That is, the labor performed for CAE preprocessing is not lost for each parametric geometry change; rather, it is reused. For complex problems requiring analyses from several different CAE disciplines, reusable CAE preprocessing becomes extremely valuable.

Through the centralized product data model contained in the CAD file and DBMS attribute tables, multiple downstream CAE applications can operate in parallel. Furthermore, design changes recommended by the results from one downstream CAE analysis can be applied to the CAD geometry without destroying the preprocessing of another CAE discipline. The implications of this capability are twofold: 1. The concurrent engineering paradigm is achieved, and 2. Optimization and MDO can more readily become part of the design process.

7.1 Future Work

This thesis advances concurrent engineering through attribute standardization, but further effort remains before the developed attribution tool fully matures. Specifically, future solutions resulting from research of the persistent naming problem should be

integrated with the developed attribution tool. Due to the persistent naming problem, the attribution tool should not be used for optimization procedures wherein topological changes are introduced to geometric objects upon which attributes exist. In addition, integration of research done for the discrete feature problem would greatly enhance the power of the attribution tool. Certain optimization problems might include the number of discrete features as a design variable. Integration of the algorithms developed for discrete feature attributes would enable the attribution tool to automate optimization procedures for discrete features.

As mentioned in Chapter 5, the ability of the attribution tool to assign only a single value to an attribute is one of the weaknesses of the attribute table DBMS structure. It may prove more useful for future applications to restructure the DBMS tables such that the value for each attribute definition could be a matrix of values. This would make using the GUI more convenient as well as reducing the number of fields required to define a single attribute.

The attribute translator code developed for this research is CAE application specific. Development of a translator that is CAE application independent seems improbable at this point. However, if a standardized attribute language were universally recognized, it would be a simple task for CAE application developers to include an attribute translator code as an accessory to their software.

REFERENCES

Agonafer, D., and Vimba, A., (1997), "Solid Model Based Preprocessor to CFD Code for Applications to Electronic Cooling Systems," *Journal of Electronic Packaging*, Vol. 119, pp. 138-143.

Anderl, R. and Mengden, R., (1995), Parametric Design and its Impact on Solid Modeling Applications, *Proceedings of the 3rd Symposium on Solid Modeling Applications*, Salt Lake City, UT, December 1995, pp.1-12.

Anderl, R. and Mendgen, R., (1996), Modeling with constraints: theoretical foundation and application, *Computer-Aided Design*, Vol. 28, pp. 155-168.

Arabshahi, S., Barton, D.C., and Shaw, N.K., (1991), "Towards integrated design and analysis," *Finite Elements in Analysis and Design*, Vol. 9, pp. 271-293.

Arabshahi, S., Barton, D.C., and Shaw, N.K., (1993), "Steps Towards CAD-FEA Integration," *Engineering with Computers*, Vol. 9, pp. 17-26.

Ardalan, S., (2000), "DrawCraft: A Spacecraft Design Tool for Integrated Concurrent Engineering," *Aerospace Conference Proceedings*, IEEE, Vol. 11, pp. 501-510.

Ault, H., (1999), Using Geometric Constraints to Capture Design Intent, *Journal for Geometric and Graphics*, Vol. 3, No. 1, pp. 39-45.

Baily, M., Irani, R., Finnigan, P., Rohl, P., Badhrinath, K., (1999), "Integrated Multidisciplinary Design," American Institute of Aeronautics and Astronautics.

Balling, R.J., (2001), *Continuum Mechanics and Finite Element Analysis*, Brigham Young University, Provo, UT.

Bordoloi, B., Bock, D. (2004), *Oracle SQL*, Prentice Hall, Upper Saddle River, NJ.

Chandrupatla, T.R., Belegundu, A.D. (1996), *Introduction to Finite Elements in Engineering*, Prentice Hall, New York, NY.

Chappell, D., Trimble, J.H., (2002), *A Visual Introduction to SQL*, John Wiley & Sons, Inc., New York, NY.

Chen, C., Swift, F., Lee, S., Erg, R., Shen, Q., (1994), "Development of a Feature-Based and Object-Oriented Concurrent Engineering System," *Journal of Intelligent Manufacturing*, Vol. 5, pp. 23-31.

Cunha, R., Dias, A., (2002), "A Feature-Based Database Evolution Approach in the Design Process," *Robotics and Computer Integrated Manufacturing*, Vol. 18, pp. 275-281.

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

Gabbert, U., and Wehner, P., (1998), "The Product Data Model as a Pool for CAD-FEA Data," *Engineering with Computers*, Vol. 14, pp 115-122.

Gallagher, S., Monaghan, S., Irgens, C., (1997), "A Technique For Projecting the Product Model Throughout the Organization Via Commercially Available, Feature-Based CAD Systems," *Proceedings of the 1997 5th International Conference on Factory 2000 – The Technology Exploitation Process*, Cambridge, UK, Apr 2-4, 1997, pp. 337-345.

Hardwick, M., Spooner, D., Rando, T., Morris, K., (1996), "Sharing Manufacturing Information in Virtual Enterprises," *Communications of the ACM*, Vol. 39, No. 2, pp. 46-54.

Hoffman, C.M., Joan-Arinyo, R., (1998), "CAD and the Product Master Model," *Computer Aided Design*, Vol. 30, No. 11, pp. 905-918.

Hoffman, C.M. and Kim, K., (2001), Towards valid parametric CAD models, *Computer-Aided Design*, Vol. 33 No. 1, pp. 81-90.

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

Hoschek, J. and Dankwort, W., (1994), *Parametric and Variational Design*, B.G. Teubner Stuttgart, Germany, pp. 63-72.

Jensen, C.G., Flores, R. and Shelley, J., (2002), A web enabled process for accessing customized parametric designs, *Proceedings of ASME DETC 2002*, Montreal, Canada, Sept. 29-Oct. 2.

Jensen, C.G., Jones, C., Rohm III, T. and Tucker, S., (2000), Parametric engineering design tools and applications, *Proceedings of ASME Design Automation Conference*, Baltimore, MD, Sept. 10-13, pp. 657-664.

Jiang, Z., Harrison, D.K., Cheng, K., (2002), "An Integrated Concurrent Engineering Approach to the Design and Manufacture of Complex Components," *The International Journal of Advanced Manufacturing Technology*, Vol. 20, pp.319-325.

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

Krause, F., Luddermann, S., (1997), "Processing of CAD-Data – Conversion, Verification and Repair," *Solid Modeling*, Vol. 21, pp. 248-254.

Lee, K., (1999), *Principles of CAD/CAM/CAE Systems,* Addison Wesley, Reading, MA.

Ma, Y.S., Tong, T., (2003), "Associative Feature Modeling for Concurrent Engineering Integration," *Computers in Industry,* Vol. 51, pp. 51-71.

Marcheix, D., Pierra, G., (2002), "A Survey of the Persistent Naming Problem," *Proceedings of the Symposium on Solid Modeling and Applications,* Saarbrucken, Germany, June 17-21, 2002, pp. 13-22.

McMahon, C., Browne, J., (1998), *CADCAM: Principles, Practice, and Manufacturing Management,* Addison-Wesley, Reading, MA.

O'Bara, R., Beall, M., Shephard, M., (2002), "Attribution Management for Engineering Analysis," *Engineering with Computers,* Vol. 18, pp. 339-351.

Perng, D., Chang, C., (1996), "A New Feature-Based Design System with Dynamic Editing," *Computers Ind. Engineering,* Vol. 32, No. 2, pp. 383-397.

Prasad, B., (1996), *Concurrent Engineering Fundamentals Integrated Product and Process Organization,* Prentice Hall, Upper Saddle River, N.J.

Reddy, J., (1984), *An Introduction to the Finite Element Method,* McGraw-Hill, New York, N.Y.

Remondini, L., Leon, J.C., Trompette, P., (1998), *Towards an integrated architecture for the structural analysis of mechanical structures*, Computers & Structures, Vol. 67, Issue 5, pp. 299-307.

Rohm III, T., (2001), *Graphical Creation of CAD Parametric Application Programs,* M.S. Thesis, Brigham Young University.

Samareh, J.A., (1999), "Status and Future of Geometry Modeling and Grid Generation for Design and Optimization," *Journal of Aircraft,* Vol. 36, pp. 97-104.

Sederberg, T., (2002), "Computer Aided Geometric Design," Computer Science Department, Brigham Young University, Provo, UT. (http://tom.cs.byu.edu/~557/text/cagd.pdf; email: tom@byu.edu)

Shah, J. and Rangel, F., (2002), "Integration of commercial CAD/CAM system with custom CAPP using Orbix Middleware and CORBA standard," *Proceedings of ASME DETC 2002*, Montreal, Canada, Sept. 29-Oct. 2, 2002.

Shapiro, V. and Vossler, D.L., (1995), "What is a parametric family of solids?" *Proceedings of the 3rd Symposium on Solid Modeling Applications,* Salt Lake City, UT, December 1995, pp.43-54.

Shephard, M.S., (1988), "The Specification of Physical Attribute Information for Engineering Analysis," *Engineering with Computers*, Vol. 4, pp. 145-155.

Shephard, M.S., (2000), "Meshing environment for geometry-based analysis," *International Journal for Numerical Methods in Engineering*, Vol. 47, pp. 169-190.

Stroud, I., (1993), "Modelling Techniques for Handling Non-Geometric Information," *Proceedings of the 2nd Association for Computing Machinery Solid Modeling Conference*, Montreal, Canada, 1993, pp. 367-376.

Subrahmanyam, S., DeVries, W., and Pratt, M., (1995), "Feature Attributes and their Role in Product Modeling," *Proceedings of the Symposium on Solid Modeling and Applications*, Salt Lake City, UT, May 17-19, 1995, pp. 115-124.

Sutherland, I., (1963), *Sketchpad, A Man-Machine Graphical Communication System.* PhD thesis, Massachusetts Institute of Technology.

Tucker, S., (2000), *Parametric Application Program Interface Function Mapping of Computer-Aided Design Systems*, M.S. Thesis, Brigham Young University.

Wilson, M.G., (2004), *Integration of Rapid Prototyping Preprocessing Operations with a Commercial CAD System*, M.S. Thesis, Brigham Young University.

Zeid, I., (1991), *CAD/CAM Theory and Practice*, McGraw-Hill, New York, NY.

Zeid, I., (2005), *Mastering CAD/CAM*, McGraw-Hill

Appendix

Appendix A: *UI Styler GUI Header Files and Source Code*

```
//File:        Attrdlg.hpp
//Author: Tyson Baker
//Date:        April 2004

#ifndef ATTRDLG_H_INCLUDED
#define ATTRDLG_H_INCLUDED

#include <uf.h>
#include <uf_defs.h>
#include <uf_styler.h>


#ifdef __cplusplus
extern "C" {
#endif

int CALL_ATTR_DLG();

/*------------------ UIStyler Dialog Definitions  ------------------- */
/* The following values are definitions into your UIStyler dialog.    */
/* These values will allow you to modify existing objects within your */
/* dialog.   They work directly with the UG/Open API,                 */
/* UF_STYLER_ask_value, UF_STYLER_ask_values, and UF_STYLER_set_value.*/
/*-------------------------------------------------------------------- */

#define ATTR_PAGE_BEG_0               ("PAGE_BEG_0")
#define ATTR_OPTION_DOMAIN            ("OPTION_DOMAIN")
#define ATTR_ACTION_NEW_DOMAIN        ("ACTION_NEW_DOMAIN")
#define ATTR_ACTION_REFRESH           ("ACTION_REFRESH")
#define ATTR_OPTION_SUB_DOMAIN        ("OPTION_SUB_DOMAIN")
#define ATTR_OPTION_TITLE             ("OPTION_TITLE")
#define ATTR_STR_TYPE                 ("STR_TYPE")
#define ATTR_ACTION_HELP              ("ACTION_HELP")
#define ATTR_STR_VALUE                ("STR_VALUE")
#define ATTR_ACTION_MULTIOBJECTS      ("ACTION_MULTIOBJECTS")
#define ATTR_ACTION_CLEAR_OBJ         ("ACTION_CLEAR_OBJ")
#define ATTR_LIST_MULTIOBJS           ("LIST_MULTIOBJS")
#define ATTR_PAGE_END_1               ("PAGE_END_1")
#define ATTR_PAGE_BEG_2               ("PAGE_BEG_2")
#define ATTR_RC_BEG_19                ("RC_BEG_19")
#define ATTR_LABEL_25                 ("LABEL_25")
#define ATTR_LABEL_26                 ("LABEL_26")
#define ATTR_LABEL_TITLE              ("LABEL_TITLE")
#define ATTR_LABEL_VALUE              ("LABEL_VALUE")
#define ATTR_LABEL_OBJ_NAME           ("LABEL_OBJ_NAME")
#define ATTR_LIST_DOMAIN              ("LIST_DOMAIN")
#define ATTR_LIST_SUB_DOMAIN          ("LIST_SUB_DOMAIN")
#define ATTR_LIST_TITLE               ("LIST_TITLE")
#define ATTR_LIST_VALUE               ("LIST_VALUE")
#define ATTR_LIST_OBJ_NAME            ("LIST_OBJ_NAME")
#define ATTR_RC_END_20                ("RC_END_20")
#define ATTR_ACTION_DELETE            ("ACTION_DELETE")
#define ATTR_PAGE_END_3               ("PAGE_END_3")
#define ATTR_DIALOG_OBJECT_COUNT      ( 28 )


/*---------------- UIStyler Callback Prototypes --------------- */
/* The following function prototypes define the callbacks       */
/* specified in your UIStyler built dialog.  You are REQUIRED to*/
/* create the associated function for each prototype.  You must */
/* use the same function name and parameter list when creating  */
/* your callback function.                                      */
/*-------------------------------------------------------------- */

int ATTR_constructor_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data);

int ATTR_destructor_cb ( int dialog_id,
```

```
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int ATTR_ok_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int ATTR_apply_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int ATTR_cancel_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int ATTR_domain_list_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int ATTR_new_domain_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int ATTR_refresh_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int ATTR_sub_domain_list_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int ATTR_title_list_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int ATTR_help_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int ATTR_action_multiobjs ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int ATTR_Clear_Obj_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int ATTR_value_edit_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int ATTR_value_highlight_obj_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);

int ATTR_delete_cb ( int dialog_id,
            void * client_data,
            UF_STYLER_item_value_type_p_t callback_data);



}
#endif


#endif /* ATTRDLG_H_INCLUDED */
#ifdef __cplusplus



//File:        Attrdlg.cpp
//Author: Tyson Baker
//Date:        April 2004


/* These include files are needed for the following template code.           */
#include "UGOpenMaster.hpp"
```

```
#include "Attrdlg.h"
#include "UIStylerFuncs.hpp"
#include "Library.hpp"
#include "Utilities.hpp"
#include "Newdomain.h"
#include <mysql.h>
#include "Attr.h"


Library glib;
Attribute gatt;
extern MYSQL *conn;
//extern vector<double> dbvals;

/* The following definition defines the number of callback entries */
/* in the callback structure:                                      */
/* UF_STYLER_callback_info_t ATTR_cbs */
#define ATTR_CB_COUNT ( 16 + 1 ) /* Add 1 for the terminator */

/*-------------------------------------------------------------------------
The following structure defines the callback entries used by the
styler file.  This structure MUST be passed into the user function,
UF_STYLER_create_dialog along with ATTR_CB_COUNT.
-------------------------------------------------------------------------*/
static UF_STYLER_callback_info_t ATTR_cbs[ATTR_CB_COUNT] =
{
 {UF_STYLER_DIALOG_INDEX, UF_STYLER_CONSTRUCTOR_CB  , 0, ATTR_constructor_cb},
 {UF_STYLER_DIALOG_INDEX, UF_STYLER_DESTRUCTOR_CB   , 0, ATTR_destructor_cb},
 {UF_STYLER_DIALOG_INDEX, UF_STYLER_OK_CB           , 0, ATTR_ok_cb},
 {UF_STYLER_DIALOG_INDEX, UF_STYLER_APPLY_CB        , 0, ATTR_apply_cb},
 {UF_STYLER_DIALOG_INDEX, UF_STYLER_CANCEL_CB       , 0, ATTR_cancel_cb},
 {ATTR_OPTION_DOMAIN    , UF_STYLER_ACTIVATE_CB     , 0, ATTR_domain_list_cb},
 {ATTR_ACTION_NEW_DOMAIN, UF_STYLER_ACTIVATE_CB     , 1, ATTR_new_domain_cb},
 {ATTR_ACTION_REFRESH   , UF_STYLER_ACTIVATE_CB     , 0, ATTR_refresh_cb},
 {ATTR_OPTION_SUB_DOMAIN, UF_STYLER_ACTIVATE_CB     , 0, ATTR_sub_domain_list_cb},
 {ATTR_OPTION_TITLE     , UF_STYLER_ACTIVATE_CB     , 0, ATTR_title_list_cb},
 {ATTR_ACTION_HELP      , UF_STYLER_ACTIVATE_CB     , 0, ATTR_help_cb},
 {ATTR_ACTION_MULTIOBJECTS, UF_STYLER_ACTIVATE_CB   , 1, ATTR_action_multiobjs},
 {ATTR_ACTION_CLEAR_OBJ , UF_STYLER_ACTIVATE_CB     , 0, ATTR_Clear_Obj_cb},
 {ATTR_LIST_VALUE       , UF_STYLER_ACTIVATE_CB     , 0, ATTR_value_highlight_obj_cb},
 {ATTR_LIST_VALUE       , UF_STYLER_DOUBLE_CLICK_CB , 1, ATTR_value_edit_cb},
 {ATTR_ACTION_DELETE    , UF_STYLER_ACTIVATE_CB     , 0, ATTR_delete_cb},
 {UF_STYLER_NULL_OBJECT, UF_STYLER_NO_CB, 0, 0 }
};

/*-------------------------------------------------------------------------
UF_MB_styler_actions_t contains 4 fields.  These are defined as follows:

Field 1 : the name of your dialog that you wish to display.
Field 2 : any client data you wish to pass to your callbacks.
Field 3 : your callback structure.
Field 4 : flag to inform menubar of your dialog location.  This flag MUST
          match the resource set in your dialog!  Do NOT ASSUME that changing
          this field will update the location of your dialog.  Please use the
          UIStyler to indicate the position of your dialog.
-------------------------------------------------------------------------*/
static UF_MB_styler_actions_t actions[] = {
    { "Attrdlg.dlg",  NULL,   ATTR_cbs,  UF_MB_STYLER_IS_NOT_TOP },
    { NULL,  NULL,  NULL,  0 } /* This is a NULL terminated list */
};


/*---------------- MENUBAR HOOKUP HELP Example -------------------

-------------------------------------------------------------*/
#ifdef MENUBAR_COMMENTED_OUT
extern void ufsta (char *param, int *retcode, int rlen)
{
    int  error_code;

    if ( (UF_initialize()) != 0)
          return;

    if ( (error_code = UF_MB_add_styler_actions ( actions ) ) != 0 )
    {
          char fail_message[133];

          UF_get_fail_message(error_code, fail_message);
          printf ( "%s\n", fail_message );
    }
```

```
    UF_terminate();
    return;
}
#endif /*MENUBAR_COMMENTED_OUT*/


/*-------DIALOG CREATION FROM A CALLBACK HELP Example ----------

------------------------------------------------------------------*/

//#ifdef DISPLAY_FROM_CALLBACK
extern int CALL_ATTR_DLG ( )
{
    int  error_code = 0;
    int response;
    if ( ( error_code = UF_initialize() ) != 0 )
           return (0) ;

    //string dialog = get_comb_dialog_dir("Attrdlg.dlg");

    if ( ( error_code = UF_STYLER_create_dialog ( "C:\\Attribution\\Attrdlg.dlg",
           ATTR_cbs,      /* Callbacks from dialog */
           ATTR_CB_COUNT, /* number of callbacks*/
           NULL,        /* This is your client data */
           &response ) ) != 0 )
    {
          char fail_message[133];

          /* Get the user function fail message based on the fail code.*/
          UF_get_fail_message(error_code, fail_message);
          UF_UI_set_status (fail_message);
          printf ( "%s\n", fail_message );
    }


    UF_terminate();
    return (error_code);
}
//#endif /* DISPLAY_FROM_CALLBACK */



/*-------DIALOG CREATION FROM A USER EXIT HELP Example --------

------------------------------------------------------------------*/

#ifdef DISPLAY_FROM_USER_EXIT
extern void <enter a valid user exit here> (char *param, int *retcode, int rlen)
{
    int  response   = 0;
    int  error_code = 0;

    if ( ( UF_initialize() ) != 0 )
           return;

    if ( ( error_code = UF_STYLER_create_dialog ( "Attrdlg.dlg",
           ATTR_cbs,      /* Callbacks from dialog */
           ATTR_CB_COUNT, /* number of callbacks*/
           NULL,        /* This is your client data */
           &response ) ) != 0 )
    {
          char fail_message[133];

          /* Get the user function fail message based on the fail code.*/
          UF_get_fail_message(error_code, fail_message);
          UF_UI_set_status (fail_message);
          printf ( "%s\n", fail_message );
    }


    UF_terminate();
    return;
}


/*-------------------------------------------------------------------------

-------------------------------------------------------------------------*/

extern int ufusr_ask_unload (void)
```

```
{
     /* unload immediately after application exits*/
     return ( UF_UNLOAD_IMMEDIATELY );

     /*via the unload selection dialog... */
     /*return ( UF_UNLOAD_SEL_DIALOG );   */
     /*when UG terminates...              */
     /*return ( UF_UNLOAD_UG_TERMINATE ); */
}

/*---------------------------------------------------------------------------

---------------------------------------------------------------------------*/
extern void ufusr_cleanup (void)
{
    return;
}
#endif /* DISPLAY_FROM_USER_EXIT */




/*****************************************************************************/
/*****************************************************************************/
/*---------------------------------------------------------------------------*/
/*---------------------- UIStyler Callback Functions ------------------------*/
/*---------------------------------------------------------------------------*/
/*****************************************************************************/
/*****************************************************************************/

/* ---------------------------------------------------------------------------
 * Callback Name: ATTR_constructor_cb
 * ---------------------------------------------------------------------------*/
int ATTR_constructor_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     /* Make sure User Function is available. */
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

  /* ---- Enter your callback code here ----- */

      char path_and_name[257];
      tag_t part = UF_PART_ask_display_part ( );
      UF_CALL(UF_PART_ask_part_name (part, path_and_name));

      char file_name[132], file_path[132];
      FilestringDecomp(path_and_name/*257*/, file_name/*132*/, file_path/*132*/);

      cout<<"\n\nfile_name: "<<file_name<<"\nfile_path: "<<file_path<<endl;

      char* slashes = add_slashes((char*)path_and_name);
      gatt.SetPartName(file_name);

      cout<<"after ask and set part name..."<<endl;


//connect to sql server using "attributes" database (check utilities.cpp for more info)
      sql_connect();
      printf("\nYou are currently connected to MySQL server version: %s\n",
        mysql_get_server_info(conn));

//get the domains column
      vector<string> domainAll = select_string_query("select domain from library");

//erase the repeated domains
      vector <string> domainsOne = singlize(domainAll);
//set vector of domains for library data structure
      glib.setDomainsList(domainsOne);

//fill up the UIS option box with the values in domainsOne
      UIS_setListString(dialog_id, "OPTION_DOMAIN", glib.getDomainsList(), 0, true);

//insert the part into the parts table using a query
      vector<string> parts = select_string_query("select part_name from parts");
      char whatever[222];
      sprintf(whatever, "insert into parts values ('', '%s')", file_name);
      cout<<"whatever: "<<whatever<<endl;
      int numrepeat=0;
```

```
      //check and make sure the part isn't already in the table
      for (int i=0; i<parts.size(); i++)
      {
           if(parts[i]==file_name)
           {
                numrepeat++;
           }
      }

      //if part name is not already in the parts table, then add it
      if (numrepeat==0)
      {
          int check = mysql_query(conn, (char*)whatever);   //0=success, 1=failure
      }


//if any attributes exist for this part, get them from the database and store them in the
data structure
      string query2 = "select parts.id from parts where part_name ='";
      query2.append(gatt.GetPartName());          //gatt.GetPartName() set above in this
function
      query2.append("'");
      vector<int> partid = select_int_query((char*)query2.c_str());
      gatt.SetPartID(partid[0]);

      //check for consistency in the object names in the sql database and in the CAD
database
      char what[133];
      sprintf(what, "select object_name from part_attributes where part_id = %d",
gatt.GetPartID());
      vector<string>objnames = select_string_query(what);

      for (i=0; i<objnames.size(); i++)
      {
           tag_t objtag = NULL_TAG;
           int check = UF_OBJ_cycle_by_name((char*)objnames[i].c_str(), &objtag);

           if(objtag==0)  //if objtag==0, then there's no object w/ that name, thus DELETE
entry from sql database
           {
                sprintf(what, "delete from part_attributes where object_name = '%s'",
(char*)objnames[i].c_str());  //SQL DELETE
                cout<<what<<endl;
                mysql_query(conn, what);
           }
      }

      //fill up vectors containing libid, value, object_name

      objnames.clear();
      vector<string> values, libids;

      sprintf(what, "select object_name from part_attributes where part_id = %d",
gatt.GetPartID());
      objnames = select_string_query(what);
      gatt.SetObjNames(objnames);


      sprintf(what, "select lib_id from part_attributes where part_id = %d",
gatt.GetPartID());
      libids = select_string_query(what);
      gatt.SetLibids(libids);

      sprintf(what, "select value from part_attributes where part_id = %d",
gatt.GetPartID());
      values = select_string_query(what);
      gatt.SetValues(values);


      //fill up the UIS fields on page 2
      UIS_setListString(dialog_id, "LIST_VALUE", gatt.GetValues(), 0, TRUE);
      UIS_setListString(dialog_id, "LIST_OBJ_NAME", gatt.GetObjNames(), 0, TRUE);

      vector<string> xdomains, ydomains, xsubs, ysubs, xtitle, ytitle;

      for(i=0; i<libids.size(); i++)
      {
           sprintf(what, "select domain from library where library.id = %s",
(char*)libids[i].c_str());
           xdomains = select_string_query(what);
           ydomains.push_back(xdomains[0]);
```

```
          sprintf(what, "select subdomain from library where library.id = %s",
(char*)libids[i].c_str());
          xsubs = select_string_query(what);
          ysubs.push_back(xsubs[0]);
          sprintf(what, "select title from library where library.id = %s",
(char*)libids[i].c_str());
          xtitle = select_string_query(what);
          ytitle.push_back(xtitle[0]);
     }

     UIS_setListString(dialog_id, "LIST_DOMAIN", ydomains, 0, TRUE);
     UIS_setListString(dialog_id, "LIST_SUB_DOMAIN", ysubs, 0, TRUE);
     UIS_setListString(dialog_id, "LIST_TITLE", ytitle, 0, TRUE);

     gatt.SetDomains(ydomains);
     gatt.SetSubdomains(ysubs);
     gatt.SetTitles(ytitle);



    UF_terminate ();

   /* Callback acknowledged, do not terminate dialog */
   return (UF_UI_CB_CONTINUE_DIALOG);
   /* A return value of UF_UI_CB_EXIT_DIALOG will not be accepted    */
   /* for this callback type.  You must continue dialog construction.*/

}


/* ------------------------------------------------------------------------
 * Callback Name: ATTR_destructor_cb

 * ------------------------------------------------------------------------*/
int ATTR_destructor_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
    /* Make sure User Function is available. */
    if ( UF_initialize() != 0)
         return ( UF_UI_CB_CONTINUE_DIALOG );

    /* ---- Enter your callback code here ----- */

    UF_terminate ();

   /* Callback acknowledged, do not terminate dialog.              */
   /* A return value of UF_UI_CB_EXIT_DIALOG will not be accepted  */
   /* for this callback type.  You must continue dialog destruction*/
   return (UF_UI_CB_CONTINUE_DIALOG);

}


/* ------------------------------------------------------------------------
 * Callback Name: ATTR_ok_cb

 * ------------------------------------------------------------------------*/
int ATTR_ok_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
    /* Make sure User Function is available. */
    if ( UF_initialize() != 0)
         return ( UF_UI_CB_CONTINUE_DIALOG );

    /* ---- Enter your callback code here ----- */

     mysql_close(conn);
    UF_terminate ();

   /* Callback acknowledged, terminate dialog             */
   /* It is STRONGLY recommended that you exit your       */
   /* callback with UF_UI_CB_EXIT_DIALOG in a ok callback.*/
   /* return ( UF_UI_CB_EXIT_DIALOG );                    */
   return (UF_UI_CB_EXIT_DIALOG);

}


/* ------------------------------------------------------------------------
```

```
 * Callback Name: ATTR_apply_cb

 * -------------------------------------------------------------------------*/
int ATTR_apply_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     /* Make sure User Function is available. */
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

     /* ---- Enter your callback code here ----- */

//get the values from domain, subdomain, title and type fields
      int whichsub = UIS_getIntValue(dialog_id, "OPTION_SUB_DOMAIN");
      int whichdom = UIS_getIntValue(dialog_id, "OPTION_DOMAIN");
      int whichtitle = UIS_getIntValue(dialog_id, "OPTION_TITLE");
      string title = glib.getTitleList()[whichtitle];
      string domain = glib.getDomainsList()[whichdom];
      string subdomain = glib.getSubDomainList()[whichsub];
      char* type = UIS_getStringValue(dialog_id, "STR_TYPE");

      vector<string> monames = gatt.GetMultiObjNames();
     //run a loop to create an attribute for every object name in mMultiObjNames
      for (int i=0; i<monames.size(); i++)
      {
               //get the lib_id of the selected attribute
                string query = "SELECT library.id FROM library where subdomain = '";
                query.append(subdomain);
                query.append("' and type = '");
                query.append(type);
                query.append("' and domain = '");
                query.append(domain);
                query.append("' and title = '");
                query.append(title);
                query.append("'");

                vector<int> libid = select_int_query((char*)query.c_str());
                gatt.SetLibID(libid[0]);


               //insert record into part_attributes (id, part_id, lib_id, value,
object_name)
                string query3 = "insert into part_attributes values(NULL, ";

                char charpid[66];
                char charlid[66];
                itoa(gatt.GetLibID(), charlid, 10);
                itoa(gatt.GetPartID(), charpid, 10);
                query3.append(charpid);
                query3.append(", ");
                query3.append(charlid);

                if (glib.getType()=="string"
||glib.getType()=="integer"||glib.getType()=="double")
                {
                     query3.append(", '");
                     char* vstr =  UIS_getStringValue(dialog_id, "STR_VALUE");
                     gatt.SetValue(vstr);

                     query3.append(vstr);
                     query3.append("', '");
                }

                else
                {
                     query3.append(",'', '");
                     gatt.SetValue("");
                }

                query3.append(monames[i]);
                query3.append("')");

                cout<<query3<<endl;
                mysql_query(conn, (char*)query3.c_str());//***************** DO QUERY -
with object name



                //add the applied attribute to the list of attributes on page 2
                vector<string> ydomains = gatt.GetDomains();
```

```
                vector<string> ysubs = gatt.GetSubdomains();
                vector<string> ytitles = gatt.GetTitles();
                vector<string> yvalues = gatt.GetValues();
                vector<string> yobjnames = gatt.GetObjNames();

                //add the new attribute to the vectors in the data structure
                ydomains.push_back(domain);
                ysubs.push_back(subdomain);
                ytitles.push_back(title);
                yvalues.push_back(gatt.GetValue());
                //yobjnames.push_back(gatt.GetObjectName());
                yobjnames.push_back(monames[i]);

                //set the values in the data structure
                gatt.SetDomains(ydomains);
                gatt.SetSubdomains(ysubs);
                gatt.SetTitles(ytitles);
                gatt.SetValues(yvalues);
                gatt.SetObjNames(yobjnames);

                //fill in the selection boxes
                UIS_setListString(dialog_id, "LIST_DOMAIN", ydomains, 0, TRUE);
                UIS_setListString(dialog_id, "LIST_SUB_DOMAIN", ysubs, 0, TRUE);
                UIS_setListString(dialog_id, "LIST_TITLE", ytitles, 0, TRUE);
                UIS_setListString(dialog_id, "LIST_VALUE", yvalues, 0, TRUE);
                UIS_setListString(dialog_id, "LIST_OBJ_NAME", yobjnames, 0, TRUE);

           }


     UF_terminate ();

    /* Callback acknowledged, do not terminate dialog                */
    /* A return value of UF_UI_CB_EXIT_DIALOG will not be accepted    */
    /* for this callback type.  You must respond to your apply button.*/
    return (UF_UI_CB_CONTINUE_DIALOG);

}


/* ------------------------------------------------------------------------
 * Callback Name: ATTR_cancel_cb

 * ------------------------------------------------------------------------*/
int ATTR_cancel_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     /* Make sure User Function is available. */
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

     /* ---- Enter your callback code here ----- */

      mysql_close(conn);
     UF_terminate ();

    /* Callback acknowledged, terminate dialog             */
    /* It is STRONGLY recommended that you exit your       */
    /* callback with UF_UI_CB_EXIT_DIALOG in a cancel call */
    /* back rather than UF_UI_CB_CONTINUE_DIALOG.          */
    return ( UF_UI_CB_EXIT_DIALOG );

}


/* ------------------------------------------------------------------------
 * Callback Name: ATTR_domain_list_cb**********************************

 * ------------------------------------------------------------------------*/
int ATTR_domain_list_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     /* Make sure User Function is available. */
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

     /* ---- Enter your callback code here ----- */
```

```
      vector <string> domains, subdomains;   // domains initially set in constructor_cb
      domains = glib.getDomainsList();


//get the int associated w/ the selected domain option
      int which = UIS_getIntValue(dialog_id, "OPTION_DOMAIN");
      string query = "select library.subdomain from library where library.domain = '";
      query.append(domains[which]);
      query.append("'");

// get the list of all the subdomains from the subdomain column in the library table
      vector<string> subdomainAll = select_string_query((char*)query.c_str());

      //throw away the repeated subdomains
      vector<string> subdomainOne = singlize(subdomainAll);

      glib.setSubDomainList(subdomainOne);

      UIS_setListString(dialog_id, "OPTION_SUB_DOMAIN", glib.getSubDomainList(), 0, true);


     UF_terminate ();

    /* Callback acknowledged, do not terminate dialog */
    return (UF_UI_CB_CONTINUE_DIALOG);

    /* or Callback acknowledged, terminate dialog.    */
    /* return ( UF_UI_CB_EXIT_DIALOG );               */

}


/* ------------------------------------------------------------------------
 * Callback Name: ATTR_new_domain_cb==================================>>>>>>

 * ------------------------------------------------------------------------*/
int ATTR_new_domain_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     /* Make sure User Function is available. */
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

     /* ---- Enter your callback code here ----- */

      CALL_NEWDOMAIN_DLG();

     UF_terminate ();

    /* Callback acknowledged, do not terminate dialog */
    return (UF_UI_CB_CONTINUE_DIALOG);

    /* or Callback acknowledged, terminate dialog.    */
    /* return ( UF_UI_CB_EXIT_DIALOG );               */

}

/* ------------------------------------------------------------------------
 * Callback Name: ATTR_refresh_cb

 * ------------------------------------------------------------------------*/
int ATTR_refresh_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     /* Make sure User Function is available. */
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

     /* ---- Enter your callback code here ----- */

//get the domains column
      vector<string> domainAll = select_string_query("select domain from library");

//erase the repeated domains
      vector <string> domainsOne = singlize(domainAll);
//set vector of domains for library data structure
      glib.setDomainsList(domainsOne);

//fill up the UIS option box with the values in domainsOne
```

```
      UIS_setListString(dialog_id, "OPTION_DOMAIN", glib.getDomainsList(), 0, true);


     UF_terminate ();

    /* Callback acknowledged, do not terminate dialog */
    return (UF_UI_CB_CONTINUE_DIALOG);

    /* or Callback acknowledged, terminate dialog.    */
    /* return ( UF_UI_CB_EXIT_DIALOG );               */

}


/* ------------------------------------------------------------------------
 * Callback Name: ATTR_sub_domain_list_cb**********************************

 * ------------------------------------------------------------------------*/
int ATTR_sub_domain_list_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     /* Make sure User Function is available. */
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

     /* ---- Enter your callback code here ----- */

      int which = UIS_getIntValue(dialog_id, "OPTION_SUB_DOMAIN");
      int whichdom = UIS_getIntValue(dialog_id, "OPTION_DOMAIN");
      vector <string> domains, subdomains, titles;

      domains = glib.getDomainsList();
      subdomains = glib.getSubDomainList();

      string query = "select title from library where domain = '";
      query.append(domains[whichdom]);
      query.append("' and subdomain = '");
      query.append(subdomains[which]);
      query.append("'");

      titles = select_string_query((char*)query.c_str());

      glib.setTitlesList(titles);

      UIS_setListString(dialog_id, "OPTION_TITLE", glib.getTitleList(), 0, true);

     UF_terminate ();

    /* Callback acknowledged, do not terminate dialog */
    return (UF_UI_CB_CONTINUE_DIALOG);

    /* or Callback acknowledged, terminate dialog.    */
    /* return ( UF_UI_CB_EXIT_DIALOG );               */

}


/* ------------------------------------------------------------------------
 * Callback Name: ATTR_title_list_cb
 * ------------------------------------------------------------------------*/
int ATTR_title_list_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     /* Make sure User Function is available. */
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

     /* ---- Enter your callback code here ----- */
      int whichsub = UIS_getIntValue(dialog_id, "OPTION_SUB_DOMAIN");
      int whichdom = UIS_getIntValue(dialog_id, "OPTION_DOMAIN");
      int whichtitle = UIS_getIntValue(dialog_id, "OPTION_TITLE");
      vector <string> domains, subdomains, titles;

      domains = glib.getDomainsList();
      subdomains = glib.getSubDomainList();
      titles = glib.getTitleList();

      string query = "select type from library where domain = '";
      query.append(domains[whichdom]);
```

```
      query.append("' and subdomain = '");
      query.append(subdomains[whichsub]);
      query.append("' and title = '");
      query.append(titles[whichtitle]);
      query.append("'");

      vector<string> types = select_string_query((char*) query.c_str());
      glib.setType(types[0]);


      //UIS_setStringValue(dialog_id, "STR_TYPE", (char*)types[0].c_str());
      UIS_setStringValue(dialog_id, "STR_TYPE", (char*)glib.getType().c_str());

      //set the sensitivity of the values based on type
      string type = glib.getType();
      if (type == "integer")
      {
          UIS_setSingleSens(dialog_id, "INT_VALUE", 0);
          UIS_setSingleSens(dialog_id, "STR_VALUE", 1);
          UIS_setSingleSens(dialog_id, "REAL_VALUE", 0);
      }
      else if (type == "string")
      {
          UIS_setSingleSens(dialog_id, "STR_VALUE", 1);
          UIS_setSingleSens(dialog_id, "REAL_VALUE", 0);
          UIS_setSingleSens(dialog_id, "INT_VALUE", 0);
      }
      else if (type == "double")
      {
          UIS_setSingleSens(dialog_id, "REAL_VALUE", 0);
          UIS_setSingleSens(dialog_id, "INT_VALUE", 0);
          UIS_setSingleSens(dialog_id, "STR_VALUE", 1);
      }
      else
      {
           UIS_setSingleSens(dialog_id, "REAL_VALUE", 0);
          UIS_setSingleSens(dialog_id, "INT_VALUE", 0);
          UIS_setSingleSens(dialog_id, "STR_VALUE", 0);
      }

     UF_terminate ();

    /* Callback acknowledged, do not terminate dialog */
    return (UF_UI_CB_CONTINUE_DIALOG);

    /* or Callback acknowledged, terminate dialog.    */
    /* return ( UF_UI_CB_EXIT_DIALOG );               */

}



/* ------------------------------------------------------------------------
 * Callback Name: ATTR_Clear_Obj_cb

 * ------------------------------------------------------------------------*/

int ATTR_Clear_Obj_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     /* Make sure User Function is available. */
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

     /* ---- Enter your callback code here ----- */

      vector<string> junk = gatt.GetMultiObjNames();
      junk.clear();
      gatt.SetMultiObjNames(junk);
      UIS_clearList( dialog_id, "LIST_MULTIOBJS");

     UF_terminate ();

    /* Callback acknowledged, do not terminate dialog */
    return (UF_UI_CB_CONTINUE_DIALOG);

    /* or Callback acknowledged, terminate dialog.    */
    /* return ( UF_UI_CB_EXIT_DIALOG );               */

}
```

```
/* ------------------------------------------------------------------------
   Callback Name:  ATTR_value_edit_cb
 * ------------------------------------------------------------------------*/
int ATTR_value_edit_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     /* Make sure User Function is available. */
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

     /* ---- Enter your callback code here ----- */

      //on double-click, bring up a box that allows you to change the value
      vector <int> which = UIS_getListInt(dialog_id, "LIST_VALUE");
      vector<string> values = gatt.GetValues();

      vector<string> domains = gatt.GetDomains();
      vector<string> subdomains = gatt.GetSubdomains();
      vector<string> titles = gatt.GetTitles();
      vector<string> objnames = gatt.GetObjNames();

      char libidquery[222];
      sprintf(libidquery, "select id from library where domain = '%s' and subdomain = '%s'
and title = '%s'",
            (char*)domains[which[0]].c_str(), (char*)subdomains[which[0]].c_str(),
(char*)titles[which[0]].c_str());
      //cout<<"libidquery: "<<libidquery<<endl;

      vector<int> id = select_int_query(libidquery);
      //cout<<"id[0]: "<<id[0]<<endl;

      char paidquery[222];
      sprintf(paidquery, "select id from part_attributes where object_name = '%s' and
lib_id = %d and value = '%s' and part_id = %d",
            (char*)objnames[which[0]].c_str(), id[0], (char*)values[which[0]].c_str(),
gatt.GetPartID());
      vector<int> paid = select_int_query(paidquery);
      //cout<<"paid[0]: "<<paid[0]<<", paid.size: "<<paid.size()<<endl;

      int ir3=0;
      char ca1[144];
      sprintf(ca1, "Enter new value");
      string ca2 = values[which[0]];
      int resp = UF_CALL(uc1600 (ca1,  (char*)ca2.c_str(), &ir3 ));
      cout<<"ca2new: "<<ca2<<endl;
      values[which[0]]=ca2;
      gatt.SetValues(values);
      UIS_setListString(dialog_id, "LIST_VALUE", gatt.GetValues(), 0, TRUE);

      char valquery[222];
      sprintf(valquery, "select value from part_attributes where lib_id = %d and
object_name = '%s' and part_id = %d",
           id[0], (char*)objnames[which[0]].c_str(), gatt.GetPartID());
      //cout<<"valquery: "<<valquery<<endl;
      char valupdate[222];
      sprintf(valupdate, "update part_attributes set value = '%s' where id=%d",
(char*)ca2.c_str(), paid[0]);
      mysql_query(conn, (char*)valupdate);


      vector<string> value = select_string_query(valquery);
      cout<<"dbvalue: "<<value[0]<<endl;

     UF_terminate ();

    /* Callback acknowledged, do not terminate dialog */
    return (UF_UI_CB_CONTINUE_DIALOG);

    /* or Callback acknowledged, terminate dialog.  */
    /* return ( UF_UI_CB_EXIT_DIALOG );             */

}

/* ------------------------------------------------------------------------
 * Callback Name: ATTR_value_highlight_obj_cb
 * ------------------------------------------------------------------------*/
int ATTR_value_highlight_obj_cb ( int dialog_id,
```

```
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     /* Make sure User Function is available. */
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

     /* ---- Enter your callback code here ----- */
      vector <int> which = UIS_getListInt(dialog_id, "LIST_VALUE");
      cout<<"which highlight: "<<which[0]<<endl;
      vector<string> objnames = gatt.GetObjNames();
      cout<<"hightlight objname: "<<objnames[which[0]]<<endl;

      vector<string> sobjnames = singlize (objnames);
      for (int i=0; i<sobjnames.size(); i++)
      {
           if (sobjnames[i]==objnames[which[0]])
           {
                //highlight that object
                tag_t object_id = NULL_TAG;
                UF_OBJ_cycle_by_name ((char*)objnames[which[0]].c_str(), &object_id );
                UF_DISP_set_highlight (object_id, 1 ); //0=off, 1=on
           }
           else
           {
                //turn off highlight
                tag_t objectoff_id = NULL_TAG;
                UF_OBJ_cycle_by_name ((char*)sobjnames[i].c_str(), &objectoff_id );
                UF_DISP_set_highlight (objectoff_id, 0 );
           }
      }


     UF_terminate ();

    /* Callback acknowledged, do not terminate dialog */
    return (UF_UI_CB_CONTINUE_DIALOG);

    /* or Callback acknowledged, terminate dialog.    */
    /* return ( UF_UI_CB_EXIT_DIALOG );               */

}



/* ------------------------------------------------------------------------
 * Callback Name: ATTR_delete_cb
 * ------------------------------------------------------------------------*/
int ATTR_delete_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     /* Make sure User Function is available. */
     if ( UF_initialize() != 0)
          return ( UF_UI_CB_CONTINUE_DIALOG );

     /* ---- Enter your callback code here ----- */

      //int which = UIS_getIntValue(dialog_id, "LIST_VALUE");
      vector<int> which = UIS_getListInt(dialog_id, "LIST_VALUE");
      cout<<"which: "<<which[0]<<endl;

      vector<string> ydomains = gatt.GetDomains();
      vector<string> ysubs = gatt.GetSubdomains();
      vector<string> ytitles = gatt.GetTitles();
      vector<string> yvalues = gatt.GetValues();
      vector<string> yobjnames = gatt.GetObjNames();


      //give a warning before you delete it
      /*char* title_string = "WARNING";
      UF_UI_MESSAGE_DIALOG_TYPE dialog_type = UF_UI_MESSAGE_WARNING;
      char* messages = "Delete the attribute?";
      UF_UI_message_buttons_t  buttons = { TRUE, TRUE, FALSE, "YES", "NO", NULL, 1, 2, 0
};
      int response;
      UF_CALL(UF_UI_message_dialog (title_string, dialog_type, &messages, 1, FALSE,
&buttons, &response ));*/

      //if(response==1)
      //{
```

```
      if(yvalues.size()>0)
      {
           //delete it from the sql database
           char what[222];
           sprintf(what, "select library.id from library where domain='%s' and
subdomain='%s' and title='%s'",
                (char*)ydomains[which[0]].c_str(), (char*)ysubs[which[0]].c_str(),
                (char*)ytitles[which[0]].c_str());

           cout<<"what1:   "<<what<<endl;

           vector<int> libids = select_int_query(what);

           sprintf(what, "delete from part_attributes where object_name = '%s' and value =
'%s' and lib_id = %d and part_id = %d",
                (char*)yobjnames[which[0]].c_str(), (char*)yvalues[which[0]].c_str(),
libids[0], gatt.GetPartID());

           cout<<"what2:  "<<what<<endl;

           mysql_query(conn, what);


           //delete the 'which' from the vectors
           ydomains.erase(ydomains.begin() + which[0]);
           ysubs.erase(ysubs.begin() + which[0]);
           ytitles.erase(ytitles.begin() + which[0]);
           yvalues.erase(yvalues.begin() + which[0]);
           yobjnames.erase(yobjnames.begin() + which[0]);

           //set the values in the data structure
           gatt.SetDomains(ydomains);
           gatt.SetSubdomains(ysubs);
           gatt.SetTitles(ytitles);
           gatt.SetValues(yvalues);
           gatt.SetObjNames(yobjnames);

           //fill in the selection boxes
           if(yvalues.size()>1)
           {
                cout<<"Size>1"<<endl;
                UIS_setListString(dialog_id, "LIST_DOMAIN", ydomains, 0, TRUE);
                UIS_setListString(dialog_id, "LIST_SUB_DOMAIN", ysubs, 0, TRUE);
                UIS_setListString(dialog_id, "LIST_TITLE", ytitles, 0, TRUE);
                UIS_setListString(dialog_id, "LIST_VALUE", yvalues, 0, TRUE);
                UIS_setListString(dialog_id, "LIST_OBJ_NAME", yobjnames, 0, TRUE);
           }
           else
           {
                cout<<"Size=1"<<endl;
                UIS_removeListItem( dialog_id, "LIST_DOMAIN", 0);
                UIS_removeListItem( dialog_id, "LIST_SUB_DOMAIN", 0);
                UIS_removeListItem( dialog_id, "LIST_TITLE", 0);
                UIS_removeListItem( dialog_id, "LIST_VALUE", 0);
                UIS_removeListItem( dialog_id, "LIST_OBJ_NAME", 0);
           }
      }

      //}

     UF_terminate ();

    /* Callback acknowledged, do not terminate dialog */
    return (UF_UI_CB_CONTINUE_DIALOG);

    /* or Callback acknowledged, terminate dialog.    */
    /* return ( UF_UI_CB_EXIT_DIALOG );               */

}


/* ------------------------------------------------------------------------
 * Callback Name: ATTR_action_multiobjs

 * ------------------------------------------------------------------------*/
int ATTR_action_multiobjs ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     /* Make sure User Function is available. */
     if ( UF_initialize() != 0)
```

```
          return ( UF_UI_CB_CONTINUE_DIALOG );

     /* ---- Enter your callback code here ----- */
      tag_t *objects;
      int cnt =  select_class_dialog("Select Objects", &objects);
      cout<<"cnt: "<<cnt<<endl;

      char name[133];
      int check = 7;
      vector<string> objnames;
      for (int i=0; i<cnt; i++)
      {
          check = UF_OBJ_ask_name (objects[i], name);
          if(check == 0)
          {
               cout<<"name: "<<name<<endl;
               objnames.push_back(name);
          }
          else
          {
               cout<<"obj "<<i<<" has no name"<<endl;
               int ir3=0;
               char ca1[144];
               sprintf(ca1, "Enter OBJECT %d name", i);
               char ca2[133];
               sprintf(ca2,"Enter_obj%d_name_no_spaces", i);
               int resp = UF_CALL(uc1600 (ca1,  ca2, &ir3 ));
               tag_t obj_tag = NULL_TAG;
               UF_CALL(UF_OBJ_cycle_by_name (ca2, &obj_tag ));

               if(obj_tag==0)
               {
                    UF_CALL(UF_OBJ_set_name (objects[i], ca2 ));
                    char capsname[33];
                    UF_OBJ_ask_name (objects[i], capsname);
                    //objnames.push_back(ca2);
                    objnames.push_back(capsname);
                    //gatt.SetObjectName(ca2);
               }
               else
               {

                    char* title_string = "WARNING";
                    UF_UI_MESSAGE_DIALOG_TYPE dialog_type = UF_UI_MESSAGE_WARNING;
                    char* messages = "Non-unique object name!  Object name will not be
set.\nTry again with a different name.";
                    UF_UI_message_buttons_t  buttons = { TRUE, FALSE, FALSE, "OK", NULL,
NULL, 1, 0, 0 };
                    int response;
                    UF_CALL(UF_UI_message_dialog (title_string, dialog_type, &messages,
                         1, FALSE,  &buttons, &response ));

               }
          }
      }


      UIS_setListString(dialog_id, "LIST_MULTIOBJS", objnames, 0, TRUE);
      gatt.SetMultiObjNames(objnames);


     UF_terminate ();

    /* Callback acknowledged, do not terminate dialog */
    return (UF_UI_CB_CONTINUE_DIALOG);

    /* or Callback acknowledged, terminate dialog.    */
    /* return ( UF_UI_CB_EXIT_DIALOG );               */

}


/* ------------------------------------------------------------------------
 * Callback Name: ATTR_help_cb
 * ------------------------------------------------------------------------*/
int ATTR_help_cb ( int dialog_id,
             void * client_data,
             UF_STYLER_item_value_type_p_t callback_data)
{
     /* Make sure User Function is available. */
     if ( UF_initialize() != 0)
```

```
        return ( UF_UI_CB_CONTINUE_DIALOG );

     /* ---- Enter your callback code here ----- */

     UF_terminate ();

    /* Callback acknowledged, do not terminate dialog */
    return (UF_UI_CB_CONTINUE_DIALOG);

    /* or Callback acknowledged, terminate dialog.    */
    /* return ( UF_UI_CB_EXIT_DIALOG );               */

}
```

Appendix B: Attribute and Library Class Headers and Source

```
//File:        Attr.hpp
//Author:      Tyson Baker
//Date:        April 2004

#include "UGOpenMaster.hpp"

class Attribute
{
     public:

          Attribute();
          ~Attribute(void);

          void SetObject(tag_t object);
          void SetObjectName(string objectname);
          void SetPartName(string name);
          void SetLibID(int libid);
          void SetPartID(int partid);
          void SetValues(vector<string> values);
          void SetLibids(vector<string> libids);
          void SetObjNames(vector<string> objnames);
          void SetDomains(vector<string> domains);
          void SetSubdomains(vector<string> subdomains);
          void SetTitles(vector<string> titles);
          void SetValue(string value);
          void SetMultiObjNames(vector<string> monames);

          tag_t     GetObject(){return mObject;}
          string    GetObjectName(){return mObjectName;}
          string    GetPartName(){return mPartName;}
          int       GetLibID(){return mLibID;}
          int       GetPartID(){return mPartID;}
          vector<string> GetValues(){return mValues;}
          vector<string> GetLibids(){return mLibids;}
          vector<string> GetObjNames(){return mObjNames;}
          vector<string> GetDomains(){return mDomains;}
          vector<string> GetSubdomains(){return mSubdomains;}
          vector<string> GetTitles(){return mTitles;}
          string    GetValue(){return mValue;}

          vector<string> GetMultiObjNames(){return mMultiObjNames;}


     private:

          string    mPartName;
          tag_t     mObject;
          string    mObjectName;
          string    mValue;
          vector<string> mValues;
          vector<string> mLibids;
          vector<string> mObjNames;
          vector<string> mDomains;
          vector<string> mSubdomains;
          vector<string> mTitles;

          vector<string> mMultiObjNames;

          int       mPartID;
          int       mLibID;


};


//File:        Attr.cpp
```

```
//Author:       Tyson Baker
//Date:         April 2004

#include "Attr.h"

Attribute::Attribute()
{

}


Attribute::~Attribute()
{

}

void Attribute::SetPartName(string name)
{
     mPartName = name;
}

void Attribute::SetObject(tag_t object)
{
     mObject = object;
}

void Attribute::SetObjectName(string objectname)
{
     mObjectName = objectname;
}

void Attribute::SetLibID(int libid)
{
     mLibID = libid;
}

void Attribute::SetPartID(int partid)
{
     mPartID = partid;
}

void Attribute::SetValues(vector<string> values)
{
     mValues.clear();
     mValues = values;
}

void Attribute::SetLibids(vector<string> libids)
{
     mLibids.clear();
     mLibids = libids;
}

void Attribute::SetObjNames(vector<string> objnames)
{
     mObjNames.clear();
     mObjNames = objnames;
}

void Attribute::SetDomains(vector<string> domains)
{
     mDomains.clear();
     mDomains = domains;
}

void Attribute::SetSubdomains(vector<string> subdomains)
{
     mSubdomains.clear();
     mSubdomains = subdomains;
}


void Attribute::SetTitles(vector<string> titles)
{
     mTitles.clear();
     mTitles = titles;
}

void Attribute::SetValue(string value)
{
     mValue = value;
```

```
}

void Attribute::SetMultiObjNames(vector<string> monames)
{
     mMultiObjNames = monames;
}



//File:        Library.hpp
//Author:      Tyson Baker
//Date:        April 2004

#include "UGOpenMaster.hpp"

class Library
{
public:
     Library();
     ~Library(void);

     vector <string> getDomainsList(){return mDomainsList;}
     vector <string> getTitleList(){return mTitleList;}
     vector <string> getSubDomainList(){return mSubDomainList;}
     //vector <string> getTokens(){return mTokens;}
     string              getTitle(){return mTitle;}
     string              getDomain(){return mDomain;}
     string              getSubdomain(){return mSubdomain;}
     string              getType(){return mType;}


     //void setTokens(vector<string> tokens);
     void setDomainsList(vector<string> domains);
     void setSubDomainList(vector<string> subdomains);
     void setTitlesList(vector<string> titles);
     void setTitle(string title);
     void setDomain(string domain);
     void setSubdomain(string subdomain);
     void setType(string type);



private:
     //vector <string> mTokens;
     vector <string> mDomainsList;
     vector <string> mTitleList;
     vector <string> mSubDomainList;

     string              mTitle;
     string              mDomain;
     string              mSubdomain;
     string              mType;

};


//File:        Library.hpp
//Author:      Tyson Baker
//Date:        April 2004

#include "Library.hpp"

Library::Library()
{

}

Library::~Library()
{

}


void Library::setDomainsList(vector <string> domains)
{
     mDomainsList.clear();
     for (int i=0; i<domains.size(); i++)
          mDomainsList.push_back(domains[i]);
}
```

```
void Library::setSubDomainList(vector<string> subdomains)
{
     mSubDomainList.clear();
     for (int i=0; i<subdomains.size(); i++)
          mSubDomainList.push_back(subdomains[i]);
}


void Library::setTitlesList(vector<string> titles)
{
     mTitleList.clear();
     for (int i=0; i<titles.size(); i++)
          mTitleList.push_back(titles[i]);

}

void Library::setDomain(string domain)
{
     mDomain = domain;
}

void Library::setSubdomain(string subdomain)
{
     mSubdomain = subdomain;
}

void Library::setTitle(string title)
{
     mTitle = title;
}

void Library::setType(string type)
{
     mType = type;
}
```

Appendix C: Attribute Translator Code

```
proc loadmysqltcl { dir } {
    set oldcwd [pwd]
    cd $dir
    load mysqltcl[info sharedlibextension]
    cd $oldcwd
}

puts "<<<-------------IT STARTS HERE--------------->>>"

set dir "c:/windows/system32"
loadmysqltcl $dir
package ifneeded mysqltcl 2.50 [list loadmysqltcl $dir]

#-----------------------------------------------
#connect to the sql database, use attibutes
#-----------------------------------------------

set db [mysqlconnect -host localhost -user root -db attributes]


#-----------------------------------------------
#get the jobname, which should be the same as the
#part name, and (for now) can only be 8 characters
#-----------------------------------------------

set jobnamejunk [ans_getvalue active,,jobnam,,start,0]
set ljn [string length $jobnamejunk]
#puts "jobname length = $ljn"
set trimjn [string trimright $jobnamejunk " "]
set ljn2 [string length $trimjn]
#puts "trimmed jobname length = $ljn2"
#puts "$jobnamejunk.***"
#puts "$trimjn.***"
set jobname $trimjn
#puts "$jobname.*** at last!!!"

#if there are already some volumes, clear and start new
if {[catch {
        set Vcnt [ans_getvalue volu,,count]
        if {$Vcnt > 0} {
                puts "volumes already existing"
                ans_sendcommand "fini"
                ans_sendcommand "/cle"
        }
} catchresult]} {
        global errorInfo
        puts $catchresult
        puts "***Tcl TRACE***"
        puts $errorInfo
} else {
        puts "command ok"
}

#-------------------------------------------------------
#query SQL to get part_id and list of object names
#-------------------------------------------------------

set idquery [format "select id from parts where part_name = '%s'" $jobname]
puts $idquery
set id [mysqlsel $db $idquery -list]
puts "id = $id"

set objquery [format "select object_name from part_attributes where part_id =%d" $id]
puts $objquery
set objnames [mysqlsel $db $objquery -list]
puts $objnames
set objnames_size [llength $objnames]

set part_id [mysqlsel $db "select parts.id from parts where part_name = '$jobname'" -list]
```

```
#-------------------------------------------------------
#import the UG part
#-------------------------------------------------------

catch {
ans_sendcommand "~UGIN, $jobname, prt,,all,1-256,0"
}

#-----------------------------------------------------------------------
#parse the *.tbl file to create Ansys objects associated w/ the UG/SQL obj names
#-----------------------------------------------------------------------

set pathname "c:/attribution/ansystuff/$jobname.tbl"
puts "pathname: $pathname"

if [catch {open $pathname r} fileID] {
        puts stderr "Can't open $fileID"
        #puts "CAN'T open $fileID"
}  else {
        #read and process the file
        puts "***FILE OPENED SUCCESSFULLY!***\n"
        set contents [split [read $fileID] \n]
        set contents_size [llength $contents]
        #puts "contents_size = $contents_size"

        for {set i 0} {$i < $contents_size} {incr i} {
                set temp_line [lindex $contents $i]
                set temp_line_size [llength $temp_line]

                for {set j 0} {$j<$temp_line_size} {incr j} {
                        set temp_word [lindex $temp_line $j]

                        for {set k 0} {$k < $objnames_size} {incr k} {
                                set objnametemp [lindex $objnames $k]
                                if {$objnametemp == $temp_word} {
                                        set temp_type [lindex $temp_line 1]
                                        set temp_ansysid [lindex $temp_line 2]
                                        puts "$temp_word : $temp_type : $temp_ansysid"

                                        if {$temp_type == "LINE"} {
                                                ans_sendcommand "lsel,s,,,$temp_ansysid"
                                                ans_sendcommand "cm,$temp_word,line"
                                        } elseif {$temp_type == "AREA"} {
                                                ans_sendcommand "asel,s,,,$temp_ansysid"
                                                ans_sendcommand "cm,$temp_word,area"
                                        } else {
                                                ans_sendcommand "vsel,s,,,$temp_ansysid"
                                                ans_sendcommand "cm,$temp_word,volu"
                                        }
                                }
                        }
                }
        }

close $fileID
}

#-------------------------------------------------------------------------------------------
#perform attribute operations
#-------------------------------------------------------------------------------------------

set single_objnames [mysqlsel $db "select object_name from part_attributes where part_id =
$part_id group by object_name" -list]

set smobjnames_size [llength $single_objnames]
puts "smobjnames: $single_objnames"



set etypenum [expr 2] ;#used to increment the element type number
set esizeflag "luke"
set etypeflag "duke"

#set tet10_92 to be the default element
ans_sendcommand "ET,1,SOLID92"

for {set i 0} {$i < $smobjnames_size} {incr i} {
        set temp_objname [lindex $single_objnames $i]
        set temp_paids [mysqlsel $db "select part_attributes.id from part_attributes where
object_name='$temp_objname'" -list]
        set temp_paids_size [llength $temp_paids]
```

```
        #puts "temp_paids: $temp_paids"

        #loop thru all the attributes.id's of objname [i] for the desired part.id
        for {set j 0} {$j<$temp_paids_size} {incr j} {
                set temp_pa_id [lindex $temp_paids $j]
                set temp_libid [mysqlsel $db "select lib_id from part_attributes where id =
$temp_pa_id" -list]
                set temp_domain [mysqlsel $db "select domain from library where library.id
                = $temp_libid" -list]

                set temp_subdomain [mysqlsel $db "select subdomain from library where
                library.id = $temp_libid" -list]

                set temp_title [mysqlsel $db "select title from library where library.id =
                $temp_libid" -list]

                set temp_value [mysqlsel $db "select value from part_attributes where id =
                $temp_pa_id" -list]

                #puts "temp_objname: $temp_objname"
                #puts "temp_domain= $temp_domain, temp_subdmn: $temp_subdomain, temp_title:
$temp_title"



        #-------------------------------------------------------------------------------
        #perform mesh operations
        #-------------------------------------------------------------------------------

                if {$temp_domain == "mesh" && $temp_subdomain == "element_type"} {
                        #puts "element type found"
                        set etypeflag "true"

                        if {$temp_title == "brick8"} {
                                ans_sendcommand "ET,$etypenum,SOLID45"
                                ans_sendcommand "TYPE, $etypenum"
                                set etypenum [expr $etypenum + 1]
                        }
                        if {$temp_title == "tet10_92"} {
                                ans_sendcommand "TYPE, 1"
                        }
                }
                if {$temp_domain == "mesh" && $temp_subdomain == "none"} {
                        #set global volume element size
                        if {$temp_title == "volume_element_size"} {
                                set esizeflag "true"
                                ans_sendcommand "esize,$temp_value,0,"
                        }
                        if {$temp_title == "vol_smrt_elmnt_sz"} {
                                ans_sendcommand "smrt, $temp_value"
                                set esizeflag "true"
                        }
                }
                puts "etypeflag: $etypeflag, esizeflag: $esizeflag"

        #element type and size must be defined before the mesh command should be called

                catch {
                if {$etypeflag == "true" && $esizeflag == "true"} {
                        ans_sendcommand "cmsel,s,$temp_objname"
                        ans_sendcommand "vclear,$temp_objname"
                        ans_sendcommand "vmesh,$temp_objname"
                        set etypeflag "bo"
                        set esizeflag "duke"
                } }

        #-------------------------------------------------------------------------------
        #end mesh operations
        #-------------------------------------------------------------------------------
        }
}

#check if meshed
ans_sendcommand "allsel,all,volu"
set volcnt [ans_getvalue volu,,count]
puts "volcnt: $volcnt"
for {set k 1} {$k<= $volcnt} {incr k} {
        set ismeshed [ans_getvalue volu,$k,attr,type]
        puts "vol $k ismeshed= $ismeshed"
        catch {
        if {$ismeshed == 0} {
```

```
                puts "vol$k isn't meshed"
                ans_sendcommand "smrt,10"
                ans_sendcommand "type, 1"
                ans_sendcommand "vmesh,$k"
        }
        }
}

#-----------------------------------------------------------------------------------------
#perform material operations
#-----------------------------------------------------------------------------------------

set exflag "false"
set prxyflag "false"
set matnum [expr 1] ;#used to control which material number to work with

for {set i 0} {$i < $smobjnames_size} {incr i} {
        set temp_objname [lindex $single_objnames $i]
        set temp_paids [mysqlsel $db "select part_attributes.id from part_attributes where
object_name='$temp_objname'" -list]
        set temp_paids_size [llength $temp_paids]
        puts "objname: $temp_objname"
        #puts "temp_paids: $temp_paids"

        #loop thru all the attributes.id's of objname [i] for the desired part.id
        for {set j 0} {$j<$temp_paids_size} {incr j} {
                set temp_pa_id [lindex $temp_paids $j]
                set temp_libid [mysqlsel $db "select lib_id from part_attributes where id =
$temp_pa_id" -list]

                set temp_domain [mysqlsel $db "select domain from library where library.id
= $temp_libid" -list]

                set temp_subdomain [mysqlsel $db "select subdomain from library where
library.id = $temp_libid" -list]

                set temp_title [mysqlsel $db "select title from library where library.id =
$temp_libid" -list]

                set temp_value [mysqlsel $db "select value from part_attributes where id =
$temp_pa_id" -list]

                #puts "temp_objname: $temp_objname"
                #puts "temp_domain= $temp_domain, temp_subdmn: $temp_subdomain, temp_title:
$temp_title"

                #-------------------------------------------------------------------------
                #perform material operations
                #-------------------------------------------------------------------------

                puts "------------------ Start of Material Operations ------------------"

                if {$temp_domain == "material" && $temp_subdomain ==
"str_lnr_elstc_istrpc"} {

                        catch {
                        puts "matnum = $matnum"
                        if {$temp_title == "EX"} {
                                ans_sendcommand "mp,ex,$matnum,$temp_value"
                                set exflag "true"
                        }
                        if {$temp_title == "PRXY"} {
                                ans_sendcommand "mp,prxy,$matnum,$temp_value"
                                set prxyflag "true"
                        }

                        #selects the working volume
                        if {$exflag == "true" && $prxyflag == "true"} {
                                ans_sendcommand "vsel,s,volu,,$temp_objname"
                                puts "vsel,s,volu,,$temp_objname"
                                puts "matnum = $matnum"
                                ans_sendcommand "allsel,belo,volu"
                                #modify the material number of the elements in the working
volume

                                ans_sendcommand "emodif,all,mat, $matnum"
                                set matnum [expr $matnum + 1]
                                set exflag "false"
                                set prxyflag "false"
                                ans_sendcommand "allsel,all"
                        }
```

```
                        } ;#end catch

                }
                puts "------------------ End of Material Operations ------------------"

                #-----------------------------------------------------------------------------
                #perform load operations
                #-----------------------------------------------------------------------------

                puts "------------------ Start of LOAD Operations ------------------"
                catch {
                if {$temp_domain == "structural" && $temp_subdomain == "none"} {
                        if {$temp_title == "displacement"} {
                                #set the displacement value for the temp_objname
                                ans_sendcommand "DA,$temp_objname,ALL,$temp_value"
                                puts "DA,$temp_objname,ALL,$temp_value ---> ;)"
                        }

                        if {$temp_title == "pressure_load"} {
                                #set the pressure load on the temp_objname
                                ans_sendcommand "SFA,$temp_objname,1,PRES,$temp_value"
                                puts "SFA,$temp_objname,1,PRES,$temp_value ---> ;)"
                        }

                } }

                puts "---------------END OF LOAD OPERATIONS--------------------"

        }
}

#-------------------------------------------------------
#perform solution operations
#-------------------------------------------------------


puts ">>>----------------------THE END----------------------<<<"
```