Automatic Geometric Data Migration Throughout Views of a Model Fidelity Family

Hans L. Soderquist
Brigham Young University - Provo

AUTOMATIC GEOMETRIC DATA MIGRATION THROUGHOUT VIEWS OF A MODEL FIDELITY FAMLY

by

Hans Lars Soderquist

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

April 2006

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Hans Lars Soderquist

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

Date	C. Greg Jensen, Chair
Date	Spencer P. Magleby
Date	Jordan J. Cox

BRIGHAM YOUNG UNIVERSITY

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

Date	C. Greg Jensen Chair, Graduate Committee

Accepted for the Department

Matthew R. Jones
Graduate Coordinator

Accepted for the College

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

ABSTRACT

AUTOMATIC DATA MIGRATION THROUGHOUT
VIEWS OF A MODEL FIDELITY FAMILY

Hans Lars Soderquist

Department of Mechanical Engineering

Master of Science

Changes in geometric model parameters are constant throughout the design process. Each group in an organization needs different model information at varying times during the design process. As a result many different models of the same part or assembly arise for the needs and use of each group in each design phase (from conceptual design to full product definition). When one group makes changes to a model, those changes need to be reflected in all of the models which describe the product in all groups and design phases in order for those changes to be verified against all design criteria that were set, and for those changes to be seen by downstream users in the design process. This thesis describes a method for linking these models together which will provide revision control, assuming all models can be updated from any of the other models which define the product and allow for these models to be parameterized using different schemes.

ACKNOWLEDGEMENTS

My sincere thanks go to my wife Natalie and my daughter Adalyn for their patience. My appreciation also goes to Paul Nelson, Dr. C. Greg Jensen, Dr. Spencer Magleby, Dr. Jordan Cox, Diane Emmerson and Robert Kline for their technical help and support.

TABLE OF CONTENTS

List of Figures ix

CHAPTER 1: Introduction 1

1.1 Problem Statement 2

1.2 Thesis Objective 3

1.3 Delimitation of the Problem 4

CHAPTER 2: Background 5

2.1 Knowledge Based Engineering 5

2.1.1 Parametric Models 6

2.1.2 Inter-model Relations 7

2.1.2.1 Inter-model Geometry 7

2.1.2.2 Inter-model Parameters 8

2.2 Design Processes 9

2.2.1 Linear Design Process 10

2.2.2 Non-linear Design Process 11

2.3 Constraint Solvers 12

2.3.1 Numeric & Geometric Constraint Solvers 12

2.3.2 Two Dimensional and Three Dimensional Constraint Solvers 13

CHAPTER 3: Literature Review 15

3.1 Data Reuse 15

3.2 Model Views 16

3.2.1 Cellular Models 17

3.2.2 Master Models 18

3.3 Concurrent Engineering 20

3.3.1 Process Flow Software 20

3.3.2 PLM Solutions 20

3.3.3 Integrated PLM Solutions 21

CHAPTER 4: Method 23

4.1 Assumptions 24

4.2 General Approach 25

4.3 Data Storage 25

4.3.1 Model Geometry 26

4.3.2 Model Fidelity Family Links 27

4.3.3 Unconventional Constraints for Model Fidelity Family Members 28

4.3.3.1 Fixed Value Constraints 29
4.3.3.2 Proportional Constraints 31
4.3.3.3 Relative Point Constraints 31
4.4 Link Resolution 32
4.5 CAx Integration 36
4.5.1 Read/Write Capabilities 36
4.5.2 Update 36
4.6 Sub-objectives 38
4.6.1 Revision Control 38
4.6.2 Different Parameterization Schemes 38
4.6.3 Multi-Directional Dependancies 38
CHAPTER 5: Results and Discussion of Results 41
5.1 Specific Objectives 41
5.2 Approach 42
5.3 The Geometric Fidelity Linker 42
5.3.1 Creating a Model Fidelity Family Using the Geometric Fidelity Linker 44
5.3.2 Updating a Model Fidelity Family Using the Geometric Fidelity Linker 45
5.3.3 Test Cases 46
5.3.3.1 Corners Models 46
5.3.3.2 Turbine Cases of a Gas-turbine Engine 48
5.3.4 Updating According to Design Intent 50
5.3.5 API Problems 51
5.4 Analysis of Revision Control 51
5.5 Analysis of Different Parameterization Schemes 52
5.6 Analysis of Multi-directional Dependencies 53
CHAPTER 6: Conclusion 55
6.1 Automatic Updating of a Model Fidelity Family 55
6.1.1 Revision Control 56
6.1.2 Different Parameterization Schemes 56
6.1.3 Multi-directional Dependencies 57
6.2 Projected Contributions 57
6.3 Future Work 58
6.3.1 Directional constraints to absolve the problems with multiple solutions to constraints 59
6.3.2 Indication that geometry models are fully constrained 60
6.3.3 Expressions resolution 60
6.3.4 3D implementation 61
6.3.5 Management scheme for when updates are accepted into the database 62
6.3.6 Homogenous transformations for differently oriented models. 62
6.3.7 Assemblies 63
6.3.8 Topology changes 63
6.3.9 Model Attributes 63
6.3.10 Other Engineering Applications 63
References 65
Appendix A: Graphical User Interface 71
Appendix B: Geometric Fidelity Linker Code 75

LIST OF FIGURES

Figure 1: Parametric block showing the driving parameters 6
Figure 2: Panel assembly created using inter-model geometry 8
Figure 3: Inter-part expressions example -- the blades of a gas-turbine engine 9
Figure 4: An idealized, linear design process (arrows show dependency) 10
Figure 5: A more realistic design process view (arrows show dependancy).................. 11
Figure 6: An example of model views.. 17
Figure 7: Master model architecture with client views (Hoffman [18]).......................... 19
Figure 8: Representation of a bezier curve as its control points as opposed to points along the curve 26
Figure 9: The linking structure of a model fidelity family 27
Figure 10: Models of varying specificity with varying topology 29
Figure 11: Fixed value constraint and their inherent problem... 30
Figure 12: Proportional constraint example... 31
Figure 13: Relative point constraint example 32
Figure 14: Two models from a model fidelity family to illustrate the transfer functions.. 33
Figure 15: The data structure used in the Geometric Fidelity Linker.............................. 43
Figure 16: Polymorphing of classes from the data domain to the NX2 domain.............. 44
Figure 17: The three corners models 47
Figure 18: The results of a spatial change in position for the turbine cases case study. 49
Figure 19: The problem of multiple solutions of a given constraint................................ 59
Figure 20: An example of a directional constraint... 59
Figure 21: Example of how expressions work... 61
Figure 22: Initial Option GUI 71
Figure 24: Model Selection GUI.. 72
Figure 25: Link Constraints GUI 73

CHAPTER 1: INTRODUCTION

The challenge to create new and innovative products in a shorter amount of time is constantly increasing. Companies continually search for methods to increase productivity and decrease design cycle times. Modern computer technology has allowed for rapid creation of geometric models and their analyses. Parametric technology has allowed quick iterations of those models and analyses for optimization. However, a product needs to be optimized and checked for conformance to company design rules at all phases of the design cycle. This means that each group in an organization has to share the changes that occur in that group with all other groups responsible for the definition of the product. Also, each group in an organization needs different model information at varying times. As a result many different models of the same part or assembly arise for the needs and use of each group. These models may be specific in one area of the model and less specific in others. For preliminary design the model may be defined by primitive shapes, whereas for final design a detailed model over the entire product is required. These varying levels of specificity may be considered varying levels of model fidelity, and all of these models together may be considered a model fidelity family.

1.1 Problem Statement

Each group in an organization may play a specific role in the creation and form of the product. As one group needs to make changes to a model (i.e. move a wall out or make a boss thinner), the other groups need to see that change, in order for them to provide feedback. Propagation of these changes throughout a model fidelity family needs to occur in order to maintain model design consistency in the various groups, even when communications between the groups is infrequent. All models in the model fidelity family must be constantly validated against the design criteria which have been set for that product at each step of the design process.

Current parametric technology allows for the rapid updating and changing of model parameters. However, many models of a fidelity family may not be parameterized similarly. This is because each group designs their models with a different intent. Models which represent the same product for one group may have a completely different use in another group. This inconsistency in parameterization schemes creates problems when one group "simply" changes a parameter in their model, where other groups may take hours to make the same change in their models due to the parameterization scheme.

The production of multiple models in a fidelity family also leads to a problem with revision control. Each member of the family is owned by the group which uses that model. However, multiple users of the model in a group will tend to have multiple copies of the same family member model. As a result, confusion can arise as to which model is the actual master, and what its final parameter set is as the model's data is passed to other groups for further product definition. If the wrong data leaves the group, this can be very costly to an organization.

Another problem that arises with current parametric technology is the dependence of one model on another. There are many methods in commercially available CAD packages which allow the user to link one model to another (i.e. inter-part expressions, linked geometric entities, etc.). However, these methods require that one model be dependant on another. This unidirectional dependence is undesirable. Often changes made at a detailed design stage need to be seen at the preliminary design stage in order to verify that those changes do not affect the system-wide performance of the product. This unidirectional dependence can also create problems when two groups need to work on their individual models at the same time, independent of one another, yet one model is dependant on the other. Thus, a means must be provided for constraining the various models in a model fidelity family to one another while maintaining model independence.

1.2 Thesis Objective

This research will develop a means to address the various geometry models used to describe the same product in the design process. This will lead to the creation of a model management system for concurrent engineering which will utilize a method to allow these changes to occur automatically. The method will allow the models to be parameterized differently, eliminate unidirectional model dependency (allowing each model to drive all of the other models) and provide revision control measures to eliminate passing undesirable data to other groups.

1.3 Delimitation of the Problem

This research will focus on the process of automatically updating the members of a model fidelity family. It will consider the case of 2D parametric CAD model geometry. It will only consider 2D sketch geometry as there are no suitable 3D constraint solvers available. Assemblies and model attributes will not be considered in this work as they are a natural extension of the method described. Also, other CAD/CAM/CAE (CAx) applications will not be considered. Geometry changes will be executed on the case study, and all models of the fidelity family inspected for consistency to prove the method.

CHAPTER 2: BACKGROUND

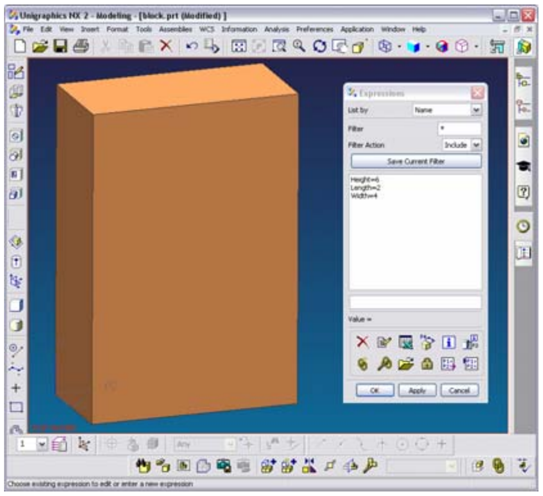
An understanding of current technology and nomenclature is required in order to link the models of a model fidelity family. This chapter is a discussion of these technologies and will provide necessary background for the method which will be describe in this thesis.

2.1 Knowledge Based Engineering

Most companies have certain sets of rules and methods for creating their products. These are necessary to ensure that a product meets a company's and customers' expectations. With the advent of computer technology, the need to infuse those rules and methods into the computer tools used to design a product has developed. Thus the computer tools help engineers to design according to the "in-house" rules of the company, maintaining product integrity. This is known as Knowledge Based Engineering (KBE). CAD/CAM/CAE (CAx) tools have been an important set of tools for KBE integration. CAx applications are the main means for product design, development, manufacturing and support in industry today. In particular CAD applications use parametric models and inter-model relations to rapidly modify and update models according to necessary design changes [1].

2.1.1 Parametric Models

Parametric CAD models have greatly decreased the design time necessary for product definition. A designer defines certain parameters for creation of a model. These parameters are then available for change, much like variables in algebraic equations. The designer may change these variables at will to modify a model as necessary. Figure 1 shows a parametric block and its driving parameters: Height, Length and Width. By changing the values of these parameters, the size and shape of the block can be changed to any desired configuration.


Unigraphics NX 2 - Modeling - [block.prt (Modified)]
File Edit View Insert Format Tools Assemblies WCS Information Analysis Preferences Application Window Help
Expressions
Save Current Filter
Value =
OK
Apply
Cancel

List by: Name
Filter: *
Filter Action: Include

Height=6
Length=2
Width=4

Choose existing expression to edit or enter a new expression

Figure 1: Parametric block showing the driving parameters

The parameters of a parametric model may be set equal to other parameters in the model as well as performing computations and conditional operations. Thus "knowledge" can be built into the parameters of a model. External applications may have access to these model parameters and can thus be used for other calculations (to further integrate company knowledge) or for optimization to meet certain product performance standards. Other possibilities exist for integrating company knowledge into the CAD system. These include rule-based design methods (which will not be discussed in this thesis), inter-model relations, and others.

2.1.2 Inter-model Relations

Inter-model relations are means for relating information in one CAD model to another. This facilitates KBE by allowing the company knowledge and methods to trickle down through parts of an assembly or family. For purposes of this discussion, the model where the information is defined is the parent model, and the model to which the information is given is the child model.

2.1.2.1 Inter-model Geometry

Inter-model geometry is geometry elements which are defined in the parent model yet visible and accessible in the child model. The inter-model geometry may be constrained to and used as a feature for other geometries in the child model and may include points, curves, surfaces, solids, and datums. This makes the child model dependant on the parent model. Only changes in the parent model can facilitate a change in the inter-model geometry in the child model. This does, however, have the advantage of being able to drive complex assemblies with a few defining sketches.

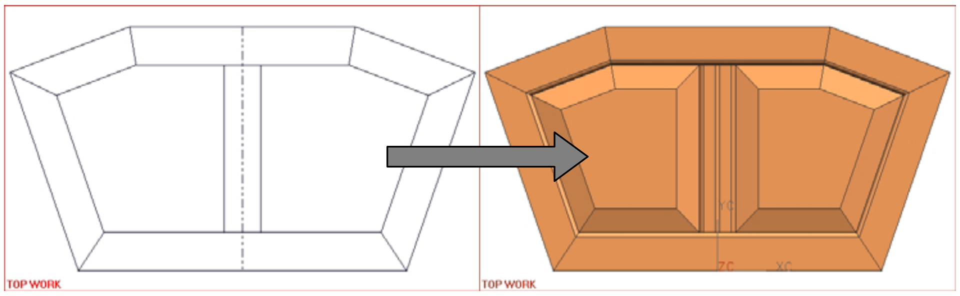


Figure 2: Panel assembly created using inter-model geometry

Figure 2 is an example of a model made with inter-model geometry. The cradle assembly on the right of Figure 2 is a model of the sketch on the left of the same figure. The outline of the cradle panel frame is contained within the parent model which subsequently drives the shape of the assembly parts. This assembly contains 9 parts, all of which are dependant on the parent model.

2.1.2.2 Inter-model Parameters

Inter-model parameters are parameters defined in the child model which are set equal to parameters defined in the parent model. Thus the child model parameter value is set equal to the parent model parameter value and that value is updated whenever the parent model parameter value changes. Therefore, parent model parameters drive child model parameters.

Once again, a dependency is created. However, the child model parameter may be set to some other value than the parent model parameter. This has the effect of breaking the link between the parameters and making the child model independent of the parent model. On the other hand, changing child model parameters would also do away with the KBE interface between the models which was desired in the first place.

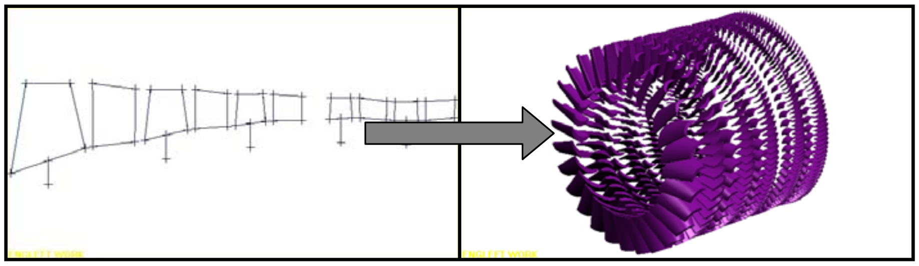


Figure 3: Inter-part expressions example -- the blades of a gas-turbine engine

Figure 3 is an example of a model with inter-model parameters. The sketch on the left represents a 2D projection of airfoils in a gas-turbine engine compressor. The corner points of the airfoils are defined by parameters in that model. Parameters in the 3D blades in the model on the right of Figure 3 are set equal to those parameters in the 2D model. Changing the parameter values in the 2D definition will automatically change the values of the parameters in the 3D definition.

2.2 Design Processes

The creation of a product follows some process of steps, from concept initiation to manufacturing and product support. Each of the steps taken along the path to full product definition has its own set of methods and rules which must be followed.

2.2.1 Linear Design Process

Most often, the design process which is expected is a linear design process such as in Figure 4.

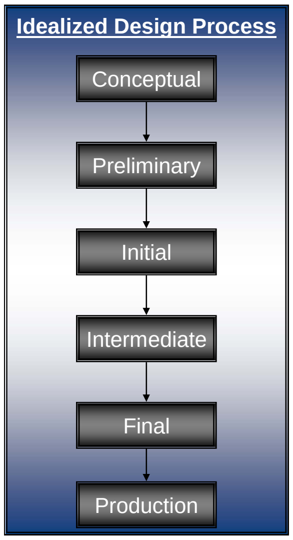


Figure 4: An idealized, linear design process (arrows show dependency)

Each phase of the design process is only dependant on the phase before, and once a phase is completed, it is never revisited. In the end, a product may be developed simply by stepping through each phase of the design process until it is completed. If this design process were followed, the KBE tools described in section 2.1 would be quite useful and adequate. However, this process is unrealistic. Iteration and reevaluation is always necessary in the design of a product. Models at any given phase of a design process cannot be solely dependant on the phases before it, but must be dependant on phases which occur after that stage of design definition.

2.2.2 Non-linear Design Process

A more realistic approach to understanding a design process is to understand that each phase in the design process may in fact be dependant on every other phase (see Figure 5).

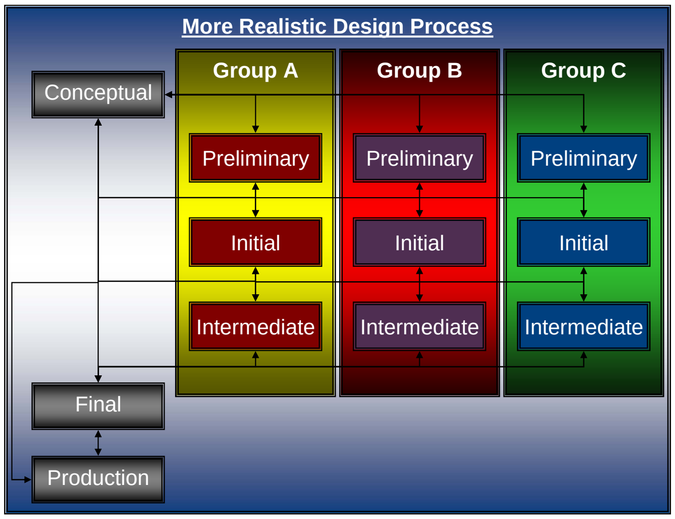


Figure 5: A more realistic design process view (arrows show dependancy)

A conceptual design may be created under certain product performance requirements. This conceptual design is then sent to various groups in the company for further product definition. As the product becomes more defined, each model will be checked against the performance requirements set by the conceptual design. Any necessary changes to meet those conceptual criteria must be handled by the individual groups again at all levels of the design definition. This process repeats itself as final and production designs are created as well. Thus all phases of the design process are dependant on all other phases of the design process. The reality of the necessity of these interdependencies is the foundation for this thesis.

2.3 Constraint Solvers

Parametric CAD models have created the need for the integration of constraint solvers for solving the set of parameters on the geometry as those parameters are modified. It is the ability to solve these constraints that sets parametric CAD packages apart from their predecessors. In order to link the members of a model fidelity family together independently of one another, constraints between the models must be created, which will then propagate change to the parametric model parameters and finally to the model as a whole. Thus, a constraint solver becomes necessary.

Constraint solvers have evolved along two lines, numeric and geometric. This section describes the constraint solvers available and being developed today.

2.3.1 Numeric & Geometric Constraint Solvers

Originally constraint solvers evolved through numeric means. The equations defining the various geometry elements were organized in such a manner as to be solved by direct computation. This provides solutions quickly and accurately. However, as geometry became more complex, the numeric methods employed became either too cumbersome or impossible to solve. This led to geometric constraint solvers.

Geometric constraint solvers employ iterative techniques to solve the geometry of a model. This set of tools allows for solutions which are not easily computed or impossible to compute by the original numeric means, and provide for a wider range of solutions and geometry configurations.

Current constraint solvers employ both numeric and geometric techniques for solving constraints. They combine these two fields and find the fastest solution using whichever means necessary.

2.3.2 Two Dimensional and Three Dimensional Constraint Solvers

Two dimensional (2D) constraint solvers today are much more evolved than three dimensional (3D) constraint solvers. 2D constraint solvers allow for updating of the parameters of almost all 2D elements available in commercial CAD packages. The exception is non-uniform rational b-splines (NURBS). Most commercial packages do not have great functionality in constraining NURBS. Many researchers are working on improving constraint solver technology [2][3][4][5][6][7].

3D constraint solvers are still lacking in their capability to solve systems of constraints. Many solvers have the capability of constraining the same 2D elements as the 2D solvers, except they can solve these in 3D space. However, 3D geometric elements such as surfaces and solids have not been well integrated into 3D constraint solvers. This is not surprising since most surfaces and solids are made of NURBS surfaces. There are however, many 3D constraint solvers which are commercially available that are useful in constraining assemblies. Most of these use an open loop kinematics regression technique for solving the positions of parts, but this assumes that all of the parts are rigid [8].

Because of the lack of a reasonable 3D constraint solver for surfaces and solids, the method in this thesis will be limited to 2D geometry. However, as appropriate 3D constraint solvers become available, the method should be extended to the 3D realm.

CHAPTER 3: LITERATURE REVIEW

The following is a review of existing research pertaining to the proposed research. There is little research related directly to this topic; however there are related ideas which contribute to this work. This reviewed research provides a framework from which the proposed research can be accomplished. Many of these ideas will be used together to help implement the method described in this thesis. The research was concentrated into the following categories:

- Data Reuse
- Commercial CAD Model Linking
- Model Views
- Concurrent Engineering

3.1 Data Reuse

Reuse of model data is a concept on which there has been much research. Altmeyer [9] states that ". . . a chance to reduce the design time is reusing existing results." They implemented a search algorithm that searches a database and finds models of similar design and function as the model needed. This sets the model file as a database member, and models are changed and reused. Such an algorithm would be useful for new product definition by extracting data from the various models.

Koegst [10] propose a method which builds off Altmeyer's method by using not only models, but procedures as well. The models and procedures are instantiated into new configurations and procedures which are in turn stored in a library for future reference and use. The reuse of procedures is a common practice in commercial engineering, and in fact it is the insertion of proprietary procedures or "knowledge" which is the basis for knowledge based engineering (see Section 2.1).

Hayes [11] propose a method for tracking the changes made to a model and the rationale for those changes. They are vague in proposing applications of these changes. Having a knowledge of these changes and their rationale can provide a starting point for future models and products as well as exploration of design options previously not considered.

The reuse of data is the foundation of parametric modeling. Parametric models are reused time and again to produce a new product which is similar to others. The methods described above allow the user to make a decision as to which models or data should be modified in order to create a new product. In the production of that product, a model fidelity family could be used and modified, or perhaps models added to the family to further the design process.

3.2 Model Views

A view is a data or model subset of the total product definition. It is often described as the product model as seen by varying engineering tools (CAD, CAM, FEA, CFD, etc.). Each tool needs a different set of information in order to perform its specific task. One of the tasks for the development of concurrent engineering is to develop a method for updating views automatically.

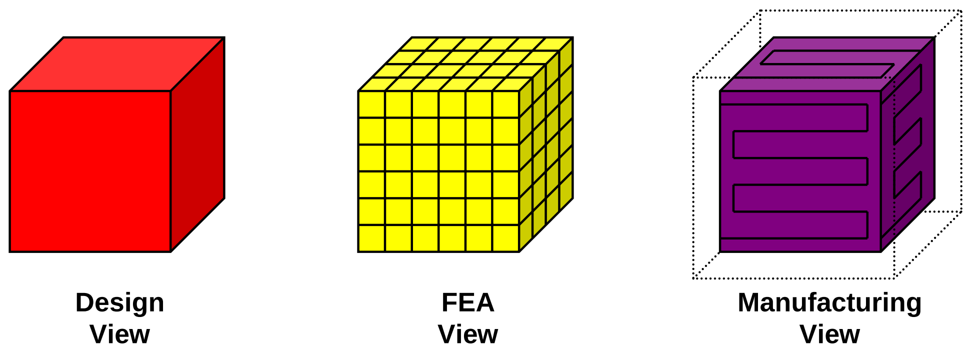


Figure 6: An example of model views

3.2.1 Cellular Models

Bronsvoort [12] introduced a method of understanding models called a cellular model. The models are made up of cells and the cells are described as "... volumetric; they can have overlapping boundaries, but they cannot have overlapping volumes. They reflect all feature intersections, and therefore can have an arbitrary shape." Cells are distinguished from model features in that features make up the boundaries of the volumes which define the cells. They state that the cellular model will be better suited for feature operations than the traditional history based feature model which has been predominant in commercial feature based operations. This allows a feature to retain its intent as opposed to being dumb geometry and has been explored extensively by Bidarra [13][14][15].

de Kraker [16] expanded the usefulness of cellular models to multiple view applications. They show that links between views allow users in each view to modify the geometry in that view and have that geometry changed in all other views. They incorporated means including constraint solvers to propagate this inter-view information.

Bronsvoort [17] continued the expansion of de Kraker's idea by developing a method for updating assembly and part views. Bronsvoort's method includes views for conceptual design, assembly design, detail design, and manufacturing. The proposed methods described the connectivity of the assembly at each successive level.

The cells of the model are used as a check after the views have been updated for consistency between model views. Although the assembly portion of the method is able to consider geometry from earlier design phases than detail design, they admit that they are lacking in a method updating consistency for those earlier phases. The views that have been incorporated using the cellular model could be considered a model fidelity family. The method that will be described in this thesis could be used, along with the methods developed by de Kraker to propagate the data of the models throughout the database. In other words, the cellular models already created could be a foundation upon which the method in this thesis could be used.

3.2.2 Master Models

Hoffman [18] developed the idea of a product master model. He states that a "master model is an object-oriented repository that provides essential mechanisms for maintaining the integrity and consistency of the deposited information structures." The information structures include the net shape of the model, analysis solutions, model attributes, material properties, etc. They propose that the master model be stored within a database structure that is accessible to all views in which the model may be used.

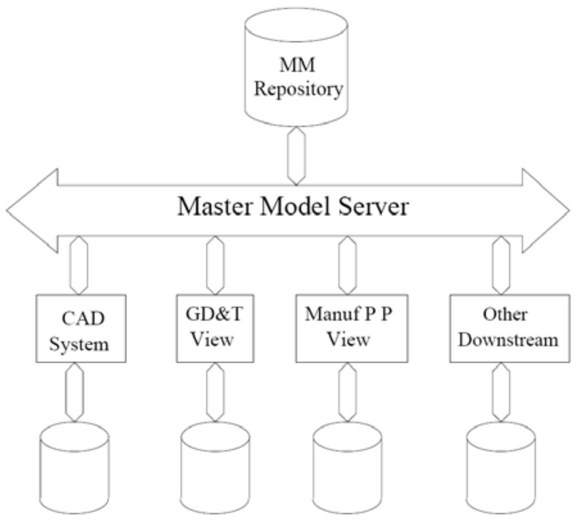


Figure 7: Master model architecture with client views (Hoffman [18])

Views may include design, manufacturing, analysis, etc. The database would extract the model data necessary for the given application and create an instance of that model in the application view. For example, if a user wanted to look at the model in a given CAD package, the model would be extracted specifically for that package, and the user could then view and update the model as necessary. The model information would then be stored again in the master model repository upon saving the model.

Several methods of providing consistency between views are given in [18] and [19]. They include reconciling the differences in shape outside the CAx tool, resolving constraint schemes within the CAD package, and redefining features as necessary. These methods deal solely with models which represent the same net shape, and reconciling the differences between the two.

3.3 Concurrent Engineering

Concurrent engineering focuses on simultaneous use of data by various users for product development.

3.3.1 Process Flow Software

Commercially available packages such as Fiper [20] look at the design process. Process flow diagrams allow the engineer to define how a product should be developed. Once one task is completed, downstream tasks are cued automatically, and the appropriate application or engineer is notified for the next set of tasks to be set in motion. These process flow methods assume a quasi-linear relationship between processes. They allow for iterating sections of the overall process, but do not allow the flexibility of any process lending information to any other. This may be a desirable trait in order to ensure engineering standard practices are met. However, it may be a hindrance to the communication needed between the groups of an organization which are working on a particular product.

3.3.2 PLM Solutions

Product Lifecycle Management (PLM) tools such as Teamcenter Engineering [21] are other commercially available products which try to address the concurrent engineering problem. Teamcenter Engineering's approach is to split a product into different models which are accessible by the various design groups. This is similar to Broonsvort's views in that they are split into manifestation (tool path), specification (drawings and documentation), and alt rep (variants of the base model), as well as the base model itself. Teamcenter Engineering holds the models in a database and requires those who wish to modify them to check them out, thus preventing confusion when models are updated. The main problem with this is that a model may only be checked out by one person at a time. Thus, as the product definition continues, only one group can modify the model at any particular time. This does not allow for ownership of the model by any group or true concurrent engineering since only one group can work on a model at a time. It does, however, keep a sacred model as a master.

3.3.3 Integrated PLM Solutions

Recent research efforts in concurrent engineering have tried to combine these two methods. Fife [22] has proposed a framework for the integration of the process flow methods found in Fiper into the PLM framework. He proposes organizing the work flow with the data and models stored in the PLM system, and using that same system to notify engineers and automatically run processes. This allows an organization wide implementation of a process.

Lund [23] has proposed a less specific method by combining the ideas of PLM and [18] and [19]. He would store CAx and other parametric models as data in the PLM framework and create instantiations of those models in whatever client was pertinent to the group accessing the data (i.e. CAx, spreadsheets, etc.). A base instantiation would be saved in the native model form, and variants stored as metadata in the database. This would allow for quick retrieval, representation and manipulation of the data, without the mass storage of proprietary tools.

CHAPTER 4: METHOD

This thesis proposes a method for linking the various members of a model fidelity family. This section describes the steps and tools necessary to implement this method.

In addition to linking the various members of a model fidelity family, there are several sub-objectives which are necessary for this method to be useful in a company which uses parametric CAD.

The method needs to incorporate some means of revision control so that there is a model which represents what all other groups in the company will see, and that represents the true state of the product as the design process progresses. Without this, there would be no means of controlling which group's model was the true model, and severe confusion could ensue.

The method must be able to handle models that have been parameterized differently. If all of the models were parameterized the same, a simple updating of the expressions would be applicable, and there would be no merit to this thesis. Allowing the models to be parameterized differently will preserve the original design intent of the model. This is important because an engineer may have critical parameters on which a given model is based, and these are the driving parameters for that model. Thus the model has an intended design and the designer has a design intent for that model. Many members of a model fidelity family will have specific design intents for the functionality of that model within the various groups which will contribute to the overall product definition. These intents must be preserved in order for the models to be functional, and thus the models must be allowed to be parameterized differently.

The method also needs to be able to update all of the models in the model fidelity family from any of the other models in the family. This is important because of the non-linear nature of the design process used in companies today.

4.1 Assumptions

This method assumes that the organization which will use it has a well defined design scope. Thus they have set forth specific design criteria which they are trying to meet. It is these design criteria against which all models of a model fidelity family must be verified. The method also assumes that there is a seed file for creating the model fidelity family. Models may be added to or removed from the family as design decisions may require. Ideally, all members of the model fidelity family would be created at the time of product inception, but this may not be a realistic expectation.

The method also assumes an intimate understanding of the design process and scope. Model designers know the design intent of individual models within that family. Certain designers also have to know how two particular models relate to one another, and can use this understanding of the models' design intents to create a link which will be created between them. This link will define how the models relate and change relative to one another. Thus this method makes no attempt to define the intent of the link between the models, but uses the directions implicit in the constraints which will be applied between the two models.

4.2 General Approach

After a user has made design changes to a member of a model fidelity family in the CAx package of his or her choice, propagation of those changes to all members of the family will be accomplished in the following manner:

1. Model geometry is extracted from the CAx application into a data storage scheme which holds model geometry and links between model fidelity family members.
2. The links between model fidelity family members are resolved.
3. Necessary methods are implemented to integrate other information from the CAx package into the data storage scheme (i.e. parameters in parametric CAD models).

4.3 Data Storage

A storage system needs to be created in order to contain all of the information to define a model fidelity family. This system needs to store the members of the family as well as links between the members which will describe how the models relate to each other.

4.3.1 Model Geometry

Model geometry will need to be stored in such a way as to represent the geometry and be able to recreate the geometry in any model view which may be used by a company. Geometry may be stored as representative values for the definition of a full geometric feature. It would be impractical to try to store the thousands of data points which would define a curve to a reasonable definition. Instead, curve control points should be stored, from which any point on a curve could be calculated and constructed. Figure 8 shows a Bezier curve which can be defined at any point by the four points which make up the control structure of the curve. These four points are all that are necessary to store.

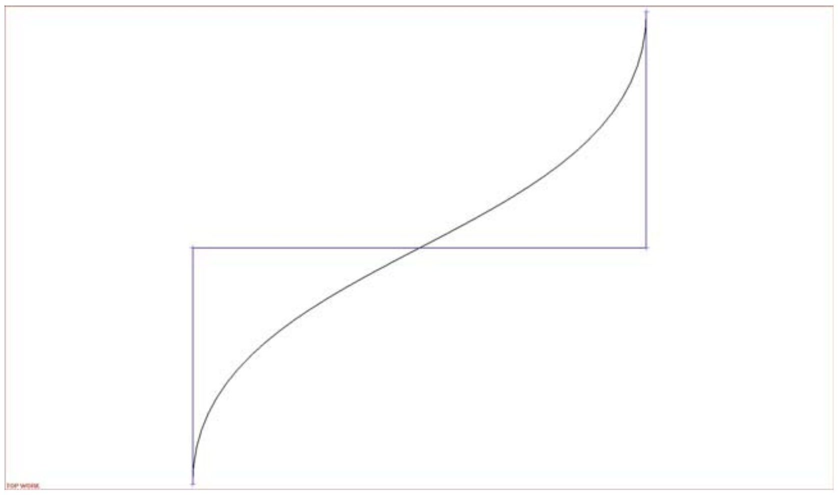


Figure 8: Representation of a bezier curve as its control points as opposed to points along the curve

A data repository as described in [18] and [19] or [23] would serve as an appropriate storage method. This repository will provide the necessary revision control to protect the developing product from being incidentally misrepresented to other groups.

4.3.2 Model Fidelity Family Links

In order to maintain the independence of each member of a model fidelity family (i.e. the model may be completely changed, independent of any other members of the model fidelity family) and to allow all members of the family to be updated by any single member of the family, links between members of the family need to be created. It would be impractical to create a link between one model and every other model in the family, since a model fidelity family could potentially contain hundreds of models. However, it is quite practical that each member of a model fidelity family be constrained to one other member in the family. This will allow all of the members of the model fidelity family to be linked to each other through all of the other members (see Figure 9).

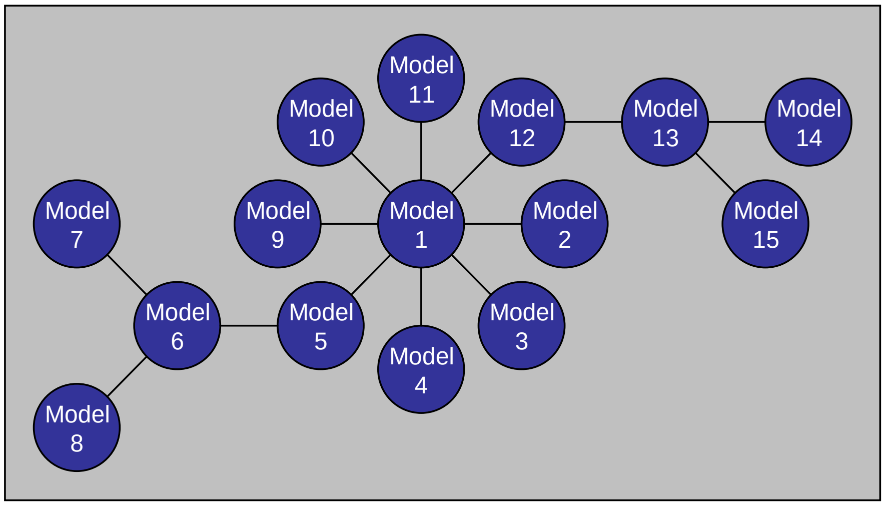


Figure 9: The linking structure of a model fidelity family

This allows the designer to choose the model which is most like the model he or she wishes to add to the model fidelity family. This will reduce the complexity of the link and make the process much quicker. Also, the ability to create unique constraint schemes between models allows a designer to introduce a distinct method for one model to update relative to another. This provides another means by which design intent could be placed into a model. Perhaps driving parameters in one model would be critical to driving parameters in another model. The links created within a model fidelity family could define the relationship between those two sets of driving parameters and provide a structured means for models to interrelate.

A link between two models of a model fidelity family needs to contain the following information:

1. The models which are to be linked together
2. A set of constraints between geometry in the models

These links should be stored in the data repository that holds all of the models in the model fidelity family.

4.3.3 Unconventional Constraints for Model Fidelity Family Members

Because the goal of this method is to link models of varying specificity, not all members of a model's topology in a more complex model will be fully described by the topology of the less specific model (see Figure 10).

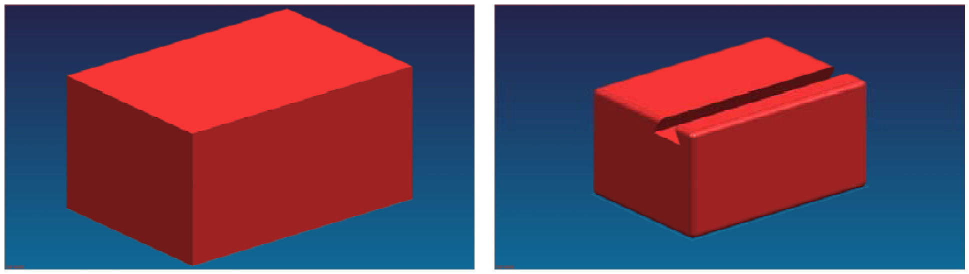
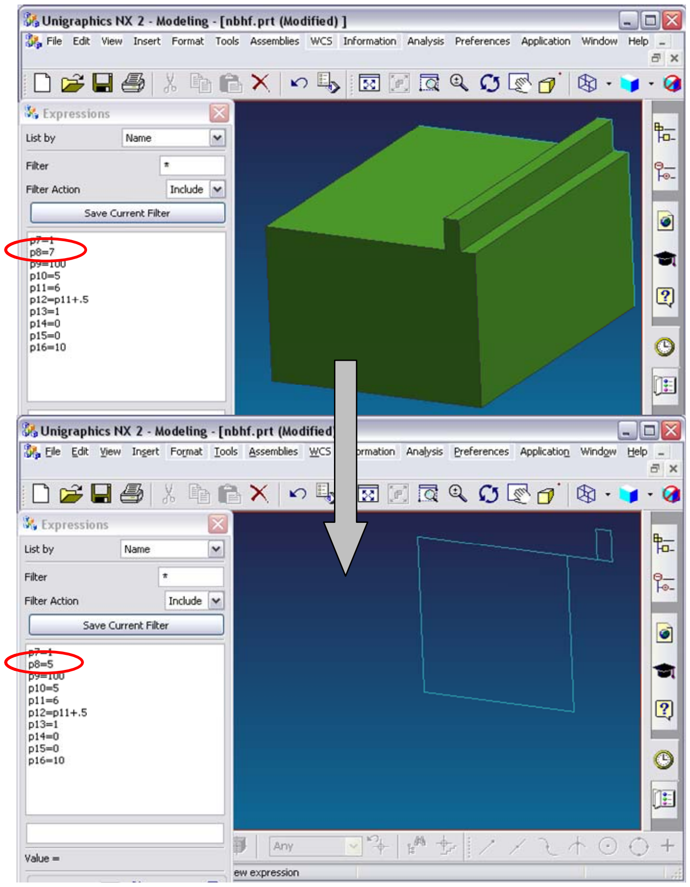


Figure 10: Models of varying specificity with varying topology

In the case that the model which has already changed is the more specific model, and the model which is to be changed is the less specific model, this would most likely not be problematic as long as there are no constraints between the geometry in the less specific model and the extra geometry in the more specific model. However, in the reverse situation all of the topology of the updating model would not be constrained to the topology of the changed model. This situation must be resolved. Following are several possibilities for fixing this problem. This does not represent a complete set of constraints to solve this problem, but just a few solutions which could be implemented.

4.3.3.1 Fixed Value Constraints

One method for resolving this geometry which is defined in one model and not in the other is to fix the position of the feature in the more specific model relative to another feature in either of the models. This would always give a position for the free geometry.


Unigraphics NX 2 - Modeling - [nbhf.prt (Modified)]
File Edit View Insert Format Tools Assemblies WCS Information Analysis Preferences Application Window Help
Expressions
Unigraphics NX 2 - Modeling - [nbhf.prt (Modified)
Expressions
Any
ew expression

List by: Name
Filter: *
Filter Action: Include
Save Current Filter

```
p7=1
p8=7
p9=100
p10=5
p11=6
p12=p11+.5
p13=1
p14=0
p15=0
p16=10
```

List by: Name
Filter: *
Filter Action: Include
Save Current Filter

```
p7=1
p8=5
p9=100
p10=5
p11=6
p12=p11+.5
p13=1
p14=0
p15=0
p16=10
```

Value =

Figure 11: Fixed value constraint and their inherent problem

However, if this fixed value is too large for the geometry to which the feature is attached, this could cause impossible geometry configurations and will most likely cause severe errors in a CAx tool (see Figure 11).

4.3.3.2 Proportional Constraints

Another possibility for solving this problem is proportional constraints. This sets the position or size value of a constraint proportional to the value of some other feature in one of the models (see Figure 12).

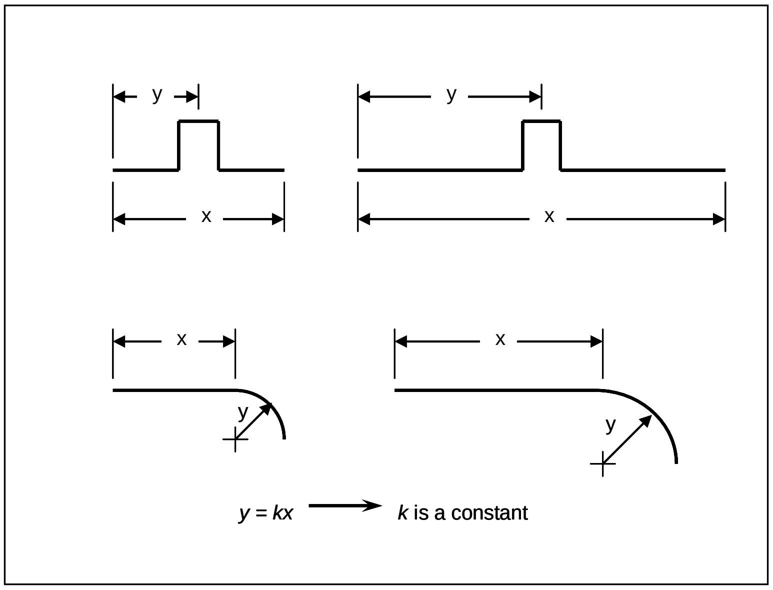


Figure 12: Proportional constraint example

This constraint will only have problems if the value goes to zero and is the value of an arc, cylinder, etc. However, this would require that the value of the feature to which it is constrained also goes to zero.

4.3.3.3 Relative Point Constraints

Still another possibility for a constraint is a relative point constraint. This type of constraint allows certain points on one model to be constrained to points on the other model (see Figure 13). This will be a very generic type of constraint and could have many configurations including points in directions, points on curves with parameter values, etc.

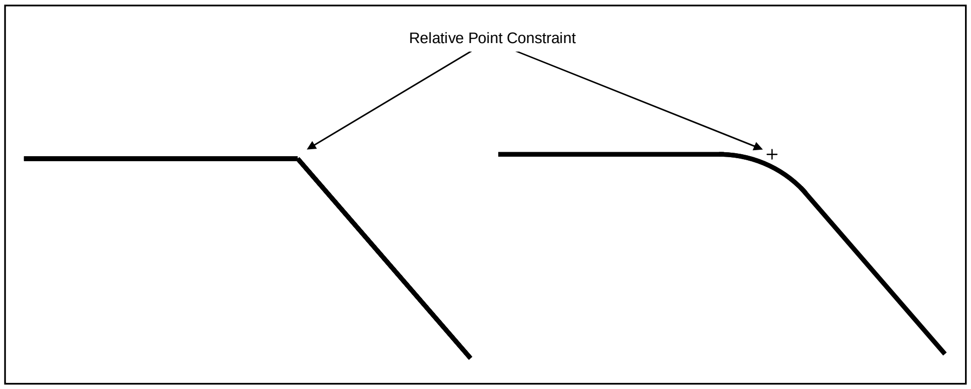


Figure 13: Relative point constraint example

4.4 Link Resolution

As design changes are made, the propagation of those design changes to all members of a model fidelity family will be accomplished by resolving the links between the members of the model fidelity family.

As was implied in section 4.3.1, model geometry elements (curves, surfaces, solids, etc.) need not be stored as thousands of points which approximate the true nature of the element. Instead they can be stored as a set of data from which the element can be calculated at all points. This not only allows for less storage, but allows for computation of links between the models. Thus a model can be represented by a set of points M_i from which the entire model may be described. In a second model in the model fidelity family, the model may be described by the points $\underline{M}_j$. The object of a link is to map the geometry changes of one model to those of another. Thus with when a link is applied, a transfer function between M_i and M_j is implemented as:

$$M_i = A_{ji} M_j \tag{1}$$

Conversely a change in M_j would create a change in M_i such that:

$$M_j = A_{ij} M_i \tag{2}$$

A_ji and A_ij are not the same because the models are not of the same fidelity. Thus $M_\mathrm{i} \neq M_\mathrm{j}$, and the two transfer functions are unique. A simple example illustrates the problem.

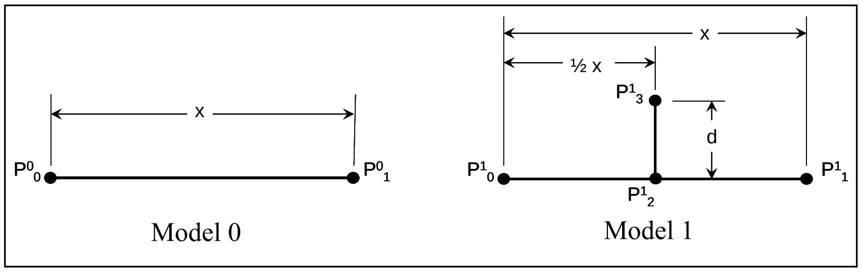


Figure 14: Two models from a model fidelity family to illustrate the transfer functions

Figure 14 shows an example of two models which could be members of the same model fidelity family. The points $P^0{}_0$ and $P^1{}_0$ and the points $P^0{}_1$ and $P^1{}_1$ are constrained coincident to one another respectively. $P^1{}_2$ and $P^1{}_3$ are constrained as shown. If Model 1 is modified, (1) becomes:

$$M_0 = A_{10} M_1 \tag{3}$$

Which can be rewritten as:

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ x_1 \\ y_1 \\ z_1 \end{bmatrix}_0 = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0&0 \end{bmatrix} \bullet \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ x_1 \\ y_1 \\ z_1 \\ x_2 \\ y_2 \\ z_2 \\ x_3 \\ y_3 \\ z_3 \end{bmatrix}_1 \quad (4)$$

Notice that M_0 is independent of points $P^1{}_2$ and $P^1{}_3$. If Model 0 were changed instead, A_{01} would be:

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ x_1 \\ y_1 \\ z_1 \\ x_2 \\ y_2 \\ z_2 \\ x_3 \\ y_3 \\ z_3 \end{bmatrix}_1 = \begin{bmatrix} 1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0 \\ 0&0&1&0&0&0&0 \\ 0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ \frac{1}{2}&0&0&\frac{1}{2}&0&0&0 \\ 0&\frac{1}{2}&0&0&\frac{1}{2}&0&0 \\ 0&0&\frac{1}{2}&0&0&\frac{1}{2}&0 \\ \frac{1}{2}&0&0&\frac{1}{2}&0&0&0 \\ 0&\frac{1}{2}&0&0&\frac{1}{2}&0&d \\ 0&0&\frac{1}{2}&0&0&\frac{1}{2}&0 \end{bmatrix} \bullet \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix}_0 \quad (5)$$

Notice in the case of (5) that $P^1{}_2$ and $P^1{}_3$ are dependant on M_0. Also notice that there is a 1 in M_0 to accommodate for the constant d. From this it can be seen that A_{ij} and A_{ji} are not invertable, and must be determined for each case since not all members of M_i may be dependant on M_j and vice versa.

If the model that is desired to be viewed is not directly linked to the model that is modified, a transfer function between the two models must be identified. This can be found by use of the transfer functions that exist between each of the models inbetween the two models in question. Thus, for the configuration shown in Figure 9, if Model 1 is changed, the solution for Model 8 is:

$$P_8 = A_{18} P_1 \tag{6}$$

$$P_8 = A_{68} A_{65} A_{51} P_1 \tag{7}$$

Thus in order for a model fidelity family to be linked solutions for all of the A_{ij}s in the family must be found. For the example in Figure 14, the transfer functions are easily found through linear means. However, as models become increasingly complex, linear transfer functions become inadequate. Thus a geometric constraint solver which uses both linear and iterative means provides the transfer functions each time a link between members of a model fidelity family is solved.

The geometry of both models will be extracted into the constraint solver domain. This provides all necessary references and geometry for solving the set of constraints within the links. The model which has already been updated according to the design changes will have all of its geometry fixed. The model which is to be updated will be drawn such that geometric constraints within that model remain in effect (i.e. if two surfaces are constrained within the model as being parallel, those surfaces should remain parallel after design changes are made). This allows the design intent of the model to be preserved. It may also reduce the number of constraints needed to fully define the link between two models.

Once the constraint solver has finished solving the system, the geometry is extracted from the constraint solver and the data storage scheme is updated with the geometric data.

4.5 CAx Integration

The methods proposed by Hoffman and Lund provide the ability to extract geometric data from any CAx application into the database and to use that data to recreate the geometry in any view. The method proposed in this thesis will have this functionality as well, and Lund's and Hoffman's works are the basis for how the method will be implemented.

4.5.1 Read/Write Capabilities

The method must have the capability to read and write (or extract and draw) geometry to the CAx packages that will be used to define and modify the geometry of the model fidelity family. The method must be able to translate the geometry into a data structure which can exactly recreate that geometry in the CAx tool in which it was created as well as any other CAx tool which will be used to view the geometry. This process must be automatic and seamless in order for members of a model fidelity family to be added, redrawn, updated, etc.

4.5.2 Update

Updating the geometry of a model fidelity family which will satisfy the objectives and sub-objectives outlined earlier in this chapter provides a unique challenge. Because state-of-the-art CAx tools are often parametric in nature, simply recreating the geometry within the CAx application may not be enough. Thus, a means must be provided for updating the geometry while preserving the parametric constraints of the geometry within the CAx tool. This may be accomplished by several means. Ideally, the constraints which have been imposed on the geometry would be removed, the geometry moved to where it needs to be to satisfy the constraints of the model fidelity family, and the constraints re-imposed. Constraints with numeric values will be updated with the new values.

Because this method can be problematic with various CAx applications in the way that they create and update models (usually a historical feature creation), other methods become much more practical. Another option which can be utilized is to recreate the model geometry completely and replace the original model file with the newly created one. This method, however creates problems with database storage and maintenance. It is desirable to preserve the original model and update it in a new configuration.

Yet another option for updating a model is to recreate the model in a different file and extract the model parameters out of the file. The new parameters are then used to update the original model file. This preserves the original model file and uses a mechanism that parametric CAx applications provide for changing the configuration of a model. Thus it becomes a natural extension of the CAx tool and a non-destructive way of updating model geometry.

There is a problem which comes into play when using parametric CAx applications which allow for knowledge between parameters. Examples of this include expressions based on other expressions. This problem is further discussed in section 6.2.1 Expression

Resolution, and no attempt will be made to solve this problem within the scope of this thesis.

Comment [CGJ1]: The next page is blank is this correct?

Comment [HLS2]: This is a formatting thing.

4.6 Sub-objectives

In addition to satisfying the main objective of transferring data between all members of a model fidelity family, the sub-objectives must also be realized.

4.6.1 Revision Control

The data storage scheme stores the model fidelity family as data outside of the CAx framework. Only when an update of the model fidelity family is called will the data which is stored in the database change. Thus a user may change a model independent of the true state of that model in the design process. This provides an organization a means of revision control and they must determine when a model fidelity family will be updated.

4.6.2 Different Parameterization Schemes

The ability to read and write to a CAx package means the ability to recreate a model with all of its constraints intact. This allows the user to create a unique parameterization scheme for each model within a model fidelity family.

4.6.3 Multi-Directional Dependancies

The link system as described earlier in this chapter is what allows the multi-directional dependencies. The transfer functions which arise from the use of the geometric constraint solvers allows any member of a model fidelity family to update all other members of that family.

CHAPTER 5: RESULTS AND DISCUSSION OF RESULTS

This chapter discusses the results obtained from the implementation of the method described in this thesis. The objective of the method is to create a tool which will link the members of a model fidelity family such that changes made on one member of the family are reflected in all other members of the family. The name of the tool that was implemented is the Geometric Fidelity Linker (GFL).

5.1 Specific Objectives

In addition to performing the previously stated main objective of the method described in this thesis, GFL must also satisfy the following sub-objectives:

1. The method must be able to handle models that have been parameterized differently.
2. The method needs to be able to update all of the models in the model fidelity family from any of the other models in the family in order to accommodate the non-linear nature of the design process.
3. The method needs to incorporate revision control to eliminate confusion as to the true description of the product at the current state of the design phase.

5.2 Approach

The Geometric Fidelity Linker (GFL) was created according to the method described in chapter four of this thesis for the implementation of the method.

5.3 The Geometric Fidelity Linker

A data structure was created to contain the geometric information which would be used in GFL. This was necessary because the implementation of Hoffman's method was unavailable, and Lund's implementation was incomplete at the time this implementation was done. This data structure was created in an object oriented C++ environment. Classes were created to mimic the structure of a feature based CAD application. Models contained expressions and features, and features were polymorphed into their respective definitions (i.e. curves, datums, sketches, etc.). A linker class was created to represent the entire model fidelity family which held the model members and the links between the models. Links contained pointers to models in the model fidelity family and the constraints between those two models (see Figure 15).

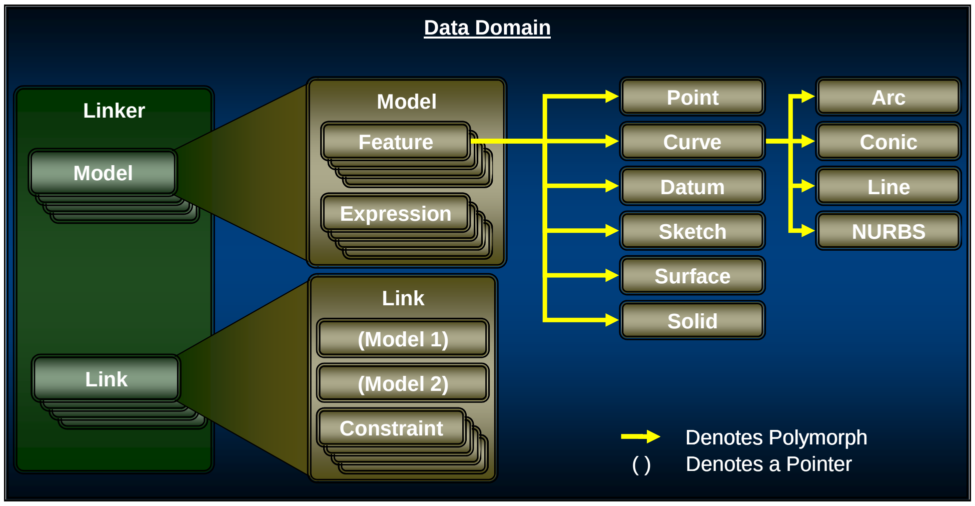


Figure 15: The data structure used in the Geometric Fidelity Linker

These classes were polymorphed into application specific classes to accommodate the CAx tool and the constraint solver. The CAD package used for this implementation was Unigraphics NX2 (NX2). Figure 16 illustrates the polymorphing from the data domain to the NX2 domain. The NX2 classes provide the capability of drawing, extracting and updating the geometric data classes in GFL.

Because there was no appropriate 3D constraint solver available to implement this thesis, the constraint solver inherent in NX2 was utilized. Thus the geometry was restricted to 2D planar geometric elements including curves, lines, arcs, points, planes, axes and sketches. GFL has the ability to extract the information directly from a NX2 model and store it in a GFL library. For this implementation, the library took the form of a text file, but integration into a true database as Lund and Hoffman proposed would follow directly. However, it was not the intent of this thesis to prove database integration, but rather the updating of model fidelity family members from prescribed constraints.

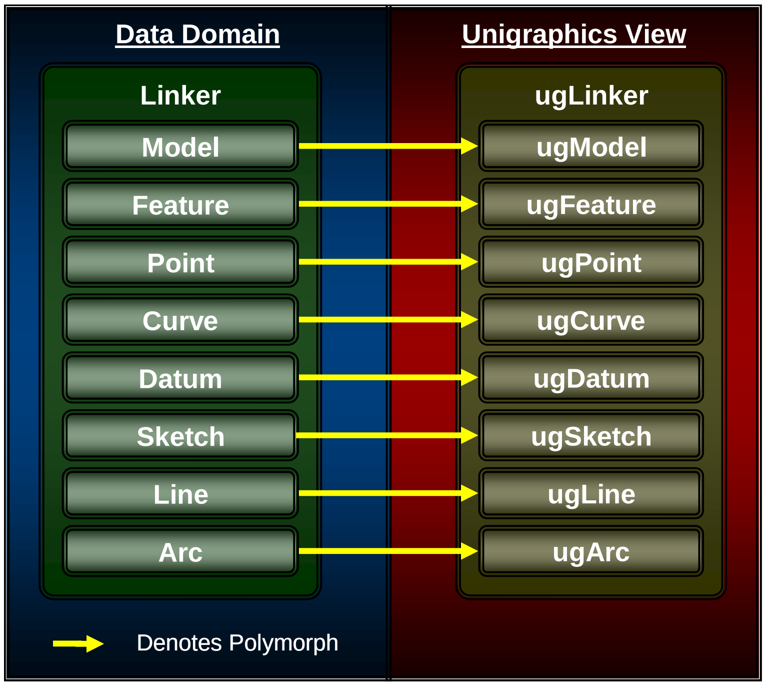


Figure 16: Polymorphing of classes from the data domain to the NX2 domain.

5.3.1 Creating a Model Fidelity Family Using the Geometric Fidelity Linker

Using NX2 as a geometry viewer, GFL allows a user to create a model fidelity family by adding a model to the a GFL library, and then linking other models to the members of that family. Adding a model consists of the following steps:

1. Chose a model fidelity family to add a model to.
2. Chose a model in the family to constrain the new model to.
3. The geometries of both models are extracted into the GFL data structure.

4. The geometries are drawn on top of one another and constraints added between the two models. As constraints are added, the new model is updated according to the topological configuration already in the family.

5. The new model and the constraints between the models are then saved in the library and the application completes.

This method allows a model to be constrained to exactly one other model in the model fidelity family. Because there is no set model to which a new model must be constrained (i.e. the model of least or highest fidelity), there is flexibility in how models may be constrained to one another.

5.3.2 Updating a Model Fidelity Family Using the Geometric Fidelity Linker

Once a model is added to the library, any member of the family may cause an update for the entire family. The updating of the model fidelity family occurs in the following manner:

1. A user modifies a member of a model fidelity family.

2. Once the user is done with modifications, the GFL tool is loaded as a user function within NX2. The GFL tool uses model attributes to determine which family a model belongs to.

3. The model geometry is then extracted and the library updated according to the new model geometry.

4. With the updated data, GFL uses the NX2 constraint solver to again overlay the models to which the modified model is linked and update the linked models

according to the modified model geometry. A recursive algorithm continues through all of the links to solve the model fidelity family for the new geometric configuration.

5. Once all of the models in the model fidelity family have been updated, the library is then saved with the new data.
6. All of the members of the model fidelity family are then opened in their respective NX2 models and the models updated according to the new data. This is accomplished by recreating the geometry with its respective constraints, extracting the expressions of the model and updating the model with the new expressions.

5.3.3 Test Cases

Two test cases were run to prove the method described in this thesis. First was a set of corners models, and second a pair of turbine case models of a gas-turbine engine.

5.3.3.1 Corners Models

The corners models are a set of three models based on a corner which may be found in any geometric model. The geometry is shown in Figure 17.

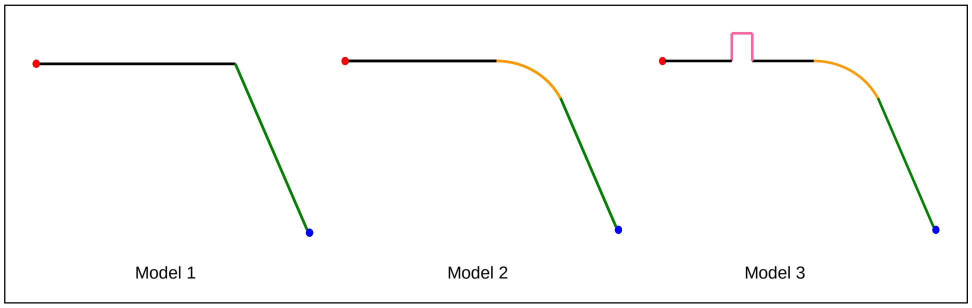


Figure 17: The three corners models

Models 2 and 3 were both linked to model 1. The start points (red) and end points (blue) were constrained coincident to one another. The angled portions (green) were constrained collinearly. The fillets (yellow) were constrained with a fixed radius, and the boss element (pink) was constrained proportionally to the length of the horizontal portion (black).

The models were flexed by changing the angle of the angled portions, the length of the horizontal portion, and the spatial position of the model relative to the WCS of the models. The results and a discussion of those results follow.

5.3.3.1.1 Changing the Angle of the Angled Portions

The models responded well to the change in the angle of the angled portions of the model fidelity family. The entire family updated as expected for each of the members of the family being chosen as the driving model.

5.3.3.1.2 Changing the Length of the Horizontal Portions

Changes in the length of the horizontal portions were more problematic. If the length of the horizontal portion is increased, the models behaved well and updated as expected. However, if the lengths were decreased, the models failed to update as expected. Model 1 updated as expected no matter which model was the driving model, however Models 2 and 3 had problems. The fillet has two possibilities which satisfy the geometric constraints imposed on fillets (coincidence and tangency). In the case where the length of the horizontal portions was shortened, the wrong option was often solved for. All of the constraints were solved for correctly as far as the constraint solver was able, however, it was not sufficient to produce the desired update in model geometry.

5.3.3.1.3 Changing the Spatial Positions of the Models

The problems with the fillets being solved for the unintended case continued in the portion where the spatial position of the model was changed. Also, other situations arose where the wrong case was solved for. For instance a horizontal dimensional constraint was solved for 1 inch on the wrong side of the datum to which it was constrained. This caused the horizontal portion of Model 2 to be 2 inches longer than anticipated. Again, all of the constraints were solved correctly for the constraint solver's ability to do so, but not as desired for this method.

5.3.3.2 Turbine Cases of a Gas-turbine Engine

These constraint solver problems persisted in the second case study, except the complexity of the models increased the complexity of the solutions. This caused completely unusable models. Figure 18 shows the results of a change in spatial position of the models.

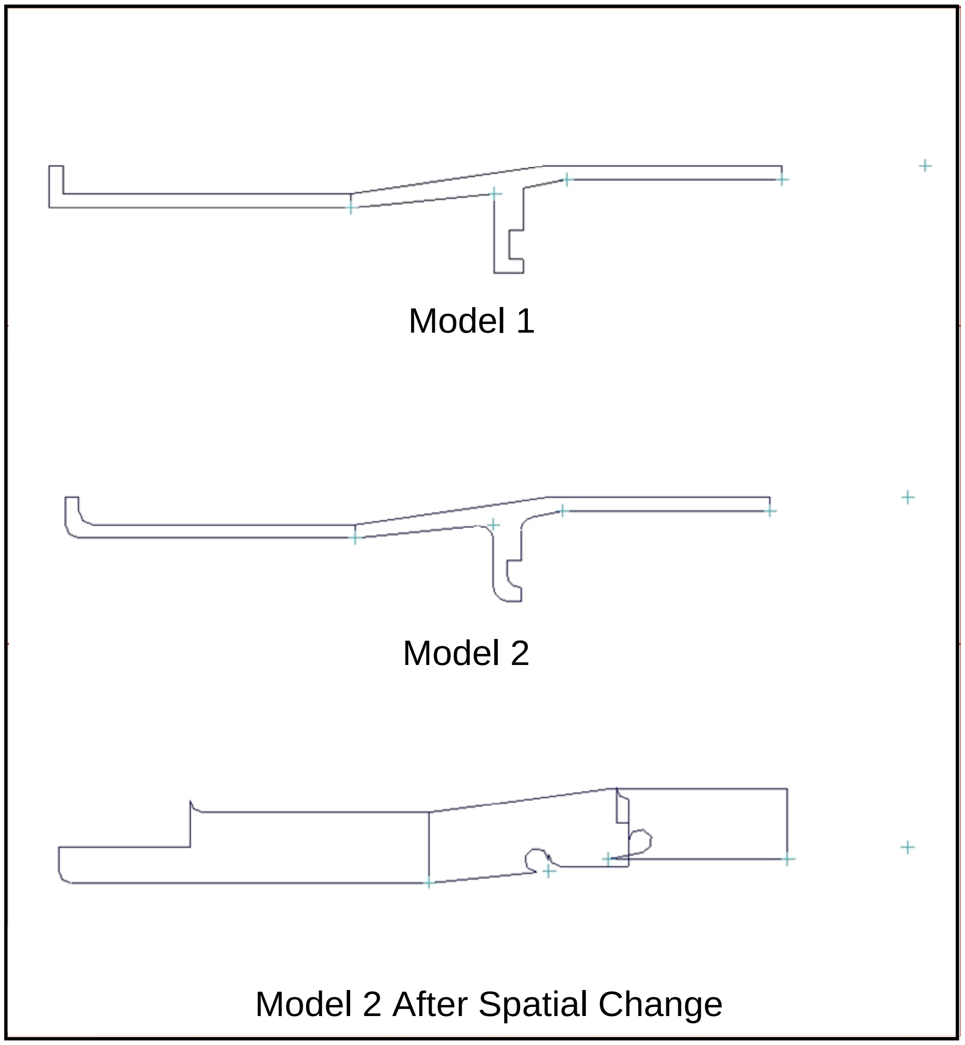


Figure 18: The results of a spatial change in position for the turbine cases case study.

5.3.4 Updating According to Design Intent

The method described in this thesis would be useless if it did not preserve the original design intent of a given model. To this extent, the method allows the models that make up a model fidelity family to be parameterized in unique ways. This preserves the original design intent of the model as it was created. Also, the ability to create constraints between two members of a model fidelity family in unique configurations provides an additional means to preserve design intent of the models which are linked together. See Section 4.1 for a further discussion of design intent in model fidelity families.

The corners models test case helps to show the preservation of the design intent. Changing the angle of the angled portions of the model fidelity family shows that the model updated according to the constraints imposed and the design intent of the link between the two models was preserved. However subsequent tests on the case study showed that there were problems with the method as implemented. Section 5.3.5 is a discussion of these problems.

5.3.4.1.1 Time Performance

For test cases, GFL took about 30 seconds to update the entire family for any given change in a member of the family. The time required to update an arbitrary family is dependant on the number of models in that family and the complexity of the models and the links between the models.

5.3.5 API Problems

In order to integrate GFL into NX2, GFL was written as a user function within the NX2 Application Protocol Interface (API), which is a set of precompiled library files which allow access to most of the functionality of NX2. However, because access to NX2 is limited to API functions and NX2 naturally desires to maintain proprietary definitions to themselves, the ability to integrate into the NX2 environment to the extent desired for the implementation of GFL was not available. Manipulation of geometric data was easily facilitated by the API, however access to the constraint solver in NX2 was very limited. In order to use the constraint solver, all of the geometry had to be defined within a sketch. This forces all geometry within a model to one plane. Ideally the geometry would be created in a constraint solver, and solved without constraints imposed by working within a sketch.

Another problem that arose from working within the NX2 environment was the inability to fully control how the geometry updated. Models of a family would update correctly when the geometrically constrained geometry was solved, but when the parameters were solved, the constraint solver would find an alternate solution for the geometry. Thus all of the constraints were solved for, but the wrong solution was found.

5.4 Analysis of Revision Control

GFL itself is not absolutely tied with any CAx application. It was written specifically for NX2, but is not limited to that application alone. Only the data which defines a model fidelity family is stored in the GFL library. Thus any changes made to a model in a CAx tool would not be reflected in the GFL library unless the library is updated. If changes are made and these changes are not saved into the library and the model fidelity family updated according to the model, the model will remain as it is defined by the data in the GFL library. Thus the GFL library defines the true state of the models which it defines. Only when the library is updated are the models changed. Thus the engineering groups which own the models are free to make changes to those models without affecting the data in the GFL library. The models are in no way restricted by the links which bind them to the other members of the model fidelity family until an update to the family is instantiated. Thus the GFL library defines the true state of the model fidelity family, and any models only define proposed design changes. Final design changes will be integrated into the model fidelity family with a GFL update.

5.5 Analysis of Different Parameterization Schemes

Excluding the issues addressed in Section 5.3.5 regarding the constraint solver, the models in the test case updated correctly with the model fidelity family. The parameterization schemes of the various models remained intact and GFL was able to update the library despite the variation in parameterizations.

In light of the issues arising from multiple solutions by the constraint solvers, it may be possible to constrain the models in a model fidelity family in such a way as to produce a unique solution every time the family is updated. This requires an intimate understanding of the problems associated with this multiple solution problem. It may also take away from the model designer the freedom to parameterize a model in such a way as to include the design intent of the model. The inclusion of design intent in the model is the driving factor which necessitates members of a model fidelity family being parameterized differently. Thus this situation was not explored further.

5.6 Analysis of Multi-directional Dependencies

The corners model fidelity family consisted of three models linked together. The model fidelity family updated no matter which model was modified, despite the fact that the geometry did not update as desired. Thus GFL provides a way for models to update independent of addition order to the model fidelity family or any fidelity level structure which would be assumed. Each model can facilitate change in the entire model fidelity family.

CHAPTER 6: CONCLUSION

The main objective of this thesis is to provide a method for propagating changes in one member of a model fidelity family throughout the entire family. Additionally, the method should do this regardless of the way in which the design intent was prescribed or the models were parameterized, allowing any of the members of the family to be the modified model which drives change in all other members of the family, and provide a means for controlling the true definition of a product as it develops. This section draws conclusions about the proposed method and gives suggestions for further research and possibilities.

6.1 Automatic Updating of a Model Fidelity Family

The implementation of the method described in this thesis shows that members of a model fidelity family may be tied together through geometric links. This method demonstrated that geometric model changes are distributed throughout the model fidelity family in a manner of seconds or minutes as opposed to the hours or days that it could take to update those models manually.

6.1.1 Revision Control

The method described in this thesis provides the means for working on a member of a model fidelity family without affecting the model stored in the database. This allows a group to work on its model independent of the model fidelity family. In fact it is not until a user updates the model fidelity family that the data which has changed in a group's model will be distributed to the family. Thus changes can be carefully considered before they become final and are distributed into the library. Each revision of a model fidelity family may be saved in the database, and the ability to revert to previous family configurations provides another means of revision control. Further methods should be employed to manage how and when the model fidelity family will be updated to reflect changes which any group may make. This will depend on the way that an organization works and how they would individually use this method.

6.1.2 Different Parameterization Schemes

The ability of a group to determine the driving parameters of a model and how those parameters will determine the final outcome of that model allows the designer to place the design intent into the model. The method described in this thesis allows this to occur. As the implementation now stands, it may be possible to parameterize the models of a model fidelity family such that they would update without the multiple solution problems encountered, but this does not allow the user the freedom of using the parameterization schemes to define a model's design intent. The development of directional constraints may alleviate this problem. See Section 6.1.4 for a discussion of directional constraints.

6.1.3 Multi-directional Dependencies

Removing the strict dependency of higher fidelity models on lower fidelity models allows all phases and levels of design to happen simultaneously and give input simultaneously. This can create organizational problems. However the organization which uses this method will have to decide on how best to resolve the true design process. This method allows flexibility in defining that design process, instead of forcing the organization into a predefined design method.

6.2 Projected Contributions

The implementation of the method presented in this thesis could have an incredible impact if used in a commercial setting. This method would allow an initial definition of a product at the conceptual level. Other models within the model fidelity family which already exists for further product definition will automatically reflect the conceptual design. Immediately work could begin on those other models at all levels as the organization's design process prescribes. As more information becomes available, the model fidelity family will be updated. Geometry checks of the model fidelity family against design criteria can be made by the organization outside of this method at all levels of design from conceptual to final and even manufacturing. These changes can be shown immediately in the model fidelity family with this method. This increases the ability of groups within a corporation to communicate and share information. This also facilitates the propagation of this information automatically, decreasing the need for manually updating those models in the family. This eliminates potentially hundreds of man hours in mundane parameter entry and calculation and eliminates many potential mistakes that would occur due to human error in the updating process.

This method can be extended to other CAx applications. The model information saved in the database can also be extended to spreadsheets, documents, manufacturing applications and even proprietary applications which require the geometric data within the database. This would further eliminate the time spent in mundane data entry tasks and help eliminate the human errors which accompany the manual data entry. The potential for time savings due to automatic data migration and rework due to mistakes is incalculable.

This thesis shows the usability of the proposed work. Much work still remains in order to make it a commercially viable method. However, as constraint, database and CAx technology advance into the realms needed to fully implement this method into a useful solid geometry tool (as those are their current directions) the full potential of the method described in this thesis may be realized.

6.3 Future Work

Future work on this method may be done in the areas of directional constraints to absolve the problem with multiple solutions to constraints, indication that geometry models are fully constrained, expression resolution, 3D implementation, management scheme for when updates are accepted into the database, homogeneous transformations for differently oriented models, assemblies, topology changes, model attributes, and other engineering applications.

6.3.1 Directional constraints to absolve the problems with multiple solutions to constraints

As it currently stands, NX2 does not support directional constraints within its sketching environment. A directional constraint differs from a conventional constraint in that the constraint has an origin and direction. This lack of directionality allows for multiple solutions within the constraint solver.

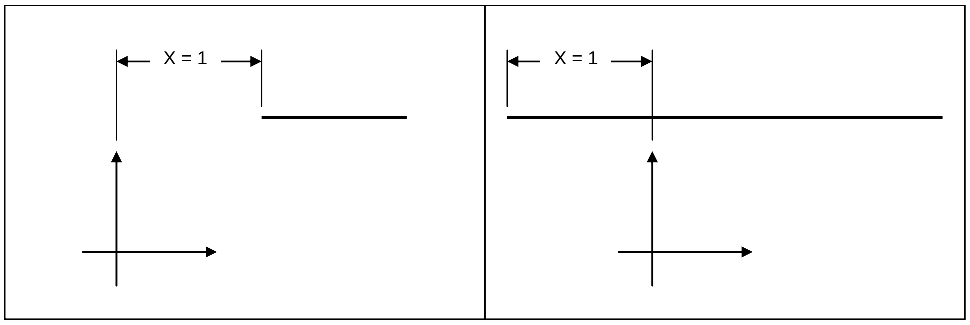


Figure 19: The problem of multiple solutions of a given constraint

Figure 19 shows an example of how a conventionally constrained horizontal line can have two solutions which satisfy the constraints. Figure 20 shows the same horizontal line constrained with directional constraints.

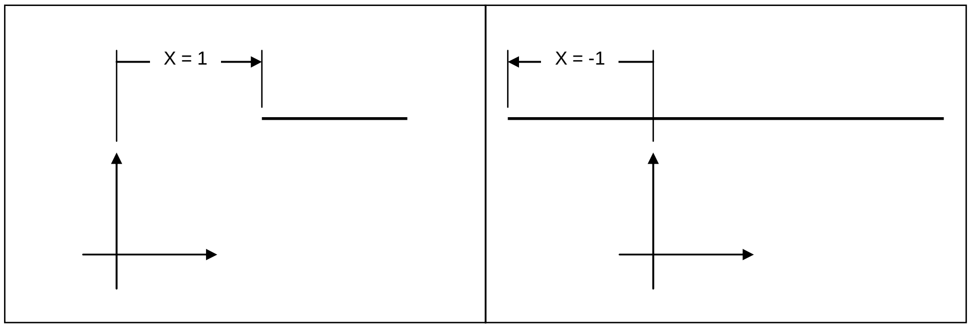


Figure 20: An example of a directional constraint

In order for the configuration on the right in Figure 19 to be realized, the value of the X would have to be –1 as shown by the configuration on the right of Figure 20. Thus a directional constraint gives a unique solution for a constraint scheme. Because NX2 does not support directional constraints, either the models must be constrained such that the design space of the model will not allow for multiple configurations, or the risk of an undesirable configuration must be accepted. If directional constraints were available in future constraint solvers, this could solve the main issues which arose in the implementation of this thesis (see Section 5.3.7).

6.3.2 Indication that geometry models are fully constrained

The method as it was implemented in this thesis made no attempt to ascertain whether or not the models were fully constrained to one another, but assumed that the user was able to intuitively know when they were. This works fine for very simple models, but as the models become more complex, it becomes necessary to know if the models are fully constrained or not. This should be implemented into the method.

6.3.3 Expressions resolution

The method described in this thesis does not take into account the use of logic in expressions. This logic takes the forms of algebraic, trigonometric and logic functions which relate expressions to one another. Thus an expression which defines the physical position of an element in a geometric model may be the computation of a function involving one or more other expressions. The method in this thesis begins with this numeric value, and the numeric values of the expressions which make up the function must be solved for. For example, if the constraint x in Figure 21 had the value of y + 5 instead of a purely numeric value, y would have to be solved for as well. In this example, the solution is straightforward and simple. However, a means needs to be created to resolve all of the parameters defined within the CAx application to update that particular model.

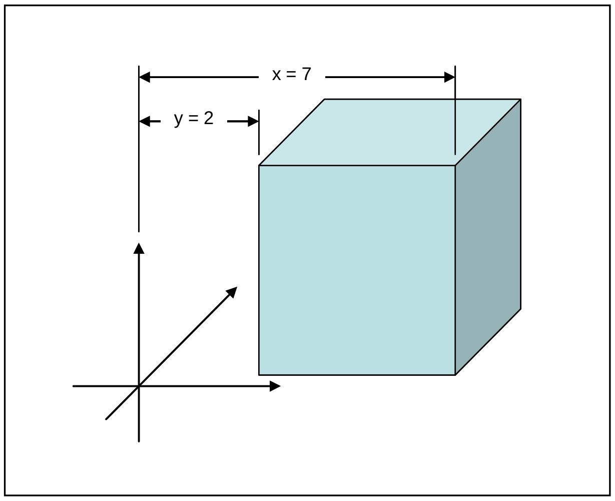


Figure 21: Example of how expressions work

6.3.4 3D implementation

The ability to implement this method in three dimensional space is the basis for its greater usefulness. The 2D implementation itself is insufficient to prove its usefulness in solid modeling applications. However, as explained in Section 2.3.2, no appropriate constraint solver exists at this time. As constraint solver technology advances, especially in the 3D realm, efforts should be made to implement this method with those constraint solvers to realize the full potential of this method.

6.3.5 Management scheme for when updates are accepted into the database

There are no control measures taken to control when a model fidelity family is updated according to changes in a member model. Implementation into methods such as Hoffman[18] and [19] or Lund[23] should include a management method for this type of updating. Also, the management and organization of a corporation could determine how the model fidelity family would be updated. Without this, any group could update the model fidelity family at any time, regardless of any other groups' needed changes, undoing important additions to a product and hindering work. This would simply result in chaos within the corporation.

6.3.6 Homogenous transformations for differently oriented models.

There is no guarantee that members of a model fidelity family will be oriented the same in relation to the world coordinate system (WCS) of the model. The WCS is the origin and XYZ directions which define the space in which a model is created. How a designer orients a model relative to that WCS is dependant on such things as ease of model creation, company standards and personal preference. A provision needs to be made to orient the models such that they are in the same coordinate system. This can be resolved by means of a homogenous transformation which describes the rotation and displacement necessary to place the models in the same space and orientation.

6.3.7 Assemblies

This thesis did not consider assemblies, however expansion of this method to assemblies would be a natural extension, and should be considered for practical industrial applications. The extension to assemblies would be rather straight forward.

6.3.8 Topology changes

This method does not consider topology changes except that if topology changes are made, the model whose topology changed could become another member of the model fidelity family and the old model removed from the family. However, a more user friendly method would be to allow those changes and the reapplication of constraints to the model fidelity family.

6.3.9 Model Attributes

The method implemented in this thesis made no attempt to account for model attributes, but only considered geometry. However, attributes are an important aspect of parametric modeling, and should be incorporated into this method. Several researchers have created methods for automatically mapping attributes between models, and these methods could be used to incorporate attributes into this method [24][25][26].

6.3.10 Other Engineering Applications

This method can be applied to many other engineering applications than CAx tools. For instance, model parameters may be saved in spreadsheets, databases and proprietary applications. This method could be extended to include all of these and potentially many others. The ramifications of extending this method to other engineering applications would mean even greater fluidity in data migration throughout a corporation. The constraint solver could be one which is not devoted solely to geometric applications, or several different constraint solvers could be used, depending on the type of data which is to be incorporated into a model fidelity family.

REFERENCES

[1] www.ugs.com/products/nx, Posted Sept. 2004.

[2] W. Bouma, et al., “Geometric constraint solver,” Computer Aided Design, vol. 27, no. 6, pp487-501, 1995.

[3] I. Fudos and C. M. Hoffman, “Correctness proof of a geometric constraint solver,” International Journal of Computational Geometry & Applications, vol. 6, no. 4, pp 405-420, 1996.

[4] Christoph M. Hoffman, et al., “Making constraint solvers more usable: overconstraint problem.” Computer Aided Design, vol. 36, no. 6, pp. 377-399, 2004.

[5] A Ershov, et al., “LGS: Geometric constraint solver,” Lecture Notes in Computer Science, vol. 2890, pp. 423-430.

[6] K. Y. Lee, et al., “A hybrid approach to geometric constraint solving with graph analysis and reduction”, Advances in Engineering Software, vol. 34, no. 2, pp. 103-113, 2003

[7] C. Castro and E. Monfroy, “Basic operators for solving constraints via collaboration of solvers,” Journal of Computer Science and Technology, vol. 17, no. 3, pp. 314-323, 2002.

[8] J.S. Kim, et al., “Solving 3D geometric constraints for closed-loop assemblies,” International Journal of advanced Manufacturing Technology, vol. 23, no. 9-10, pp. 755-761, 2004.

[9] Joachim Altmeyer, et al., “Reuse of Design Object in CAD Frameworks,” IEEE/ACM International Conference on Computer Aided Design, Digest of Technical Papers, pp. 754-761, 1997.

[10] M. Koegst, et al., “A Systematic Analysis of Reuse Strategies for Design of Electronic Circuits,” Proceedings of the Conference on Design, Automation and Test in Europe, pp. 292-296, 1998.

[11] Erik E. Hayes, et al., "Representation of Temporal Change in Solid Models," Proceedings of the Sixth ACM Symposium on Solid Modeling and Applications, pp. 317-318, 2001.

[12] Bronsvoort, et al., "Multiple-view feature modeling and conversion," Geometric Modelling: Theory and Practice – The State of the Art, pp. 159-174 1997.

[13] R. Bidarra and W. F. Bronsvoort, "History-independent boundary evaluation for feature modeling," CD-ROM Proceedings of the 1999 ASME Design Engineering Technical Conferences, 1999.

[14] R. Bidarra and W. F. Bronsvoort, "On families of objects and their semantics," Proceedings of Geometric Modeling and Processing 2000 – Theory and Applications, pp. 101-111, 2000.

[15] R. Bidarra, et al., "Boundary evaluation for a cellular model," CD-ROM Proceedings of the 2003 ASME Design Engineering Technical Conferences & Computer and Information in Engineering Conference, 2003.

[16] K. J. de Kraker, et al., "Maintaining multiple views in feature modeling," Proceedings Solid Modelling '97, Fourth ACM Symposium on Solid Modelling and Applications, pp. 123-130, 1997.

[17] W. F. Bronsvoort and A. Noort, "Multiple-view feature modeling for integral product development," Computer-Aided Design, vol. 36, no. 10, pp. 929-946, 2004.

[18] Christoph M. Hoffman and Robert Joan-Arinyo, "CAD and the product master model," Computer-Aided Design, vol. 30, no. 11, pp. 905-918, 1998.

[19] Christoph M. Hoffman and Robert Joan-Arinyo, "Distributed maintenance of multiple product views," Computer-Aided Design, vol. 32, no. 7, pp. 421-431, 2000.

[20] www.engineous.com/product_FIPER.htm, Posted Feb. 2005.

[21] www.ugs.com/products/teamcenter, Posted Sept. 2004.

[22] Nathanial Luke Fife, "Developing a design space model using a multidisciplinary design optimization schema in a product lifecycle management system to capture knowledge for reuse," M.S. thesis, Brigham Young University, Provo, 2005.

[23] Jonathan Lund, "Parametric Product Lifecycle Management" M.S. thesis, BrighamYoung University, Provo, 2005.

[24] Matthew King, "A CAD-centric Approach to CFD Analysis With Discrete Features" M.S thesis, Brigham Young University, Provo, 2005.

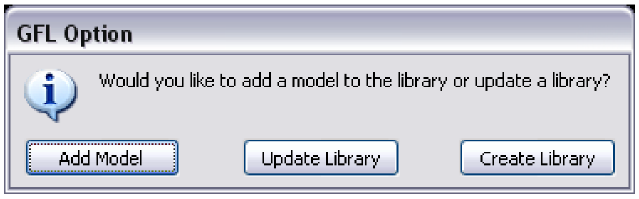
[25] Tyson Baker, “Attribution Standardization for Integrated Concurrent Engineering” M.S. thesis, Brigham Young University, Provo, 2005.

[26] Jonathan Shelley, “Incorporating Computational Fluid Dynamics Into The Preliminary Design Cycle” M.S. thesis, Brigham Young University, Provo, 2005.

APPENDIX

APPENDIX A: GRAPHICAL USER INTERFACE

This appendix shows the graphical user interface (GUI) for the Geometric Fidelity Linker (GFL). If the user selects the Add Model selection in Figure 22, the GUIs shown in Figures 23-25 follow, leading the user to add a model to a model fidelity family. If the Update Library option in Figure 22 is chosen, the library is automatically updated and the GUI is exited. If the Create Library option is chosen in Figure 22, the user is prompted for a library name and a model to start the library with and then the GUI is exited.


GFL Option

Would you like to add a model to the library or update a library?

Figure 22: Initial Option GUI

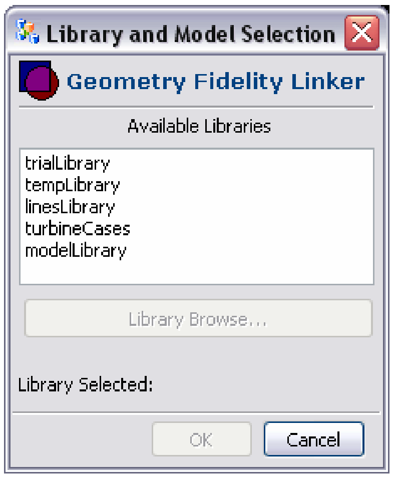
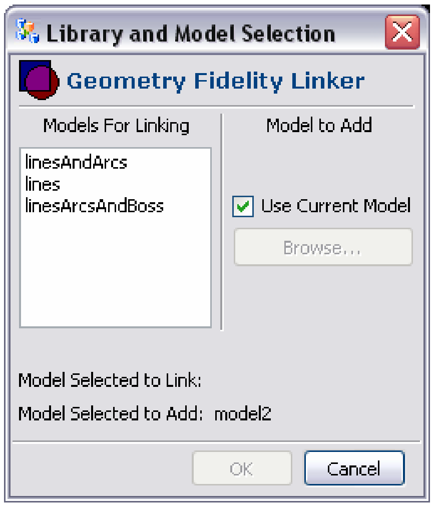


Figure 23: Library Selection GUI

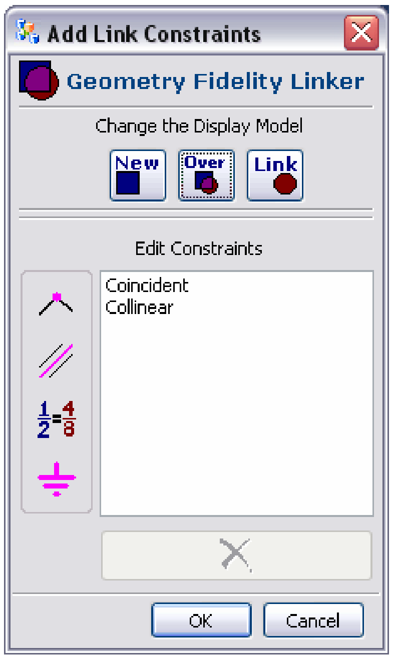


Model Selected to Link:

Model Selected to Add: model2

[✓ Use Current Model]

Figure 24: Model Selection GUI



Coincident
Collinear

Figure 25: Link Constraints GUI

APPENDIX B: GEOMETRIC FIDELITY LINKER CODE

This appendix contains pertinent header file and code to help in understanding the structure of the Geometric Fidelity Linker (GFL). Figure 15 shows a graphical view of this code.

hLinker Object Header File:

```
#ifndef HLINKER_HPP_INCLUDE
#define HLINKER_HPP_INCLUDE

#include "lgsLink.hpp"
#include "hModel.hpp"
#include "hLibrary.hpp"

class hLinker : public hTypeBase
{
public:
   hLinker(void);
   ~hLinker(void);

public:
   //---------------------------------------//
   //   Functions which can be Overridden    //
   //---------------------------------------//

   virtual bool      initializeCADFiles(vector<string> fileNames);
   virtual bool      initializeConstraintSolver(){return false;};
   virtual void      createLinkingModel(){};
   virtual void      readLibrary(string fileName);
   virtual void      writeLibrary(string fileName = "");


   //---------------------------------------//
   //       Class Specific Functions         //
   //---------------------------------------//

   bool              initialize(string libraryFileName);
   string            getLibraryName();

   void              addModel(hModel* model);
   hModel*           getModel(int which);
   int               getModelNum(hModel* model);
   void              setModel(int which, hModel* model);
   void              removeModel(int which);
   int               getNumModels(void);

   void              addLink(hLink* link);
   void              setLink(hLink* link, int which);
   hLink*            getLink(int which);
   void              removeLink(int which);
   int               getNumLinks(void);

   void              setLinkingModel(hModel* linkingModel);
   hModel*           getLinkingModel(void);

   void              setNewModel(hModel* newModel);
   hModel*           getNewModel(void);

   void              solve(hModel* lastUpdatedModel);

protected:

   vector<hModel*>   mModels;
   vector<hLink*>    mLinks;
   hLibrary          mLibrary;
   hModel*           mLinkingModel;
   hModel*           mNewModel;
};

#endif //HLINKER_HPP_INCLUDE
```

hLink Object Header File:

```
#ifndef HLINK_HPP_INCLUDE
#define HLINK_HPP_INCLUDE

#include "hConstraint.hpp"
#include "hModel.hpp"

class hLink
{
public:
 hLink(void);
 hLink(hModel* model1, hModel* model2);
 ~hLink(void);

 void              addConstraint(hConstraint* constraint);
 void              updateConstraints(void);
 void              removeConstraint(hConstraint* constraint);
 void              removeConstraint(unsigned int which);
 hConstraint*      getConstraint(int which);
 int               getNumConstraints(void);

 virtual bool      solveSystem(int
                   whichModelHasAlreadyChanged){return false;};
 virtual bool      solveSystem(string
                   modelNameThatHasAlreadyChanged){return false;};

 virtual void      read(hLibrary& library);
 void              write(hLibrary& library);

 void              addModel(hModel *model);
 void              setModel(int which, hModel *model);
 hModel*           getModel(int which);

protected:
 int               findConstraint(hConstraint* constraint);

 hModel*           mModels[2];
 vector<hConstraint*>    mConstraints;
};

#endif //HLINK_HPP_INCLUDE
```

hConstraint Object Header File:

```
#ifndef HCONSTRAINT_HPP_INCLUDE
#define HCONSTRAINT_HPP_INCLUDE

#include "cppInclude.hpp"
#include "hLibrary.hpp"
#include "hFeature.hpp"
#include "hExpression.hpp"
#include "hPoint.hpp"

enum GFL_CONSTRAINT_TYPE {GFL_NO_CONSTRAINT = -1,
                          GFL_FIXED,
                          GFL_HORIZONTAL,
                          GFL_VERTICAL,
                          GFL_PARALLEL,
                          GFL_PERPENDICULAR,
                          GFL_COLLINEAR,
                          GFL_EQUAL_LENGTH,
                          GFL_CONSTANT_ANGLE,
                          GFL_COINCIDENT,
                          GFL_CONCENTRIC,
                          GFL_MIRROR,
                          GFL_POINT_ON_CURVE,
                          GFL_MIDPOINT,
                          GFL_TANGENT,
                          GFL_RADIUS_DIM,
                          GFL_DIAMETER_DIM,
                          GFL_HORIZONTAL_DIM,
                          GFL_VERTICAL_DIM,
                          GFL_PARALLEL_DIM,
                          GFL_PERPENDICULAR_DIM,
                          GFL_ANGULAR_DIM,
                          GFL_POINT_ON_STRING,
                          GFL_SLOPE,
                          GFL_UNIFORM_SCALED,
                          GFL_PERIMETER_DIM,
                          GFL_FIXED_RADIUS,
                          GFL_PROPORTIONAL,
                          GFL_FIXED_VALUE};

enum GFL_CONSTRAINT_CATEGORY {GFL_NO_CATEGORY = -1,
                              GFL_GEOMETRIC_CONSTRAINT,
                              GFL_DIMENSIONAL_CONSTRAINT};

enum GFL_VERTEX_TYPE {GFL_NO_VERTEX = -1,
                      GFL_START_VERTEX,
                      GFL_END_VERTEX,
                      GFL_CENTER_VERTEX,
                      GFL_SPLINE_DEFINING_POINT_VERTEX,
                      GFL_ANCHOR_VERTEX,
                      GFL_TANGENT_VERTEX,
                      GFL_END_OF_VERTEX_TYPES};

class hConstraint : public hTypeBase
{
public:
 hConstraint(void);
 ~hConstraint(void);

 //functions to be overriden in derived constraint classes
 GFL_CONSTRAINT_TYPE           getConstraintType();
 virtual void
 setConstraintType(GFL_CONSTRAINT_TYPE type);

 virtual GFL_CONSTRAINT_CATEGORY getConstraintCategory(void);
 void              addFeature(hFeature* feature);
 void              setFeature(hFeature* feature, int which=0);
 hFeature*         getFeature(int which=0);
 vector<hFeature*> getFeatures(void);

 void        setVertex(enum GFL_VERTEX_TYPE type, int which = 0);
 GFL_VERTEX_TYPE   getVertex(int which = 0);

 void        setIndexParameter(int parameter, int which = 0);
 int         getIndexParameter(int which);

 void        setUseHelp(int useHelp, int which = 0);
 int         getUseHelp(int which = 0);

 void        setHelpPoint(hPoint pt, int which = 0);
 hPoint      getHelpPoint(int which = 0);

 void        setHelpParameter(double val, int which = 0);
 double      getHelpParameter(int which = 0);

 void        setExpression(hExpression* expression);
 hExpression* getExpression(void);

 void        setValue(double val);
 double      getValue(void);

 string      getDiscriptorString(void);
 virtual void read(string& line, vector<unsigned int> &featureIDs,
                   vector<unsigned int> &modelIDs, int
                   *expressionLinkerID = NULL);
 virtual string    write(void);

 virtual bool      operator==(hConstraint* constraint);
 virtual void      operator=(hConstraint* constraint);

protected:
 vector<hFeature*>       mFeatures;
 vector<GFL_VERTEX_TYPE> mVertices;
 vector<int>             mIndexParameters;
 vector<int>             mUseHelps;
 vector<hPoint>          mHelpPoints;
 vector<double>          mHelpParameters;
 GFL_CONSTRAINT_TYPE     mConstraintType;
 hExpression*            mExpression;
 double                  mValue;
};

#endif //HCONSTRAINT_HPP_INCLUDE
```

hModel Object Header File:

```
/*****************************************************************/
/* hModel.hpp                                                    */
/* Definition of the GFL hModel class - can also be thought    */
/* of as an assembly or part.  It allows for having hModels    */
/* inside it to facilitate both UG and CATIA assembly schemes  */
/*                                                               */
/* Created by:  Hans Soderquist                                  */
/* Created on:  November 29, 2004                                */
/* Last Revised:  November 29, 2004                              */
/*****************************************************************/

#ifndef HMODEL_HPP_INCLUDE
#define HMODEL_HPP_INCLUDE

#include "cppInclude.hpp"
#include "hPoint.hpp"
#include "hLine.hpp"
#include "hArc.hpp"
#include "hExpression.hpp"
#include "hTypeBase.hpp"

enum GFL_OBJECT_TYPE {GFL_NO_OBJECT = -1,
                      GFL_OBJ_POINT,
                      GFL_OBJ_LINE,
                      GFL_OBJ_ARC,
                      GFL_OBJ_CONIC,
                      GFL_OBJ_SPLINE};

class hModel : public hTypeBase
{
```

```
public:
   hModel(void);
   ~hModel(void);

public:
 virtual void      read(hLibrary& library);
 virtual void      write(hLibrary& library);

 virtual void      setLinkerID(unsigned int linkerID);

   void            addFeature(hFeature* comp);
   void            deleteFeature(int which);
   hFeature*       getFeature(int which);
 int               getFeature(hFeature* feature);
 vector<hFeature*> getFeatures(void);
   bool            setFeature(int which, hFeature* comp);
   int             getNumFeatures();

   void            setModelName(string name);
   string          getModelName();
   void            addModel(hModel* model);
   hModel*         getModel(int which);
 int               getNumModels(void);

 void              addExpression(hExpression* expression);
 hExpression*      getExpression(int which);
 vector<hExpression*> getExpressions(void);
 int               getNumExpressions(void);

 void              operator<<(hModel*);
 virtual void      operator=(hModel*);

 void              updateFromSolver(hModel* model);

protected:
   string             mModelName;
   vector<hFeature*>  mFeatures;
   vector<hModel*>    mModels;
 vector<hExpression*> MExpressions;

private:
 hFeature*   setFeaturePolymorph(hFeature* feature, int type);
 hFeature*   setCurvePolymorph(hFeature* curve, int type);
};

#endif //HMODEL_HPP_INCLUDE
```

hFeature Object Header File:

```
#ifndef HFEATURE_HPP_INCLUDE
#define HFEATURE_HPP_INCLUDE

#include "hExpression.hpp"
#include "hTypeBase.hpp"
```

```
enum GFL_FEATURE_TYPE {GFL_NO_FEATURE = -1,
                       GFL_POINT_FEATURE,
                       GFL_CURVE_FEATURE,
                       GFL_CONTOUR_FEATURE,
                       GFL_SKETCH_FEATURE,
                       GFL_DATUM_FEATURE,
                       GFL_SOLID_FEATURE,
                       GFL_SURFACE_FEATURE,
                       GFL_CSYS_FEATURE};

class hFeature : public hTypeBase
{
public:
 hFeature(void){};
 ~hFeature(void){};

 virtual void             read(hLibrary& library){};
 virtual void             write(hLibrary& library){};
 virtual void             read(string& line){};
 virtual string           write(void){return "";};

 virtual GFL_FEATURE_TYPE getType(void){return GFL_NO_FEATURE;};

 virtual void            setFeature(hFeature* feature, int
which){};
 virtual void
 setFeatures(vector<hFeature*> features){};
 virtual vector<hFeature*>     getFeatures(void){vector<hFeature*>
                               features;return features;};

 virtual void setCurves(vector<hFeature*> curves){};
 virtual void setExpressions(vector<hExpression*> expressions){};
 virtual void setInt(int num){};
 virtual void setBool(bool val){};

 virtual void getDirection(double dir[3]){};

 virtual void setName(string name){mName = name;};
 virtual string getName(void){return mName;};

 virtual bool isContainer(void){return false;};

 virtual bool operator== (hFeature* feature){return false;};
 virtual void operator= (hFeature* feature){};

 void        setOwningModel(int modelNumber)
                          {mOwningModel = modelNumber;};
 int         getOwningModel(void){return mOwningModel;};

protected:
 string      mName;
 int         mOwningModel;
};

#endif //HFEATURE_HPP_INCLUDE
```

hExpression Object Header File:

```
#ifndef HEXPRESSION_HPP_INCLUDE
#define HEXPRESSION_HPP_INCLUDE

#include "hTypeBase.hpp"

class hExpression : public hTypeBase
{
public:
 hExpression(void);
 ~hExpression(void);

 virtual void      read(string &line);
 virtual string    write(void);

 //to be overriden in child classes
 virtual void updateExpressionName(void){};

 void        setName(string name);
 string      getName(void);
 void        setValue(string value);
 string      getValue(void);

 bool        askIsOwned(void);
 void        setIsOwned(bool owned);

 string      getExpression(void);

 bool        operator==(hExpression* expression);
 void        operator=(hExpression* expression);

protected:
 string      mName;
 string      mValue;
 bool        isOwned;
};

#endif //HEXPRESSION_HPP_INCLUDE
```