2016-03-01

Neutral Parametric Database, Server, Logic Layers, and Clients to Facilitate Multi-EngineerSynchronous Heterogeneous CAD

Kelly Eric Bowman
Brigham Young University - Provo

Neutral Parametric Database, Server, Logic Layers, and Clients to Facilitate Multi-Engineer Synchronous Heterogeneous CAD

Kelly Eric Bowman

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Mechanical Engineering

Brigham Young University

March 2016

ABSTRACT

Neutral Parametric Database, Server, Logic Layers, and Clients to Facilitate Multi-Engineer Synchronous Heterogeneous CAD

Kelly Eric Bowman
Department of Mechanical Engineering, BYU
Doctor of Philosophy

Engineering companies are sociotechnical systems in which engineers, designers, analysts, etc. use an array of software tools to follow prescribed product-development processes. The purpose of these amalgamated systems is to develop new products as quickly as possible while maintaining quality and meeting customer and market demands. Researchers at Brigham Young University have shortened engineering design cycle times through the development and use of multi-engineer synchronous (MES) CAD tools. Other research teams have shortened design cycle-times by extending seamless interoperability across heterogeneous design tools and domains. Seamless multi-engineer synchronous heterogeneous (MESH) CAD environments is the focus of this dissertation. An architecture that supports both MES collaboration and interoperability is defined, tested for robustness, and proposed as the start of a new standard for interoperability. An N-tiered architecture with four layers is used. These layers are data storage, server communication, business logic, and client. Perhaps the most critical part of the architecture is the new neutral parametric database (NPDB) standard which can generically store associative CAD geometry from heterogeneous CAD systems. A practical application has been developed using the architecture which demonstrates design and modeling interoperability between Siemens NX, PTC's Creo, and Dassault Systemes CATIA CAD applications; Interoperability between Siemens' NX and Dassault Systemes' CATIA are specifically outlined in this dissertation. The 2D point, 2D line, 2D arc, 2D circle, 2D spline, 3D point, extrude, and revolve features have been developed. Complex models have successfully been modeled and exchanged in real time across heterogeneous CAD clients and have validated this approach for MESH CAD collaboration.

Keywords: Multi-user CAD, Heterogeneous CAD

ACKNOWLEDGMENTS

Research of this type does not happen in a vacuum. I would like to thank those who have worked on this problem before me because their ideas were inspirational. I would like to thank all the members of the BYU CADLab, including Dr. Jensen, Dr. Red, and Dr. Salmon. Thanks go to Dr. Tew for his contributions to our server. I could not have done this without Raytheon Missile Systems or the members of the National Science Foundation Center for e-Design: Belcan, Boeing, PCC Airfoils, Pratt & Whitney, CD-adapco, UTC Aerospace Systems, and ARDEC.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LISTINGS

NOMENCLATURE

API	Application Programming Interface
BREP	Boundary Representation
B	Serial Fraction of Algorithm
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CAM	Computer Aided Manufacturing
CAx	CAD/CAE/CAM, etc.
IAB	Industrial Advisory Board
IGES	Initial Graphics Exchange Specification
ISO	International Organization for Standards
MES	Multi-Engineer Syncronous
MESH	Multi-Engineere Synchronous Heterogeneous
MMORPG	Massive Multiplayer Online Role Playing Game
n	Number of Processors
NIST	National Institute of Standards and Technology
NMC	Neutral Modeling Command
NPDB	Neutral Parametric Database
NSF	National Science Foundation
NXConnect	MUS NX Plugin Developed at Brigham Young University
OEM	Original Equipment Manufacturer
S	Processor Speed Up
SMCH	Solid Model Construction History
SQL	Structured Query Language
STEP	Standard for the Exchange of Product Model Data (ISO 10303)
UFO	Universal Features Object
UML	Unified Modeling Language
UPR	Universal Product Representation

CHAPTER 1. INTRODUCTION

The National Institute for Standards and Technology (NIST) performed a study in 1999 and made a conservative estimate that inadequate interoperability cost the automotive industry one billion dollars per year [1]. Interaction with Industry Advisory Board (IAB) members of the National Science Foundation (NSF) Center for e-Design has revealed similar waste in the U.S. aerospace industry. The cost of interoperability problems comes from the time and money spent to convert and rework data that was not converted correctly or suffered data loss. Iterating on a design is an essential aspect of engineering to produce an optimal design, so if the turnaround time on each iteration could be shortened, then time and money would be saved.

Engineering companies are sociotechnical systems in which engineers use an array of software tools and methods to execute standard product development processes. Their practices and purposes are to design new products in as short of a design cycle as possible while maintaining high product quality and meeting or exceeding customer demands. Speedup of a parallel computing system is based on number of processors and S(n), and is modeled by Amdahls law [2]

$$S(n) = \frac{1}{B + \frac{1}{n}(1-B)} \tag{1.1}$$

where n is the number of threads of execution and B is the fraction of the algorithm that is strictly serial. Generalizing speedup of any complex system, such as design, due to adding processors is bounded by what portion of the tasks can be run in parallel and what portion must be run in serial. In this dissertation we propose a new multi-engineer synchronous heterogeneous (MESH) CAD application architecture that will allow improved parallelization of collaborative design and therefore increase the overall speedup of the engineering sociotechnical system's design cycle.

Another way of thinking of the design process is as a divide-and-conquer algorithm where an increased number of looping operations means a reduced algorithmic speed. Within a typical product development process, where many activities are serialized by workflow programs and tracking, several sources of turnback/feedback loops occur when teams attempt parallel work. Historically, whenever these loops appear, supervisors, managers, and lead engineers rework and serialize the workflow so that no two individuals are using or touching the same set of data, thus greatly restricting collaboration and parallelization.

For example, large ship designs and construction are an extremely complex type of project undertaken by engineers. Clearly they represent the largest man-made mobile structures and are vitally important, carrying over 90% of global trade and moving massive numbers of passengers. Ships like the *Allure* or *Oasis of the Seas* exceed 350 meters in length, 60 meters in height and carry more than 6,000 passengers (see Figure 1.1a). Or, consider the *CSCL Globe* or *Barzan* at over 400 meters in length, capable of transporting 19,000 containers (see Figure 1.1b).

(a) *Oasis of the Seas*

(b) *CSCL Globe*

Figure 1.1: Ships Are Large, Complex Design Problems

The process of designing any ship begins as a huge systems design, layout, and analysis project to predict and determine the characteristics and performance of the ship prior to its construction. Owners want to know the costs to build, operate, and maintain [3]. Before computers, this was all done on very large design tables with the use of ship curves backed by a myriad of hull form calculations. Figure 1.2 illustrates a graphical representation of an initial layout of waterlines, butt lines, and station lines for a conceptual hull. Imagine doing this a dozen or more times before getting the system parameter correct in order to begin the phase of detail design. Before computers, large teams of ship designers worked in close collaboration with each other to ensure quick and seamless tweaking of the conceptual design. These teams consisted of designers and naval architects working side-by-sidesteam on J-size Mylar and/or velum sheets of paper.

Today, designers and naval architects are using Computer-Aided Design (CAD) software to conceptually design and layout. While these systems have added greater detail and accuracy,

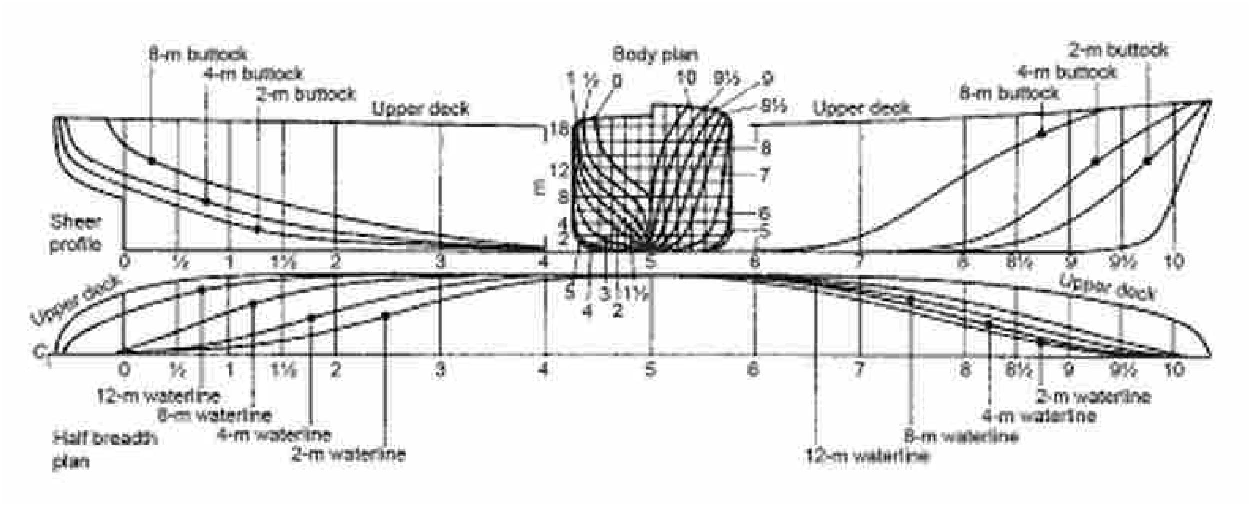


Figure 1.2: Initial Waterlines on a Conceptual Hull

they also limit access to the part, sub-assemblies, and master assembly. Only one person can have access to any one CAD file at a given moment in time. Also challenging is the fact that a 3D layout of a ship consists of millions of part files. The master ship assembly is a file consisting of location, orientation, and component information on all the millions of digital parts, which only one individual can checkout and manipulate. This is one of the major bottlenecks in todays CAD systems for naval architecture. These digital ships can be 350-400 meters in length but only one designer, architect, or engineer can be in any single part, sub-assembly, or master ship assembly. Pre-CAD naval design was much more a team effort on the largest most complex part and assembly models and drawings.

Please note that engineers who collaborate on projects must resolve design conflicts regularly, which causes costly design feedback loops. Next, take note that data must often be converted before it can be sent to suppliers. This creates another form of feedback loop; whenever there is data conversion, there is a chance for data corruption or data loss, which triggers a feedback loop. Finally, whenever supply chain members merge their work there is a chance for a design-conflict feedback loop. All these types of feedback loops are more frequent and costly the larger the project and the team. They are also more costly the later they occur within a project lifecycle.

In fact, surveys by NIST and feedback from industry members have shown the high costs of inadequate interoperability between CAD systems. As stated earlier regarding Amdahl's law, it will be necessary to improve the parallelization of design tasks in order to accelerate design processes and reduce non-recurring production costs. A flow chart that represents a typical design work flow is shown in Figure 1.3.

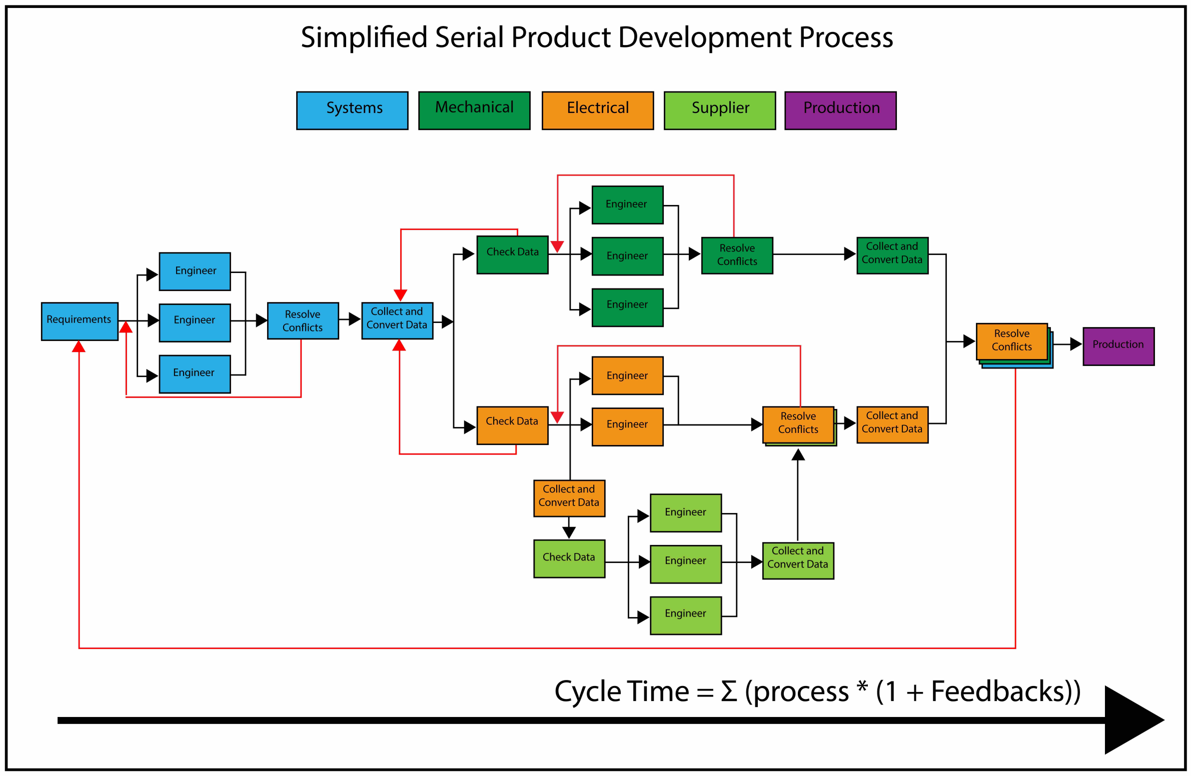


Figure 1.3: A Typical Product Development Process Has Several Sources of Feedback Loops

In this simplified product development process, there are three sources of feedback loops that prevent parallel workflows. First, when designers work in parallel without awareness of each others actions, design conflicts frequently occur and are more severe in proportion to the length of time the designers worked in isolation from each other. Second, whenever data is translated from one digital application to another (like one CAD system to another), data is often corrupted and engineering knowledge is lost, which leads to downstream confusion and thus feedback loops. This commonly occurs between OEMs and suppliers because both parties must translate between the different CAx software they use. Finally, there are cases where supply-chain members work designs in parallel with the OEM. When they synchronize their work they not only convert their data between CAD systems (the second cause of feedback loops), but they must also resolve any design conflicts caused from working in isolation from each other. The next few sections will discuss the causes and solutions of these feedback loops in more detail.

1.1 Synchronous vs Asynchronous Collaboration

The first source of collaboration feedback loops is internal design conflicts. These occur because of the inherently asynchronous nature of collaboration using modern CAD software. That is to say that engineers using CAD software may be working on the same product at the same time, but they cannot see what each other is doing and thus cannot work together effectively. On the other hand, individuals collaborating synchronously are aware of what each other is doing and can thus respond to each other in a social way. Social awareness is a critical concept for effective collaboration. Below in Figure 1.4 is an image of a common social situation. Presented with this situation, where would you sit at the table?

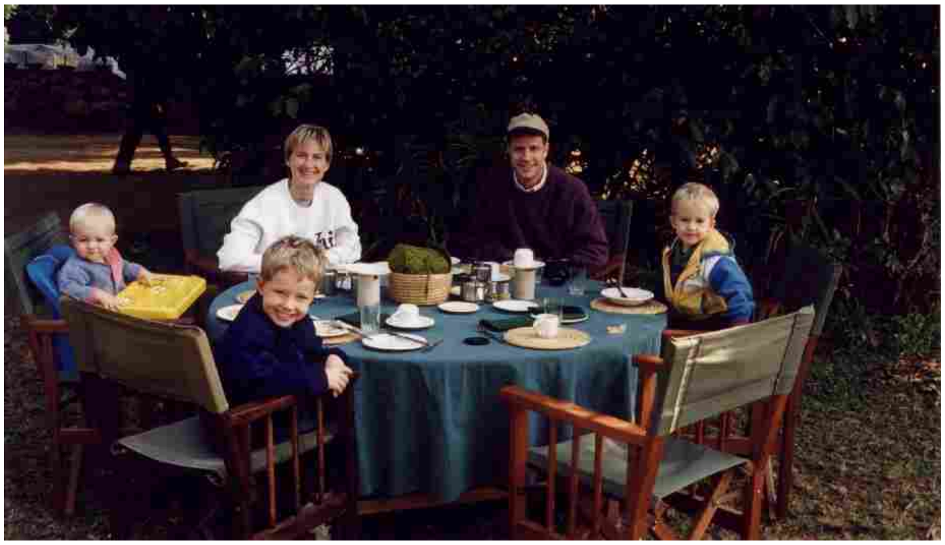


Figure 1.4: Social Situations Are Easy to Navigate Because We Are Aware of Each Other

The answer is obvious because you are socially aware; you can see that there is only one seat available. It would be ridiculous to expect each member of the dinner group to select a seat without knowing where anyone else had chosen to sit and expect them to sit without conflict, yet this is the way we ask our engineers to work in CAx environments. We hold weekly meetings to define rudimentary interfaces; following these meetings, each individual works on their own components without awareness of what others are doing. At the next review meeting, design conflicts are identified and resolution plans are determined and the cycle continues. In order to reduce these costly design cycles, we have introduced the concept of awareness into the CAD environment, changing it from an asynchronous to a synchronous collaborative environment. This is done through multi-engineer synchronous (MES) software and can reduce the flow chart shown in Figure 1.3 to that shown in Figure 1.5.

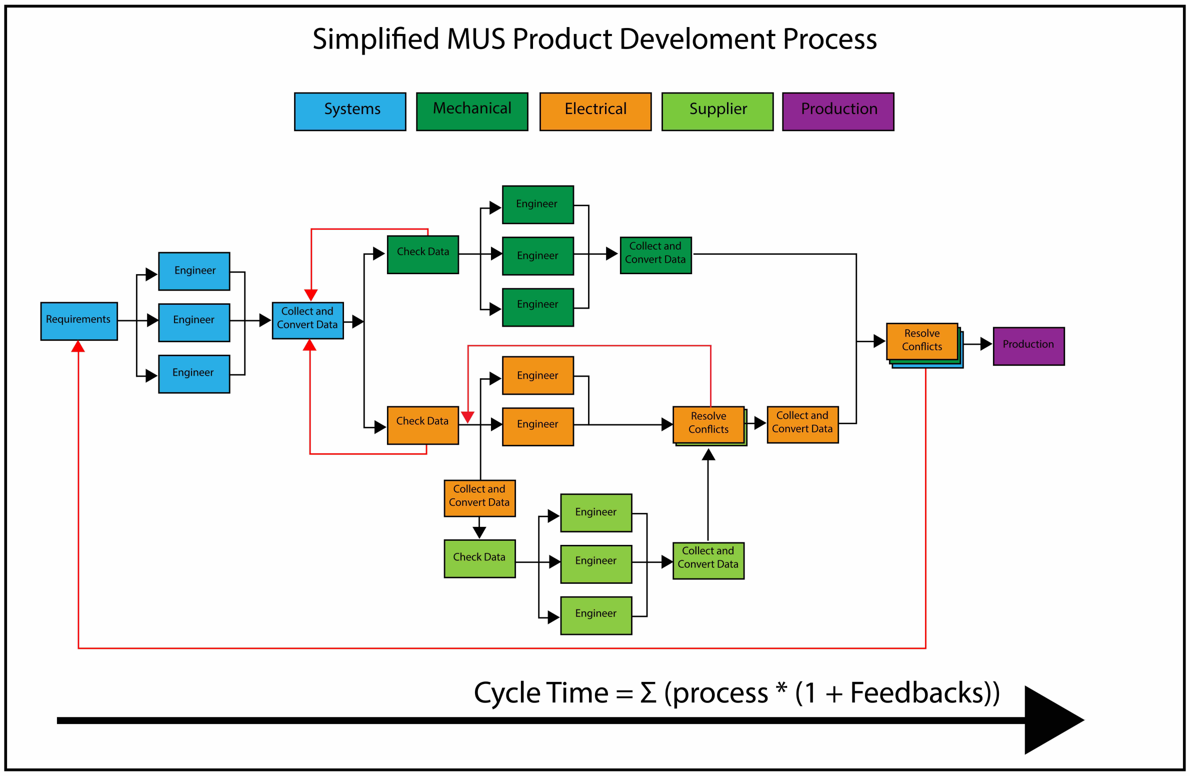


Figure 1.5: Multi-engineer Synchronous CAD Reduces Feedback Loops

1.2 Homogeneous vs Heterogeneous Collaboration

Collaborating across supply chains and disciplines simultaneously has the additional roadblock of working with heterogeneous data. That is to say that the data formats that different supply chain members use to store their CAD data or the data formats that varying engineering disciplines use to store their data are not the same and are not synchronously shareable. There are ways to translate or transfer data across supply chains and disciplines, but the translations are sequential and thus do not allow for synchronous collaboration. In order to bring the concept of awareness to supply chain and multi-disciplinary collaboration, it will be necessary to allow for collaboration on and synchronization of heterogeneous data.

The purpose of this dissertation is to create a system that can resolve the other types of feedback loops by creating a multi-engineer synchronous heterogeneous (MESH) CAD system, meaning a system that allows different CAD clients to interoperate seamlessly in real time. The result of such an appliction would be a design process with significantly reduced feedback loops as illustrated in Figure 1.6.

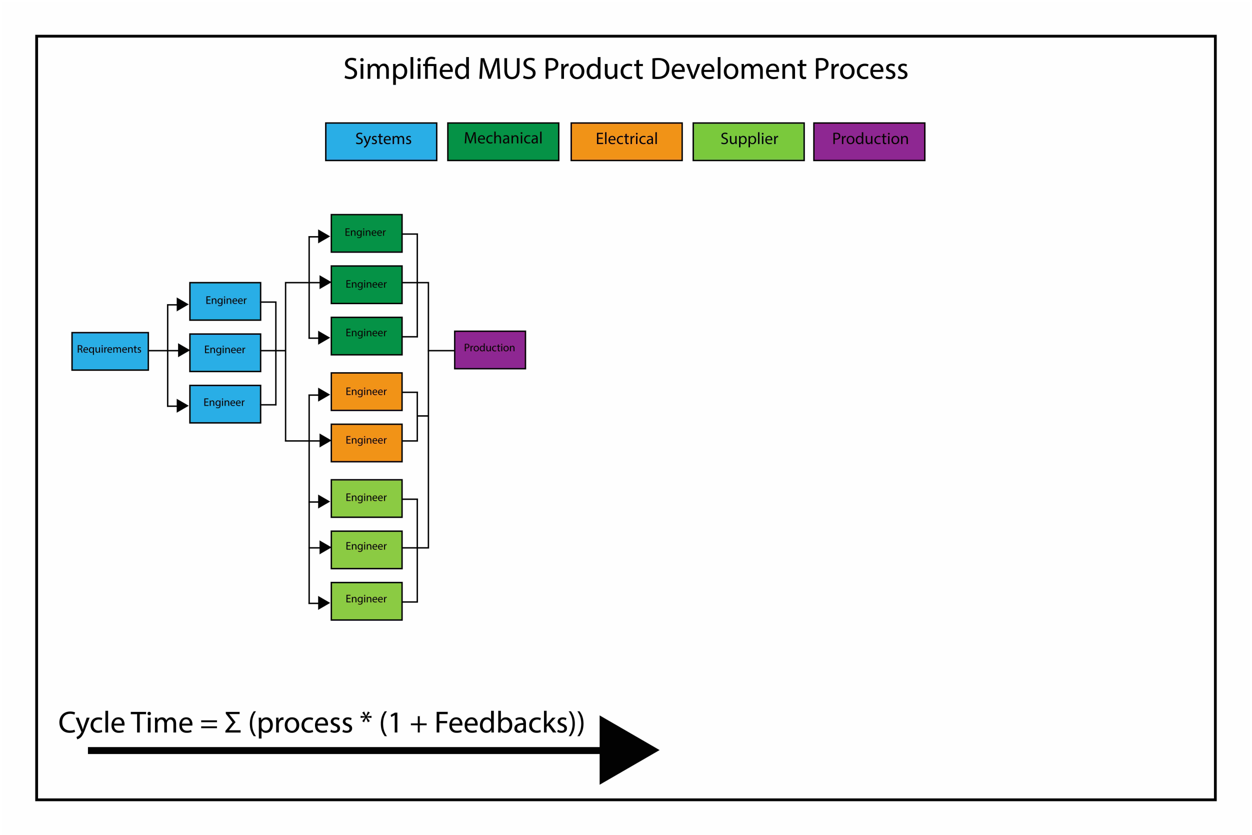

Simplified MUS Product Develoment Process
Systems
Mechanical
Electrical
Supplier
Production
Requirements
Engineer
Production
Cycle Time = Σ (process * (1 + Feedbacks))

Figure 1.6: Multi-engineer Synchronous Heterogeneous CAD Eliminates Many Feedback Loops

Jensen et al. [4] have addressed "Awareness," the first source of feedback loops through the use of CAD tools that enable synchronous collaboration by making engineers aware of what their teammates are doing. They have created a multi-engineer CAD tool, NXConnect, which is a multi-engineer plug-in to Siemens NX®. This tool eliminates the homogeneous CAD collaboration feedback loops by allowing engineers within a team to see, add, or manipulate each other's work. This turns severe design conflicts into micro-corrections that do not prohibit parallel collaboration. Chapter 2 of this dissertation provides a review of multi-engineer Awareness, CAD interoperable translation languages, and heterogeneous CAD. This background and foundational material allow one to judge the merits and contributions this work provides in the following areas:

1. Neutral Parametric Database Mathematical Model
2. Neutral Parametric Database Structured Query Language Model
3. Multi-Reference Interface Inheritance
4. MESH Server, Logic, and Client Layers
5. Large-scale MESH Considerations

The next four sections (1.5–1.8) give a brief introduction into each of these topics which are then addressed individually in their own chapters of this dissertation.

1.3 Neutral Parametric Database Mathematical Model

Modern commercial CAD systems represent their entities in a variety of different ways which greatly restricts interoperability from one system to the next. A canonical way of representing CAD entities is essential for building a heterogeneous multi-CAD collaborative design environment or multi-engineer simultaneous heterogeneous (MESH) system. Chapter 3 discusses the development and benefits of a Neutral Parametric Mathematical Model. The primary benefit of this model is that by using a core mathematical definition, the NPDB can be truly CAD-system independent.

1.4 Neutral Parametric Database Structured Query Language Model

Using a core mathematical definition for data storage can ensure that any CAD application can interoperate with the data in the NPDB, but there are other requirements for a MESH application to function. Chapter 4 discusses the benefits and process of implementing the NPDB in SQL. In short, the benefits are a high level of scalability and the prevention of data update anomalies regardless of the client or clients that are making use of the database.

1.5 Multi-Reference Interface Inheritance

One key challenge when creating a neutral parametric database is handling mutable references. For example, a sketch can be built off of a plan or a planar face. Chapter 5 describes how references of this type can be stored generically in the NPDB while avoiding update anomalies from heterogeneous clients.

1.6 Multi-engineer Synchronous Heterogeneous Server, Logic, and Client

In order to prove the concept of the NPDB, a fully functional MESH application was written. Technical challenges apart from the challenges in creating the NPDB are discussed in Chapter 6. In particular, the server must function independent of client and asynchronous data must be queried to form the CAD model tree graph in a stable way.

CHAPTER 2. BACKGROUND

Companies choose a CAD system based on many criteria, including cost, functionality, and ease of use. Each CAD system has a different feature set; that is to say that while many features overlap exactly between CAD systems, others have different but mathematically equivalent associative definitions and yet others have an equivalent mathematical boundary representation but are associatively incompatible.

In order for two CAD systems to truly interoperate, a engineer in each system should be able to take advantage of all tools that system offers. So, a neutral format should support the union of CAD features across an arbitrary set of CAD systems rather than their intersection. For example, in Siemens NX (NX), 2D sketch points can be constrained to the quadrant boundaries of a circle whereas in Dassault Systemes CATIA, they cannot. In order for engineers to collaborate across these two systems, engineers using NX should be able to use that type of constraint even though it is not supported in CATIA.

Section 2.1 will review the research that has been done in order to create a neutral CAD file format or a robust CAD translation scheme. Follwing, section 2.2 will discuss the research that has been done to develop MES CAD software. Finally, sections 2.3–2.7 outline the background of each of the core research topics will be outlined.

2.1 Current Neutral Formats

In 1979, roughly 300 major engineering firms met together and proposed the International Graphics Exchange Standard (IGES). The intent of this standard was to allow the commercial, proprietary, non-parametric CAD systems of that day to exchange wireframe and surface data (solid entities were not defined at this time) [5]. IGES was the first attempt at resolving data exchange challenges between CAD systems and is still widely used by many sectors industry. Its primary weakness is its sole focus on transferring geometric information. Additional problems are its limited geometric definitions and data corruption resulting from geometry tessellation or simplification. Holes, gaps, and slivers are common in the translated or simplified geometry, which requires significant resources to repair. IGES data corruption resulting from numerical approximations done during the outbound (CAD to IGES) and inbound (IGES to CAD) conversions must be repaired [6]. The NPDB aims to create a fully parametric CAD data standard that resolves these issues by not only representing resultant geometry but also storing the parametric modeling definition engineers need in order to edit CAD data.

To improve model data representation, the Standard for the Exchange of Product Model Data (STEP) was created in 1984 [7]. It has the advantages over IGES of storing product lifecycle information, separating the application specific data from the general shape data, and using a formal language to define the data structure, avoiding ambiguities when interpreting data [8]. As with IGES data corruption is a common problem. Unlike IGES, STEP is continuing to adapt and improve. The latest developments with STEP are Solid Model Construction History (SMCH), which is a hybrid approach [9]. B-rep and modeling construction history information are used to define features that provide both design intent and geometry information. A drawback for portability and network transfer of SMCH is that the file size can become extremely large due to B-rep and construction history [10]. In a multi-engineer environment that is constantly sending data back and forth between clients, a compact data format is necessary. Light-weight data transfer is one of the core features of the NPDB.

Another neutral format used in commercial translation software is the Universal Features Object (UFO) used by Iyer and Ganapathi in their research [11, 12]. The program created by Iyer and Ganapathi's research used the UFO to transfer binary files and translate them using the FeatureExchanger program. The FeatureExchanger is a collection of CAD application programing interface (API) calls and methods that has CAD to UFO and UFO to CAD translation processes. The UFO is a superset of all the CAD features that existed in the CAD industry at the time. The CAD translation company Aspire3D uses the UFO to perform their translations [13]. The FeatureExchanger software successfully translated a feature-based model in SolidWorks to a feature-based model in Mechanical Desktop. Ganapathi expanded the research to include CATIA V5 R8 and Unigraphics V 18.0 [12]. The limitation of the method is that the entire model is translated at once, rendering it incompatible with real-time collaborative heterogeneous CAD software.

The macro-parametric approach introduced by Guk-Heon Choi et al. has made strides in transferring features between heterogeneous CAD environments [10]. Choi came up with 167 standard modeling commands which were derived from the modeling commands of CAD systems. The research produced an external GUI where the translation of macro files between CAD systems could be handled. Duhwan Mun et al. continued the macro-parametric approach research by deciding upon a common set of modeling commands after surveying six different CAD systems [14]. A pre-processor was created that separates the source CAD system from the neutral modeling commands and a post-processor which separates the neutral modeling commands from the target CAD system. The pre- and post-processors map modeling commands between CAD system and neutral representation. This research was able to successfully translate simple parts from SolidWorks into CATIA with the feature trees in each being correct. The use of an external program, however, deters multi-engineer CAD. Standard modeling commands do not work with the actual feature data and therefore do not define a neutral format into which all CAD systems can translate.

Min Li also compiled a set of modeling commands, calling them neutral modeling commands (NMCs), and successfully translated data on a command-by-command basis [15–17]. Since the model is not translated all at once, engineers are able to collaborate more effectively by seeing updates as another engineer makes changes to the model. Each client has an add-on that performs the translation of modeling commands from the source CAD system into neutral modeling commands. The server relays messages between clients to keep each system synchronized. The command-by-command translation makes a multi-engineer experience possible but currently forces each engineer to take turns making edits. In other words, it operates like a WebEx session by passing control back and forth between engineers rather than allowing them to collaborate. The database of modeling commands in Li's research is a superset containing all modeling commands found in the CAD systems surveyed. Wanfeng Dou et al., created a similar program but instead of compiling the union of all modeling commands they created a minimum command set [18, 19]. A CAD Adaptor was attached to each CAD system that kept track of all the operations being performed in the CAD system and prepared them for transfer. A CoCAD middleware module converts the local operations from the CAD Adaptor to a neutral command and vice versa. Other contributors to the NMC approach are Song [20], Chen [21], and Zhang [22]. While these approaches almost achieve multi-engineer CAD by allowing multiple engineers to be in a session at the same time, they only allow one engineer to make edits at a time and so the environment does not truly support concurrent, synchronous collaboration. Finally, their focus was on the modeling commands themselves rather than a neutral data format with referential integrity.

Rappoport introduced a new neutral architecture called the Universal Product Representation (UPR), which provided universal support for all known data levels employed by the existing CAD systems [23]. It represented the union of data types supported by CAD systems, not just the intersection like other previous formats and methods. When the target CAD system doesn't support an incoming data type, a rewrite is performed that attempts to convert the data type into something usable by the target. The results of this research have been implemented in the commercial translation software Proficiency [24]. Feature-based CAD data translation is possible between five high-end CAD systems with high accuracy in translation. The drawback of this approach is that translation is tied to a binary file and occurs all at once. Bulk translation methods of this type do not lend themselves to synchronous applications due to their high computing overhead.

Using the neutral modeling commands of the macro-parametric approach, Tae-Sul Seo et al., presented an ontology approach [25]. The meaning, or semantics, associated with each modeling command was added to improve the interpreting of modeling commands based on what they were trying to create. A source ontology was created for each CAD system used along with a shared ontology that acts as the neutral representation. Axioms were used between source and shared ontologies to interpret the data. Sun [26] and Tessier [27] also used these methods and developed similar programs. This approach is potentially more flexible than previous approaches because the architecture can handle a wider variety of inputs. So far, the results of the approach have been the translation of very simple models. The disadvantage is still the same as with previous modeling command approaches because a neutral data representation for features is not defined and multi-engineer interaction is still limited.

Thus, while there is a significant body of research related to heterogeneous CAD translation, there is very little with respect to multi-engineer synchronous heterogeneous CAD clients. Existing applications that support real-time collaboration do so by requiring engineers to pass control back and forth so that while multiple engineers can observe at once, only one can make edits. While it is a large step forward from the isolated environment of single-engineer CAD, it is not synchronous collaboration. In order for synchronous collaboration to exist, the NPDB must support the qualities demonstrated by previous researchers while also resolving their weaknesses. In addition, the NPDB must take advantage of multi-engineer technology, which will be discussed in the next section.

2.2 Multi-Engineer Synchronous CAD

Although MES engineering software is a new development, video games have been "multi-player" since Tennis for Two in 1958. They have been network multiplayer in real-time since CAVE in 1975. Due to the early adoption of multi-player as an important concept in gaming, there has been significant development in asynchronous client-server architectures that support synchronous collaboration within the gaming industry, as well as user-interface technologies that aid in collaboration.

Three video game genres1 of interest that have applicable collaborative technologies are massive multiplayer online role-playing games (MMORPG), real-time strategy games, and sandbox games. MMORPGs involve tens of thousands of gamers playing on the same server in the same environment simultaneously. They typically control a single hero and work together in groups to fight adversaires, defeating enemies a single player never could; World of Warcraft is an example. Real-time strategy games are those in which small teams, with specialized personnel, build armies and control them in real-time against other teams; StarCraft is an example. The sandbox or open-world genre is one in which groups of people coexist in a virtual world where they can add and remove material and elements from the game in order to create whatever they wish; Minecraft is an example.

World of Warcraft has collaborative elements that allow very large groups of engineers to coexist in the same virtual world. A screenshot of the gameplay can be seen in Figure 2.1. Teammates can be identified by the green text above their heads. Each gamer can quickly see their teammates, what they are doing, and what their status is. This allows them to work together to accomplish tasks they could not efficiently complete alone.

StarCraft has elements that allow smaller teams to manage complex groups of virtual soldiers. A screenshot of the gameplay can be seen in Figure 2.2.

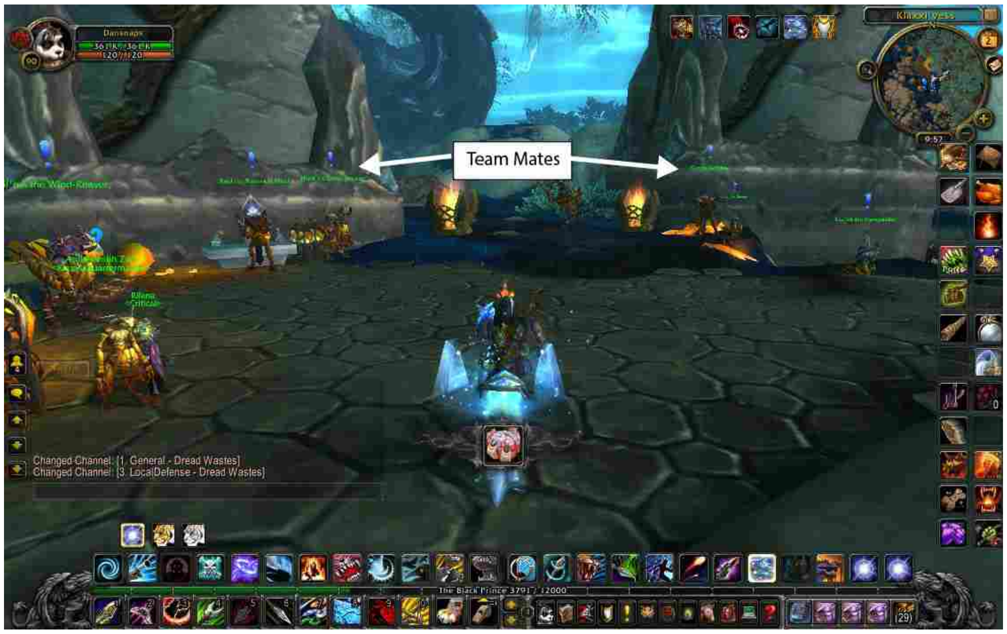

Team Mates
Changed Channel: [1. General - Dread Wastes]
Changed Channel: [3. LocalDefense - Dread Wastes]

Figure 2.1: World of Warcraft Gameplay

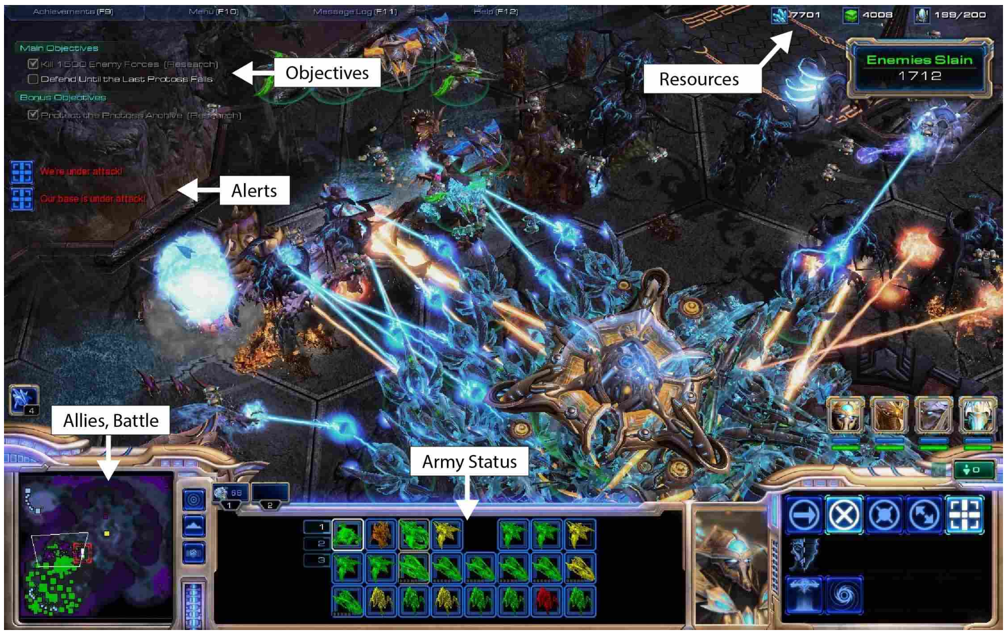
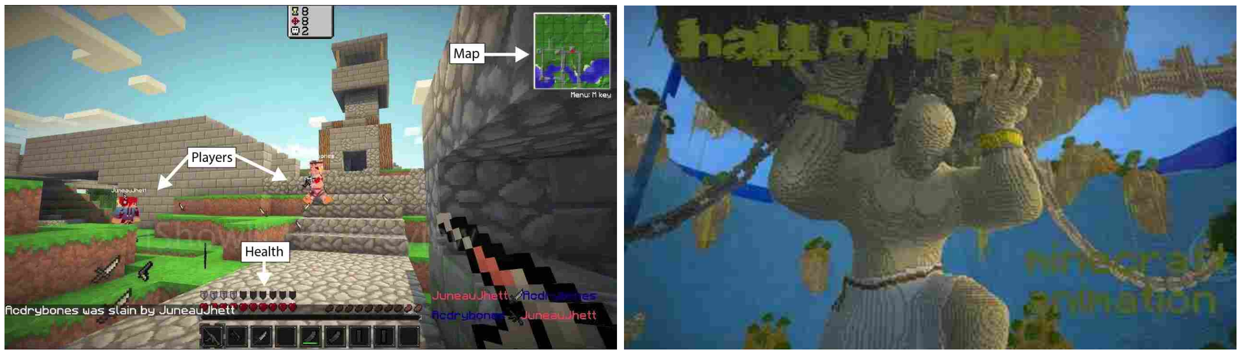


Figure 2.2: Starcraft Gameplay

Note that the engineer can quickly view the army status, troop by troop. They can see and adjust their resources, view the status of their allies, view the overall battle, accept and carry out objectives, and view alerts to any abnormal state.

Minecraft has features that aid groups in creating structures in collaboration. A screenshot of the gameplay can be seen in Figure 2.3a. Note that the engineer can quickly see other players, their health status, and an overall map of their world. The engineer interface and the building blocks of the game are simple, but the results can be surprisingly detailed, as shown in Figure 2.3b.



(a) Minecraft engineer View

(b) Complex Minecraft Design

Figure 2.3: Minecraft Gameplay

Video games have been developing synchronous collaborative technology for decades. Its no surprise that this generation of university engineering, computer science, and information technologies researchers want their engineering applications to work in the same way. Starting in 2008, the BYU CADLab (sometimes called the NSF Center for e-Design) researchers began studying how to transform single engineer commercial CAx applications into multi-engineer collaborative aware environments.

Jensen et al. have developed NXConnect which provides real-time multi-engineer modeling between multiple NX clients [4,28–32]. This technology is currently under commercialization. Cai created a multi-engineer SolidWorks experience using similar ideas to NXConnect [33]. Maher did similar research using AutoCAD [34]. While these systems support concurrent collaboration within a homogeneous CAD environment, they do not support a synchronous heterogeneous CAD design environment.

The ideal CAD environment would support concurrent collaboration with the union of CAD features across heterogeneous CAD environments. The purpose of this research is to define a neutral format that begins to merge the principles used to create a multi-engineer simultaneous CAD system with those used to create a heterogeneous CAD environment.

2.3 Architecture

It is important to use efficient architectural principles in order to create software of significant complexity . It is a widely accepted practice to use an N-tier architecture for applications that require data storage, logic, and presentation. Separating application functions into tiers lets developers focus on specific functionality within each tier. Having a few, well-made tiers might require more lines of code than writing a single procedural application, but each tier is much more clearly written and if the tiers are loosely coupled tiers can be replaced individually without disrupting the entire architecture. The tier structure used in BYU's homogeneous multi-engineer CAD tools, NXConnect, is shown in Figure 2.4.

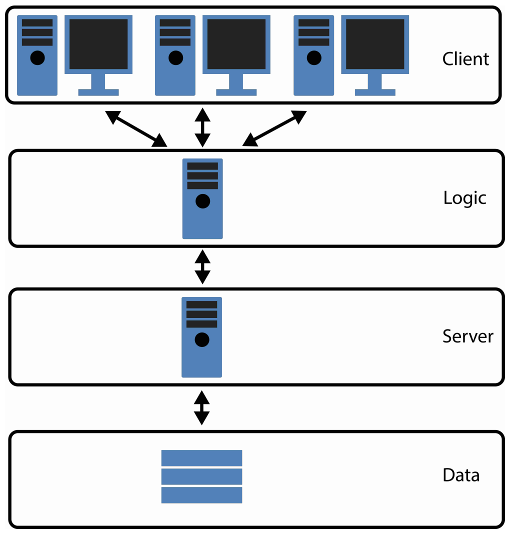


Figure 2.4: N-Tier Architecture

In the NXConnect application, client applications capture each action performed by the CAD engineers and then transmit those to a server. The server broadcasts those actions to the other clients and also saves them to a database. In this way, multiple engineers can all work on the same part at the same time.

The result is a truly collaborative CAD environment in which a team of operators are all aware of what their teammates are doing. They can respond to each others actions and resolve conflicts as they occur without causing costly feedback loops. For example, using multi-engineer CAD, the UAV shown in Figure 2.5 took a student 9 hours to model but can be modelled in only 1.5 hours by a team of six students.

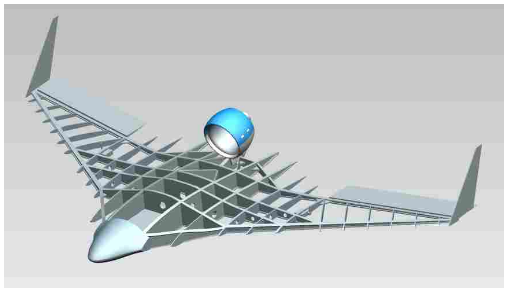


Figure 2.5: UAV Designed in 1.5 Hours Collaboratively Compared to 9 Hours Alone

This environment makes operators aware of each others actions and thus facilitates real-time collaboration. The same effect, but in a heterogeneous CAD environment, is the goal of this dissertation and so key changes and improvements must be made to the NXConnect architecture.

2.4 NPDB Mathematical Model

The most basic requirement of any CAD data format is that it represents the current state of the models under work. This is a trivial requirement for homogeneous CAD formats but not for heterogeneous CAD formats. Given a neutral data format, a CAD system should be able to read from and write to the neutral format and produce the same geometric result as another CAD system. An individual from one company or discipline cannot effectively collaborate with someone from another company or discipline using a different CAD system if they cannot both read and edit geometrically equivalent CAD models.

As stated earlier, there are many who have worked toward interoperability between heterogeneous CAD environments. The primary approach of these researchers has been to translate from one system to another using APIs like the macro-parametric approach developed by Choi [10]. The MESH architecture discussed in this dissertation takes a different approach. It develops a method to store parametric CAD data persistently in a database. This neutral parametric or NPDB is the foundation of the MESH CAD software developed in this dissertation. The contribution of this research is an architecture that combines the multi-engineer functionality developed by Jensen et al. with the heterogeneous technologies developed elsewhere. This new technology can resolve the third type of feedback loop where suppliers collaborate in parallel using heterogeneous CAD clients. The foundation of such an architecture is a data storage format that transfers full-fidelity models between heterogeneous CAD clients and can support asynchronous read/write operations from numerous clients. In order to transfer engineering knowledge across multiple heterogeneous clients, the data format should be neutral and parametric. In order to support real-world engineering projects the data should persist in a database, thus the Neutral Parametric Database (NPDB) format will be discussed in this dissertation and proposed as a new heterogeneous CAD standard.

2.5 NPDB Structured Query Language Model

Referential integrity in a CAD system provides consistency and stability in storing the data for the model. This is especially important in CAD translation software because the data is at the center of the process. Deficiencies in referential integrity are the source of data corruption when using existing neutral formats. The NPDB uses a relational database that links tables together using primary and foreign keys to preserve data consistency. This approach allows the model to be the same on all CAD systems. The language chosen to represent this relational database was the Structured Query Language (SQL). It is possible that Li used a similar database storage method but that is not clear from the literature [15–17]. If a SQL model was used then it was not the focus of the publications.

Multi-engineer CAD interaction allows multiple people who are geographically dispersed to create and edit a model collaboratively in real time. NXConnect, developed by BYU, successfully implemented a multi-engineer approach between NX clients [4]. This research has used the multi-engineer approach of NXConnect, which has a client-server architecture that can handle messages asynchronously while supporting modeling synchronously. Multi-engineer capability is the future of CAD modeling and greatly improves collaboration across the supply chain.

The BYU CADLab has been researching new tools, methods, and data structures to support the needs of its industrial members. The main body of this research has addressed the long lead times required to design complex mechanical components and assemblies by developing multi-engineer synchronous collaborative design tools. These tools reduce costs by reducing the time to design long lead time components and thus overall design cycle times. The NPDB defines the neutral data structure for many basic CAD features to enable translation between heterogeneous CAD environments. The architecture of the NPDB supports the multi-engineer CAD initiatives of the lab and is currently being developed to promote interoperability between NX, CATIA, and Creo CAD systems. The NPDB for a core set of features is defined in this dissertation and used to demonstrate multi-engineer data interoperability between Siemens NX and CATIA CAD systems.

2.6 Multi-Reference Interface Inheritance

Another important aspect of a CAD data format is that it support associative geometry. Associativity is the relationship between CAD features and is included in homogeneous CAD formats. For example, an associative line stores a pointer to 3D point features for its endpoints rather than the endpoint coordinates themselves. In this way, if one of the 3D points is edited, all associated geometry is automatically updated. This builds design intent into the models, which helps engineers collaborate with each other, and it facilitates editing the model without creating invalid geometry, which speeds up the design iteration process.

The approaches that have stemmed from Choi, Li, and Rappoport's work [10, 16, 17, 23] primarily focused on data translation rather than data neutralization and storage. Because of this, there has been very little discussion on how to store feature-to-feature references. Their publications either used text files, temporary classes, or did not outline their data storage methods. So, one important question that remains is how to store multi-reference types such as planar faces where more than one feature type could be referenced.

Most modern CAD packages store data in a proprietary format and only support interoperability through the translation of boundary representation (BREP) data through formats like IGES and STEP. BREP data is the resulting geometry of a CAD part rather than the editable, parametric definition. While this translation method has been extensively and successfully used in industry, the translation process does not result in associative geometry, limiting the applications of the transferred model. By only translating BREP data, design intent and the ability to edit are lost during translation. Any modifications or adjustments to the model must either be made on the originating system or redesigned in a new system that adds constraints and limitations to the design process. On top of these limitations, this method is inherently single-engineer. Before a design can be shared, it must be exported in a neutral format and sent to the end-engineer. Since only one engineer at a time can work on the part in a workflow like this, concurrent engineering is severely impeded.

A number of solutions have been proposed and developed to enhance the productivity and fidelity of design sharing while maintaining design intent. MES CAD implementations, which utilize a central server to route CAD commands to multiple clients, are able to accomplish the sharing of both high-fidelity models and design intent by sharing feature information. These systems allow multiple engineers dispersed both in time and space to view and actively contribute to a CAD model. While there are multiple implementations of this client-server architecture, those utilizing a thin-server, thick-client scheme are able to work with the commercial CAD systems used in industry. On this system, when a engineer performs an operation, data defining the operation is sent to the server and redistributed to all other clients. Each client machine then performs the command on the local system, generating the model. To support heterogeneous CAD environments simultaneously, CAD commands performed on a local machine must be translated to a neutral format before being transmitted to remote clients. The neutral format must contain all the data to perform the operation in any other supported CAD system. An example of this type of system is CAD Interop, a MESH CAD plug-in for Siemens NX, Dassault CATIA, and PTC Creo, which is currently being developed at the BYU CADLab.

Chapter 5 describes a method for storing the CAD neutral data in a database to ensure all associations between features are valid. This referential integrity limits the object associations to valid types only. The dialog window of a 2D sketch feature in NX 8.0 is shown in Figure 2.6. When open, the designer can choose to build the sketch on a plane feature, or on any planar surface described by BREP geometry. No other type of object can be selected. This method for referential integrity blocks corrupt, or invalid, data from being sent to other clients, or saved in the database, by ensuring that only these valid methods can be associated with the sketch.

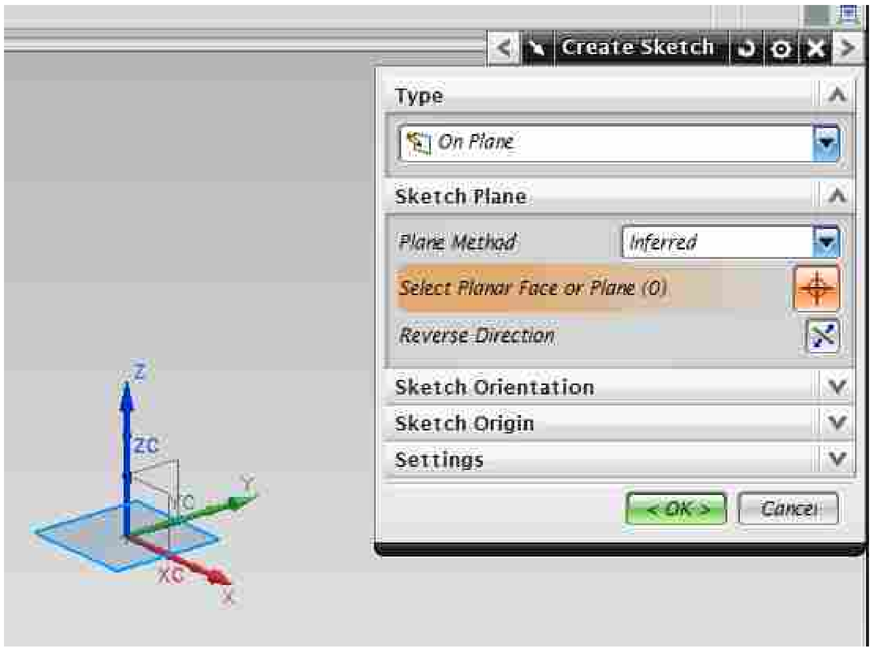


Figure 2.6: The Sketch Creation Dialog in Siemens NX 8.0 References either a Planar Face or a Plane.

2.7 MESH Server, Logic, and Client

Authors such as Choi, Li, and Rappoport [10, 16, 17, 23] typically outlined a translation method or one part of a complete system. This dissertation outlines an entire MESH architecture. While the NPDB is the foundation of this architecture, there are critical aspects of the rest of the application worthy of discussion. An N-tiered application can have N layers but the common tiers are a data storage layer that is fulfilled by the NPDB, a data translation layer that adds, reads, updates, and deletes data in the database from object-oriented classes, a communication layer that handles all communication between the multiple clients and the database, a logic layer that handles any computations necessary between the multiple clients and the database, and finally the individual client layers. In this case, both the data translation and communication tiers reside in a server application. The logic tier is referenced by both the server application and the individual client applications. The client tier resides in the client applications.

A logical architecture that supports both multi-engineer synchronous collaboration and interoperability is defined and tested for robustness and proposed as the start of a new standard for interoperability. In particular, a pseudo-singleton pattern is proposed to ensure data stability despite unordered data and a multi-engineer object class pattern is proposed to allow heterogeneous clients to interoperate even if the server has no knowledge of the client.

The singleton pattern is a design-pattern in software engineering where the instantiation of a class is limited to the creation of one object. This is done primarily to eliminate the use of global objects and variables but still allow the programmer to access static data. It also allows the user to implement interfaces which allows them to pass the singleton as an object into functions. This ability is the main difference that separates the singleton pattern from a static class. The pseudo-singleton pattern developed in this dissertation is an extension of the singleton pattern.

A MESH CAD client requires an asynchronous client-server architecture in order to allow simultaneous communication between CAD clients. The focus of this dissertation is an architecture with two core elements. First, it can communicate with the neutral parametric database (NPDB) format in a stable and efficient way. Second, it can work with unknown CAD clients in a standard way.

2.8 Background Summary

At the core of this dissertation is the idea that collaboration should be natural. We all collaborate with each other daily in many ways, but until now, CAx collaboration is unnatural. The research to date has not addressed this issue but continues to exacerbate it by ignoring the core mathematical foundation of engineering data, overlooking computer science best practices, and following oppressive practices from a socio-technical perspective. This dissertation takes a fresh look at these issues with a fundamental mathematical perspective, practical computer science approach, and modern social ideology.

CHAPTER 3. NEUTRAL PARAMETRIC DATABASE MATHEMATICAL FRAMEWORK

In order to affect MESH CAD an architecture similar to that used in NXConnect is required. That is to say that a data layer to store CAD geometry, a server layer to handle communication between clients and the server, a logic layer to translate between CAD client formats and data layer formats, and client layers for the users to interact with will be necessary. The first of these, the data layer, is the topic of this chapter. In order for MESH CAD to function this data layer must be neutral across the heterogeneous clients, it must store CAD data parametrically so that it is editable and it must be a database in the sense that it must store the CAD data persistently. These data storage needs give us the first requirement of this system, Requirement 1.

Requirement 1 *MESH CAD is only obtainable through the development, implementation, and acceptance of a Neutral Parametric Database (NPDB).*

The CAD translation methods discussed in the background have one common flaw: a dependence on CAD system APIs. CAD system APIs do not describe part geometry completely because they depend on the CAD system's kernal for geometric computation. Rather than a complete definition of the parts, they represent a set of parameters the CAD system needs to reproduce the geometry. This poses a serious flaw because a format that depends on today's CAD system APIs will not work if those APIs change or if one of those CAD systems ceases to exist. For this reason, the NPDB must contain a complete representation of any parts which leads us to Requirement 2.

Requirement 2 *The NPDB must be based on fundamental mathematical representations of CAD features.*

If the NPDB is based on a sound mathematical framework its data will be useful regardless of the CAD system used.

A common problem industry members have while working with CAD data is with data corruption. Corrupt CAD data is occasionally produced within single-user, homogeneous environments and is common when working with neutral data formats like IGES and STEP. In fact, the NIST survey cited earlier found that the majority of the billions spent due to inadequate interoperability was spent re-modeling parts that were insufficiently translated [1]. In order to allow any client written by any party to interact with a neutral data format it is critical that it prevent data corruption when data is written to the storage medium. In database terminology, any corruption of this type is called an "update anomoly" which leads us to Requirement 3.

Requirement 3 *The NPDB must prevent update anomalies.*

Another common mistake with existing neutral formats is that while they may store data well in theory, they are difficult to use in practice. A practical implementation of a MESH CAD system requires a data storage medium that is easy to work with. For this reason it should work well with the server, logic and client layers which will be written using object-oriented programming methods. This leads us to Requirement 4.

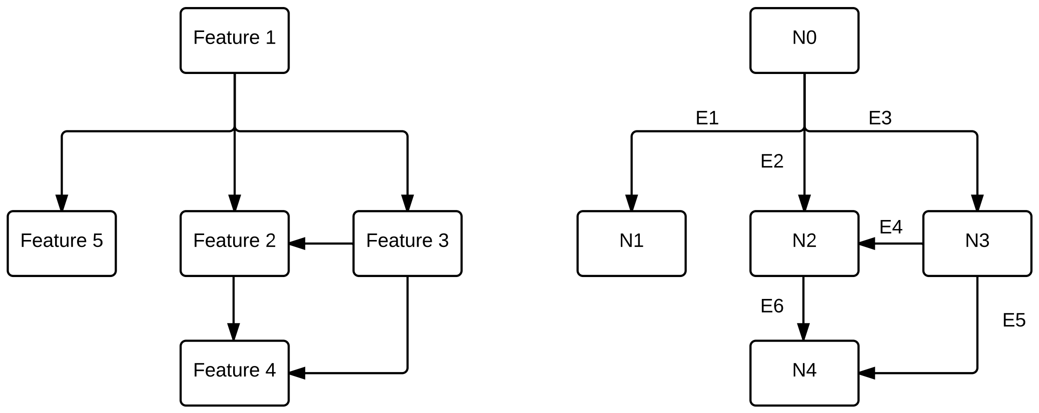
Requirement 4 *The NPDB must be compatible with an object-oriented class structure.*

Making the NPDB compatible with object-oriented classes will not only allow researchers to work with it efficiently but will help ensure timely commercialization of the technology.

3.1 Neutral Representation

The methods typically used to translate between CAD systems like Li and Mun's approach represent a part as a "part history" or list of commands that, if performed in order using CAD system APIs, would reproduce any given part. This is a natural approach to take when converting data, but is not sufficient for data storage for a few reasons. First, the set of all CAD systems with their respective APIs is unknown. Second, it does not show the relationships between features. Third, the list must be in the proper order and so has poor compatibility with asynchronous client-server architectures where messages may come out of order. Fourth, a wide variety of part histories could be derived from the same part. Finally, it will be shown that the API method does not correctly translate complex curves and surfaces like NURBS because a part history is dependant on the CAD system to produce the geometry rather than storing its complete representation. The approach taken in this research is to represent the geometry completely using a mathematical framework which includes both graphical representations and geometric definitions (canonical form) in order to meet Requirement 2. In this way, even if two CAD systems store geometry using different parameters, an affine transformation can be performed between the two.

Rather than a sequential part history, a part can be represented as a directed acyclic graph (DAG) from graph theory as shown in 3.1.



(a) Part Graph with Feature Labels (b) Part Graph with Node and Edge Labels

Figure 3.1: Graphical Representation of a CAD Part

A graph is an ordered pair of a set N of nodes and a set E of edges. In the image above the nodes are boxes which represent each feature within the part and the edges are arrows which represent the relationships between the features. Additionally there are variables like lengths and booleans which can modify any given node N. So, a part P can be represented as shown in equation 3.1.

$$P = (N, E) \tag{3.1}$$

In the sample part graph above, the set of nodes comprises each feature as shown in equation 3.2 and the set of edges comprises each inter-feature relationship as shown in equations 3.3–3.9.

$$N = \{N0, N1, N2, N3, N4\} \tag{3.2}$$

$$E = \{E1, E2, E3, E4, E5, E6\} \tag{3.3}$$

$$E1 \rightarrow [N0, N1] \tag{3.4}$$

$$E2 \rightarrow [N0, N2] \tag{3.5}$$

$$E3 \rightarrow [N0, N3] \tag{3.6}$$

$$E4 \rightarrow [N3, N2] \tag{3.7}$$

$$E5 \rightarrow [N3, N4] \tag{3.8}$$

$$E6 \rightarrow [N2, N4] \tag{3.9}$$

The graph edges are directed because references between features are one-way parent-child relationships in nature. Parts are acyclic graphs because cyclical references cannot be resolved. For example, if a part had 3 points that were built from each other in a cycle trying to determine the location of those points would result in an infinite loop, preventing the resolution of the geometry. Each different type of CAD feature represents a distinct node type and has a set of variables v which act on the node and set of references r which denote an edge relationship to a parent node. These can be written as demonstrated in equation 3.10.

$$N \rightarrow [\{v_1, v_2, v_3, ..., v_n\}; \{r_1, r_2, r_3, ..., r_n\}] \tag{3.10}$$

Each node type must be determined individually based on the feature's canonical form. The following sections will illustrate the process for identifying each feature's parameters and whether they are variables or references. The sketch point, sketch line, sketch arc, and sketch spline features are used as examples and demonstrate how Requirement 2 is met.

3.1.1 Sketch Point

A sketch is simply a collection of geometric elements that lie on a plane. The sketch node itself can be thought of as a horizontal and vertical coordinate reference frame used to position children and a plane whose vector equation is shown in 3.11.

$$n \cdot (\vec{r} - \vec{r_0}) = 0 \tag{3.11}$$

Where n is the plane normal vector, r is a known point on the plane and r_0 is any other point that satisfies this equation. In order to efficiently store a sketch S only an origin point reference O, a horizontal axis reference H, and a vertical axis reference V need be stored as demonstrated in 3.12 because r can be any point along either the horizontal or vertical axis, r_0 can be the origin, and n is the cross product of the horizontal and vertical axes.

$$S \rightarrow [\varnothing; \{O, H, V\}] \tag{3.12}$$

A sketch point P can be created in two distinct, affine ways. Its coordinates can be stored in either the part's global coordinate system as shown in 3.13 or in the sketch's local coordinate system as shown in 3.14, in which the point's Z' coordinate is always zero and need not be stored.

$$P \rightarrow [\{X, Y, Z\}; S] \tag{3.13}$$

$$P \rightarrow [\{X', Y'\}; S] \tag{3.14}$$

It is more efficient to store the point in the sketch's local coordinate system because one data member fewer need be stored and so a sketch point is represented in the NPDB by a reference S, a variable X', and a variable Y'.

3.1.2 Sketch Line

A line L in a CAD sketch is typically a line segment and can be modeled parametrically by 3.15,

$$L = P_i(1-t) + P_j t \tag{3.15}$$

where P_i is one sketch point, P_j is another, and t is a parameter that varies from 0 to 1. So, a sketch line can be stored by a reference to its parent sketch, a reference to its start point, and a reference to its end point as shown in 3.16.

$$L \rightarrow [\varnothing; \{S, P_i, P_j\}] \tag{3.16}$$

For many sketch features, the API function calls match well with the parametric definition of the geometry. In this case, one can simply look at the API methods from the CAD systems and determine a storage format from any of them as long as an affine transformation exists. This is straightforward in the case of the sketch line because the creation method of a line is equivalent in all CAD systems we have seen.

3.1.3 Sketch Arc

The base of a sketch arc is a sketch circle. A circle C has the parametric equation shown in 3.17, where r is the circle radius and t is a parameter that varies from 1 to 2π. It is stored as shown in 3.18, in which S is a reference to the parent sketch, P is a reference to the center point, and the variable r is the circle radius.

$$\begin{aligned} x &= rcos(t) \\ y &= rsin(t) \end{aligned} \tag{3.17}$$

$$C \rightarrow [r; \{S, P\}] \tag{3.18}$$

The parametric definition of an arc is the same as that of a circle except the range of the parameter t is from the start angle to the end angle of the arc. Some CAD systems work directly with this definition in their API and the storage method for an arc A would be as shown in 3.19, where θ_i and θ_j are the start and end angles, respectively.

$$A \rightarrow [\{r, \theta_i, \theta_j\}; \{S, P\}] \tag{3.19}$$

This method has the advantage that it is very close to the geometric definition of an arc. However, it has the disadvantage of low connectivity with surrounding geometry. If the arc and another sketch element both terminate at a sketch point, it is easier to handle sketch changes if the arc is directly associated to the point rather than simply an angle. For this reason, other CAD systems store sketches as shown in 3.20,

$$A \rightarrow [r; \{S, P_c, P_i, P_j\}] \tag{3.20}$$

where P_c is the arc center point, P_i is the arc start point, and P_j is the arc end point. The start angle can be found as follows in 3.21,

$$\theta_i = acos\left(\frac{H \cdot (P_i - P_c)}{\|H\| \|P_i - P_c\|}\right) \tag{3.21}$$

where H is the horizontal vector of the parent sketch. The end angle can be found in similar fashion. The endpoint coordinates can be found from start and end angles, but it is difficult to connect multiple geometric elements in such a fashion so the points are stored as references in the NPDB rather than the start and end angles.

3.1.4 Sketch Spline

Sketch splines are an element for which the inputs obtained from API methods form an incomplete parametric definition. If the API method inputs are used to create a database storage method, the inputs for a spline, Sp, in most systems would be as shown in 3.22,

$$Sp \rightarrow [\{t_0, t_1, ..., t_n\}; \{S, P_0, P_1, ..., P_n\}] \tag{3.22}$$

where the spline interpolates the points P and there is an optional tension t at each point. This storage method has very little to do with the actual mathematics of the curve. Any point on a degree d B-spline can be found using 3.23,

$$P(t) = \sum_{i=1}^{n} P_i B_i^d(t) \tag{3.23}$$

where P_i is the ith control point and B_i^d is the B-spline basis function for the ith control point. It is important to distinguish between the B-spline through points and the B-spline control points. A B-spline typically does not pass through its control points. For a more detailed treatment of B-splines and polynomial interpolation the reader is referred to Sederberg [35].

To make editing easier for end engineers, most CAD systems store points and then use some interpolation scheme to create a B-spline through those points. As a result, different CAD systems produce varying geometric results from the same set of control points. The full solution to this problem is outside the scope of this dissertation, but in short, the geometric definition of the splines could be stored as a variable in the graph and the parametric definition could be stored by references to the control points, allowing either CAD system to edit the same parametric definition while viewing an accurate geometric representation.

3.2 Asynchronous Communication Support

Storing features by their core definition ensures interoperability across heterogeneous CAD systems. The other core attribute the NPDB should have is multi-engineer simultaneous support. There are a multitude of different data storage solutions including plain text files, XML files, binary files, relational databases, object oriented databases, and NoSQL databases. A few criteria were considered when choosing a storage medium. The medium must support a high volume of simultaneous reads and writes, ensure data integrity, map to programmatic classes, and be in common use among developers. The different storage media are compared in 3.1.
Based on these criteria, a relational database was chosen. Relational databases exist in a server environment where they can support a very high volume of reads and writes, they were specifically designed to ensure data integrity, there are object relational mappers in wide use, and they are very common. In fact, most PLM solutions use a relational database to store their data for these

Table 3.1: Comparison of Data Storage Methods

Method Name	Simultaneous	Data Integrity	Class Mapping	Common Use
Text File				✓
XML File			✓	✓
Binary File				✓
Relational Database	✓	✓	✓	✓
Object Oriented Database	✓	✓	✓	
NoSQL Database	✓		✓	✓

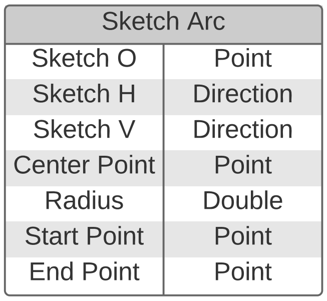
very reasons. This database should be normalized in order to prevent data corruption and it should support an object-oriented class-structure mapping.

3.2.1 Database Normalization

To have a relational database that meets requirement 3 by avoiding modification anomalies, the database must be normalized. While there are numerous normal forms that can be met, it is a generally accepted practice to meet only the first three normal forms, which are primarily focused on allowing for efficient data queries and data integrity. Later normal forms are more aimed at reducing data redundancy but come at the cost of data usability.

The first normal form of database design designates that the database should (1) eliminate repeating groups in individual tables, (2) create a separate table for each set of related data, and (3) identify each set of related data with a primary key. Taking the sketch and its elements as an example, some might store a data for each sketch arc as shown in the database diagram in Figure 3.2.

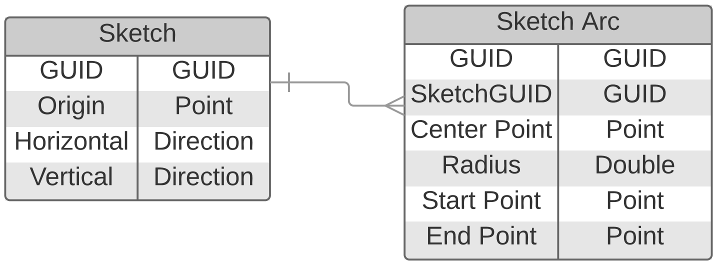
In this type of a table structure, the data for the parent sketch would be repeated in each sketch feature. In addition, it would be difficult to query all children of a particular sketch. In order to meet the first normal form, the sketch and sketch arc elements are separated into two separate tables and each is given a primary key. A primary key is a unique identifier for each entry in each table and is used to link tables together. While a primary key could be any unique attribute of a data item, it is a generally accepted practice to use a surrogate key for each table in order to ensure efficient queries. In most databases this would be a simple integer value, but in the case of



Sketch Arc	
Sketch O	Point
Sketch H	Direction
Sketch V	Direction
Center Point	Point
Radius	Double
Start Point	Point
End Point	Point

Figure 3.2: Sketch Arc with Redundant Data

a client-server architecture where the client should set the key rather than the database, a globally unique identifier (GUID) is a superior choice. Data that meets the first normal form is shown in Figure 3.3. Please note that the connections show a one-to-many cardinality where one sketch can have many sketch arcs.



Sketch	
GUID	GUID
Origin	Point
Horizontal	Direction
Vertical	Direction

Sketch Arc	
GUID	GUID
SketchGUID	GUID
Center Point	Point
Radius	Double
Start Point	Point
End Point	Point

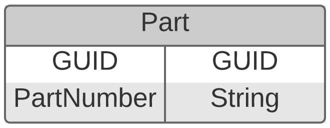
Figure 3.3: Sketch and Sketch Arc that Meet First Normal Form

The second and third normal forms respectively state that no non-prime attribute is dependent on any proper subset of any candidate key of the table and that all attributes in the table are determined only by the candidate keys of that table and not by any non-prime attributes. In other words, each table should distinctly represent one logical item and no data in each table should relate to anything except that specific logical item. These are both satisfied by the separation of sketch and sketch arc shown above. The normalization of the database prevents update anomalies and so meets Requirement 3.

3.2.2 Object-Oriented Mapping

As mentioned above in Requirement 4, another important consideration when creating an efficient storage data structure is to support hierarchical class inheritance while working with the data. The applications should be object-oriented programs that take full advantage of the benefits of class inheritance. In relational databases the two most common ways to support a class hierarchy in mapped objects are a table-per-hierarchy (TPH) and a table-per-type (TPT) method. Generally speaking, a TPH can support faster queries but allows many nullable fields, which can lead to data corruption. A TPT hierarchy, on the other hand, maintains tighter referential integrity at the cost of query speed. In the case of a neutral format like this, data integrity was placed at a higher priority than query speed and thus a TPT mapping method has been used. In order to illustrate this method we must first add a few more properties to the database that are conceptually basic but necessary for the application to function and helpful in illustrating the value of a TPT architecture.

First, more than one part will be stored in the database so a part type should be added to the database. The part type is defined as shown in Figure 3.4.



Part	
GUID	GUID
PartNumber	String

Figure 3.4: Part Database Diagram

Every feature should reference its part so that it can be easily queried. Additionally, every item in the database should have a time stamp when it was added to the database (for convenience), as well as a Boolean that marks whether or not the feature has been deleted. Items should not be truly deleted from the database except in extreme circumstances. This gives us the database structure shown in Figure 3.5.

Not only will these tables be difficult to manage because time-stamp and deletion code will have to be written for each table, but they no longer obey the third normal form because there is data in each table that does not pertain to only that table. In order to fix this, we break out all repetitive properties into their own tables. In addition, it is helpful to add a standard prefix to each

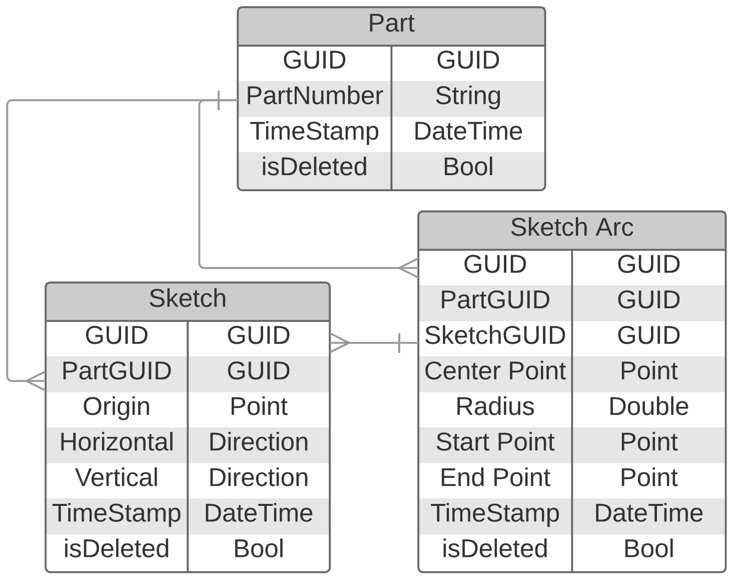


Figure 3.5: Part, Sketch, and Sketch Arc without Object Oriented Hierarchy

table and remove all white spaces from the table names in order to avoid class name conflicts and problems later on in development. The resulting hierarchy is shown in Figure 3.6 where an open arrow designates inheritance.

Please note that all tables must still have a GUID primary key in order to satisfy the first normal form, but that the GUID of a child table has a foreign key constraint to its parent table in order to maintain an inheritance relationship. By creating a table-per-type hierarchy, the database meets Requirement 4.

3.3 Remarks on NPDB Mathematical Framework

So, this chapter has described a neutral database format that is mathematically based (Requirement 2) prevents update anomalies (Requirement 3) and is compatible with an object-oriented class structure (Requirement 4). It is the NPDB and the foundation of a MESH CAD environment (Requirement 1). The first four requirements to create a MESH CAD environment are at a high, theoretical level. In order to create a practical system, the NPDB must be implemented in a language that supports these requirements. The next step is to implement a database that meets these requirements.

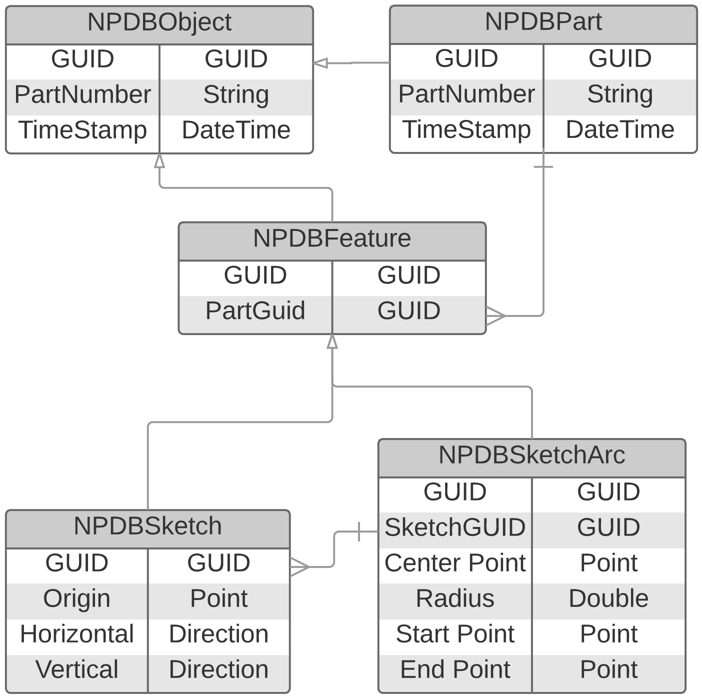


Figure 3.6: Part, Sketch, and Sketch Arc with Object Oriented Hierarchy

CHAPTER 4. NEUTRAL PARAMETRIC DATABASE STRUCTURED QUERY LANGUAGE MODEL

In the previous chapter, a mathematical framework was laid for the NPDB standard which can support a MESH envrionment. In this chapter a practical foundation is laid for this format. This chapter will discuss the requirements to create an initial implementation that proves the NPDB concept and is later extensible to a large-scale enterprise environment through commercialization.

A common pitfall in application development is professionally known as “scope creep”. Scope creep is the condition when the requirements for an application change continuously over time and prevent the completion of that application. In order to avoid this a clear scope must be defined which achieves the core goals of the application while meeting a limited scope. The snare in this situation is to attempt the impossible task of defining an application which solves all problems in the first try rather than simply taking important steps forward from the current state. This problem is so common it has its own fallacy, the nirvana fallacy, which is characterized by comparing actual things with unrealistic, idealized alternatives. This leads to Requirement 5.

Requirement 5 *The nirvana fallacy must be avoided by proving the concept with a partial feature set.*

Rather than try to develop the ideal CAD solution, this dissertation aims to prove the concept of a framework that solves a few key CAD interoperability issues using a small subset of CAD features.

Another common mistake related to the nirvana fallacy is the inability to choose a programming tool set because there is no perfect tool set. Developers and engineers often try to write their own tool sets rather than use existing ones. It is often said in industry that “Everyone wants to follow standards as long as it’s their standard”. This is even true for the developers of the STEP standard who, rather than use the IDEF1X data modeling standard that had already existed for decades or use the mature unified modeling language (UML) diagrams common to computer science professionals, invented their own data modeling language and diagrams called EXPRESS. It is important to use existing tools that are common to industry if there is to be an open dialogue with industry members which leads us to Requirement 6.

Requirement 6 *The NPDB standard must be implemented using industry best practice tools and methods.*

While it is important to use industry-standard tools and methods in order to garner the respect of industry members, is also necessary to create an application that has a clear application to their environment. Creating an application that can be scaled up to their environment is not only important in order to hold their interest, but it is also important to truly prove the possibility of a MESH CAD software system. If the application is oversimplified and if non-standard, custom tools are used then the application will not be a true test of the technology. This leads us to Requirement 7.

Requirement 7 *MESH software must be scalable to an enterprise-scale design environment*

Even though it is not necessary to create an enterprise-level application to prove MESH CAD feasibility, it is necessary to use tools and write the application in such a way that there is a clear path to commercialization.

4.1 Methods

The methods used to create the Neutral Parametric Database and test if it better solves the interoperability problem will be explained in this section. The NPDB was defined for a set of commonly used graphical commands and elements called features. The initial feature set was chosen based on engineer experience and time constraints. Journaling, where a engineer's modeling commands are recorded and saved as a text file, was used to determine the important methods needed in creating the features in each CAD system. The required data to define a feature was found by viewing the inputs to the methods. By comparing the data needed in each CAD system a neutral representation became apparent and could be defined. Multi-User Object (MUObject) classes which contain all the needed logic to translate a feature between the CAD systems supported were used to store the neutral and CAD specific data for each feature. A client-server architecture connects all the participating clients to the server which relays messages between systems and uses the MUObject classes to translate the data. By using these methods the neutral format for a feature was defined and its validity was tested within a multi-engineer CAD setting.

4.1.1 Determining a Feature Set

The ultimate goal of the interoperability project is to support the union of all CAD features. In order to avoid the nirvana fallacy and meet Requirement 5, the goal of this particular research was to support enough basic features that moderately complex parts could be modeled. An initial feature set was constructed by using the design experience of those familiar with CAD systems. The scope of this dissertation was to define a neutral format for basic sketch and solid modeling features and test their effectiveness.

An aerospace senior design project where complex CAD models of aerospace parts were created in NXConnect was analyzed as a starting point in determining the most common features used during typical modeling. NXConnect is a multi-engineer version of NX developed by BYU. The database of features created in the NXConnect software during modeling was queried to find the features used. Based on these findings and the CAD experience of others working on the project the current feature set was chosen. The features that were minimally used during the project were not listed because they weren't required for basic modeling. Also, individual sketch features were not able to be queried in the database so the sketch feature set had to be determined by experience. Many of the features listed are advanced features that can be replicated using other, simpler features and were not implemented to save time. The mirror feature, for instance, could be replicated by modeling both sides of a symmetrical model instead of just one. Sew, trimmed sheet, and thicken sheet are surface modeling features and will be implemented in later releases of the interoperability software. Any features that were auto-generated by NX during modeling were also not included. After considering the above points, Table 4.1 provides the initial feature set chosen for the CAD interoperability software. This basic set of features makes the modeling of complex parts possible. This dissertation will only discuss the 3D point, 2D point, 2D line, and 2D spline features.

By instrumenting an existing project for the features to be implemented, a reduced feature set was identified which could sufficiently prove the concept of MESH CAD in a reasonable amount of time, thus avoiding the Nirvana Fallacy and meeting Requirement 5.

Table 4.1: Interoperability Program Initial Feature Set

Sketch Features	3D Modeling Features
Point2D	Datum CSYS
Line2D	Datum Plane
Arc2D	Point
Circle2D	Line
Spline2D	Spline
	Extrude
	Revolve
	Sketch

4.1.2 Defining a Neutral Format

A great starting point to understanding the data requirements for CAD features is to record a journal or a script of the feature being used. Li and Mun used this idea to determine a set of neutral modeling commands that could be used as an intermediary between CAD specific commands [14] [15]. Most major CAD systems have journaling capability but a good understanding of the CAD system's API can allow a developer to program without it. NX and CATIA both provide journaling capabilities where the engineer starts recording, uses the feature within the program, and then stops recording. A text or script file is created which provides the actual functions called to create the feature. The inputs to the functions used are often the data needed to define the feature. This approach sheds light on the data required to define a neutral format and was done before implementing the feature in the interoperability program.

4.1.3 MUObject Structure

The architecture of the program consists in multi-engineer object classes (MUObject) which store the neutral and CAD specific versions of each feature. This is necessary to support the creation, editing, and deletion of features. These classes are written in the C# language and extract the feature data using the CAD systems' API's. Within the multi-engineer class there are methods that translate back and forth between server features and CAD specific features. These methods are called when messages are received by the server that changes have been made by a engineer. Each feature has its own multi-engineer class which handles translations on a feature by feature basis.

These approaches makes multi-engineer CAD interaction possible which greatly reduces the time to market for products. Efficiency is increased because the process becomes more parallel when collaborating through multi-engineer software.

4.1.4 Client-Server Architecture

Multi-engineer CAD requires a messaging system that can handle messages asynchronously while supporting modeling synchronously. A thick client thin server architecture achieves this by storing messages in a queue and then executing them in between the creation of other features. The interoperability program features a plug-in that is installed on each client which can communicate with a server running on the same client or elsewhere. A class for each CAD system is in charge of sending and receiving messages from the server via the MUObject classes and calls the appropriate feature class to create the needed feature on the client. The versions of each of the features are stored in MUObject classes on each client as discussed earlier. The server communicates with the database and stores the neutral data of the feature. Since translations are performed on a feature by feature basis, multi-engineer modeling is supported.

4.1.5 NPDB Verification

Creating, editing, and deleting of features in both CAD systems were tested to verify the NPDB approach. After all features passed the initial stages of testing, the modeling of a moderately complex part was tested. When features are used in combination with each other and dependencies are generated the code is required to be more robust to work properly. This last test verified that the code is stable so focus could be turned to adding more features and functionality to the program. The features tested in this research were the 3D point, 2D point, 2D line, and 2D spline features.

4.2 Implementation

Work has been done on the NPDB which defines the neutral data structure for many basic CAD features to enable translation between heterogeneous CAD systems. The architecture of the program consists of database, communication, logic, and client tiers. In order to meet Requirements 6 and 7 a standard programming tool set was chosen which is commonly used in industry.

In the database tier, data is stored in a structured query language (SQL) database which has a table for each feature type supported in the program. Microsoft SQL server was chosen because it is well integrated with the Microsoft .NET toolset but other relational databases could be used. This database is mapped to classes using an object-relational mapper (ORM). Microsoft Entity Framework was chosen as the ORM because once again, it is integrated with the .NET toolset, but another commercial ORM could be used or a custom ORM could be developed. The ORM is used so that when changes are made to the database, the communication tier handles messages to and from the server, client, and database. The logic tier contains C# classes with methods for creating and updating neutral features from CAD system features and vice versa but other languages such as C++ could be used. It is worth noting that other tool sets such as those associated with the c++ and custom ORMs offer performance improvements at significant developmental cost. In the opinion of the author these should be avoided for student projects but may be the right choice for a commercial application. The client tier tracks a engineers actions and tells the logic tier what features need translation. The overall program architecture is illustrated in Figure 4.1.

The architecture of the NPDB supports the multi-engineer CAD initiatives of the lab and is being developed to promote interoperability between NX, CATIA, and Creo CAD systems. This dissertation focuses on interoperability between NX and CATIA only. The 3D point, 2D point, 2D line, and 2D spline features have been implemented and will be discussed in this section. They have all been implemented using standard industry tools and have been described using standard UML diagrams in order to meet Requirement 6. These industry standard tools are meant to support an n-tiered application architecture which is specifically meant for enterprise-scale application development and so this tool set also meets Requirement 7.

4.2.1 3D Point Implementation

The first 3D modeling feature implemented was the 3D point feature because it is relatively easy to understand. The specific data for the NPDB of a 3D Point feature was chosen to be an X, Y, and Z coordinate. While the definition is simple, extracting the data from each CAD system and translating between the two requires more effort. NX and CATIA have their own unique methods and approaches to creating 3D points in their respective CAD systems. For example, to get the point coordinates from a CATIA point the GetCoordinates() method needs an empty array passed

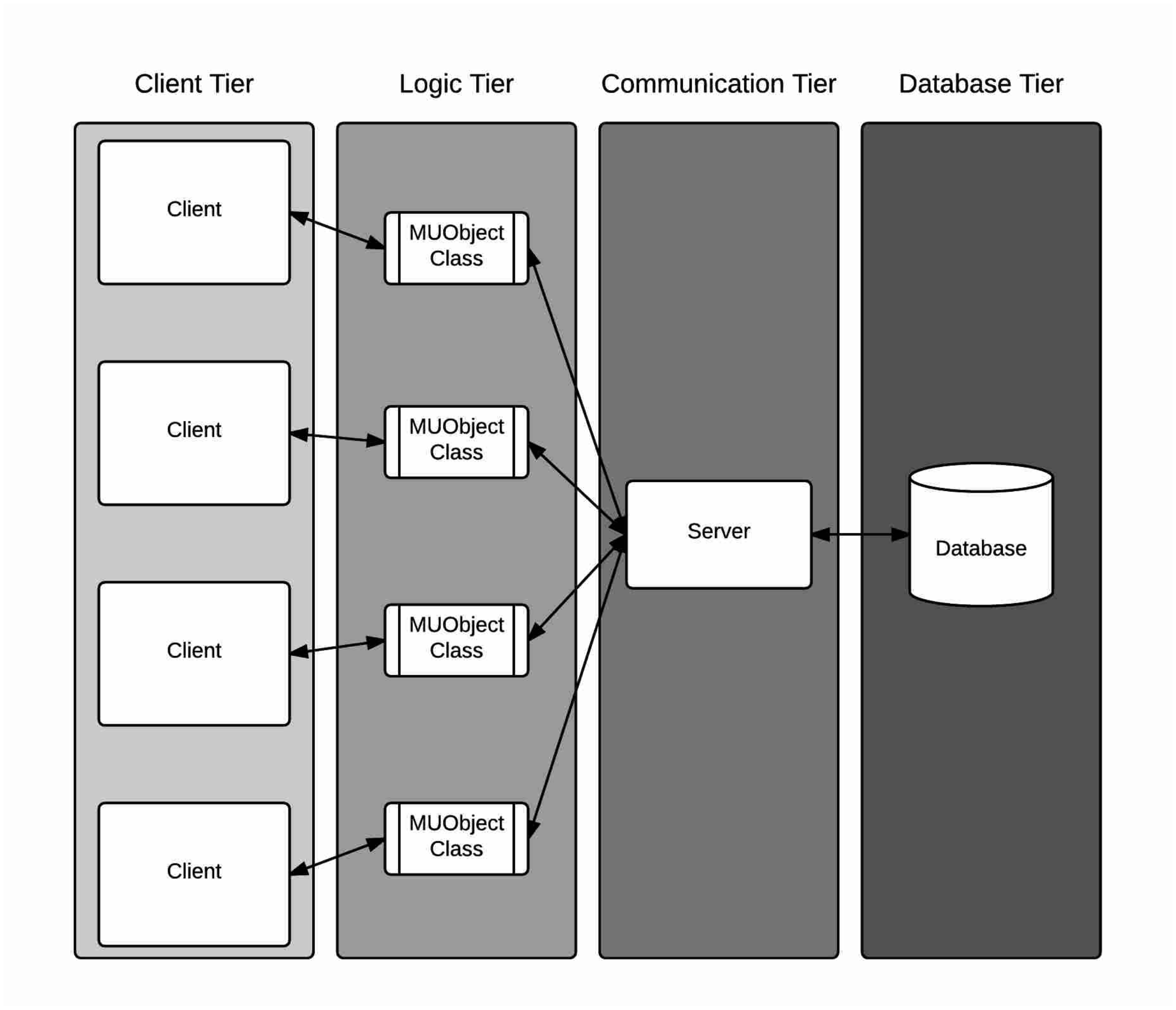


Figure 4.1: Interoperability Program Architecture

in which can be filled with the coordinate values. An NX point uses a builder which links the point created by the CreatePoint() method to a PointFeature object. Even though the methods between CAD systems differ the data they are manipulating is the same so a neutral representation can be defined.

The 3D point object is connected directly to the Feature object as seen in Figure 4.2. Outlined arrows indicate inheritance relationships and a short line crossing a main line on one end with three lines spreading out on the other end represents a one-to-many relationship. The GUID for the point is inherited from the Interop Object and other important properties are inherited from the Feature object. GUID stands for Global Unique IDentifier and ensures that each point created can be uniquely chosen from a list of others. The 3D point feature is the simplest feature that can be implemented.

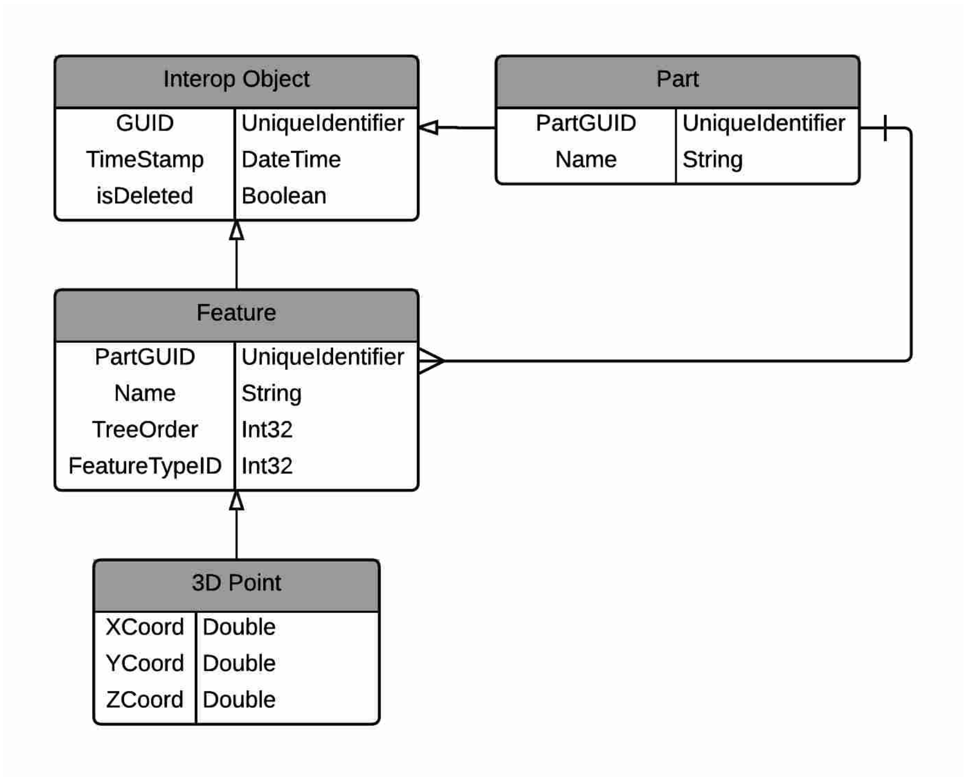


Figure 4.2: NPDB For 3D Point Feature

The NPDB for a feature encompasses more than just the basic data required to define it. It also includes the feature's relationships with other base objects in the architecture. All features implemented inherit from the Feature object so the top three boxes in the figure are the foundation of the NPDB. To extract and define the data that goes into a 3D point, the MUPoint3D class is used. This class contains the methods which assign the properties.

4.2.2 2D Point Implementation

A 2D point is the most basic sketch element that can be created. It is an important feature because many other sketch features are based on it, such as 2D lines. Even though a 2D point is the same as a 3D point constrained to a specific plane, the addition of a sketch greatly complicates the translation process. Most of these complications, however, are managed by the MESketch class created by others in this research group and only the MUPoint2D class will be explained. The unique data in the NPDB for a 2D point is an x and y coordinate. The GUID which uniquely identifies the 2D point from other 2D points is again inherited from the Interop Object through the Feature and Sketch Feature objects. The x and y coordinates position the point on the sketch plane and the SketchGUID inherited from the Sketch Feature object links it to a specific sketch. The Sketch Feature object and the Sketch object both inherit from the Feature object and the Feature and Part objects inherit from the Interop Object. The Feature object is linked to the Part object through a one-to-many relationship which means that a part can have many features. Figure 4.3 illustrates the 2D point NPDB.

4.2.3 2D Line Implementation

The 2D line feature adds some complexity to translation because a 2D line depends on two 2D points. The NPDB for the 2D line feature is a start point GUID and an end point GUID. The 2D line feature inherits a sketch GUID from the Sketch Feature object just like a 2D point. Just as a Sketch Feature object's SketchGUID property links the feature to the proper sketch, the start and end point GUID properties link the line to the proper 2D points in the database. Figure 4.4 shows the added complexity of the NPDB for the 2D line feature. Notice that the foundation of the NPDB is the same but there are some added relationships and data for the 2D line.

Two one-to-many relationships attach two 2D point objects to the 2D line object. The MULine2D class extracts the data for the 2D line NPDB and differs from the MUPoint2D class in the inputs required for the methods. Instead of just needing the feature and the sketch to be passed in, a 2D line also needs the start and end points passed in. This is because when a line is created or edited the start and end points need to be created or edited first. The MULine2D class is able to call the MUPoint2D class and use it to create the points. The same thing is done with the sketch. Separating the code into feature based classes is central to the software's ability to perform translation on a feature by feature basis.

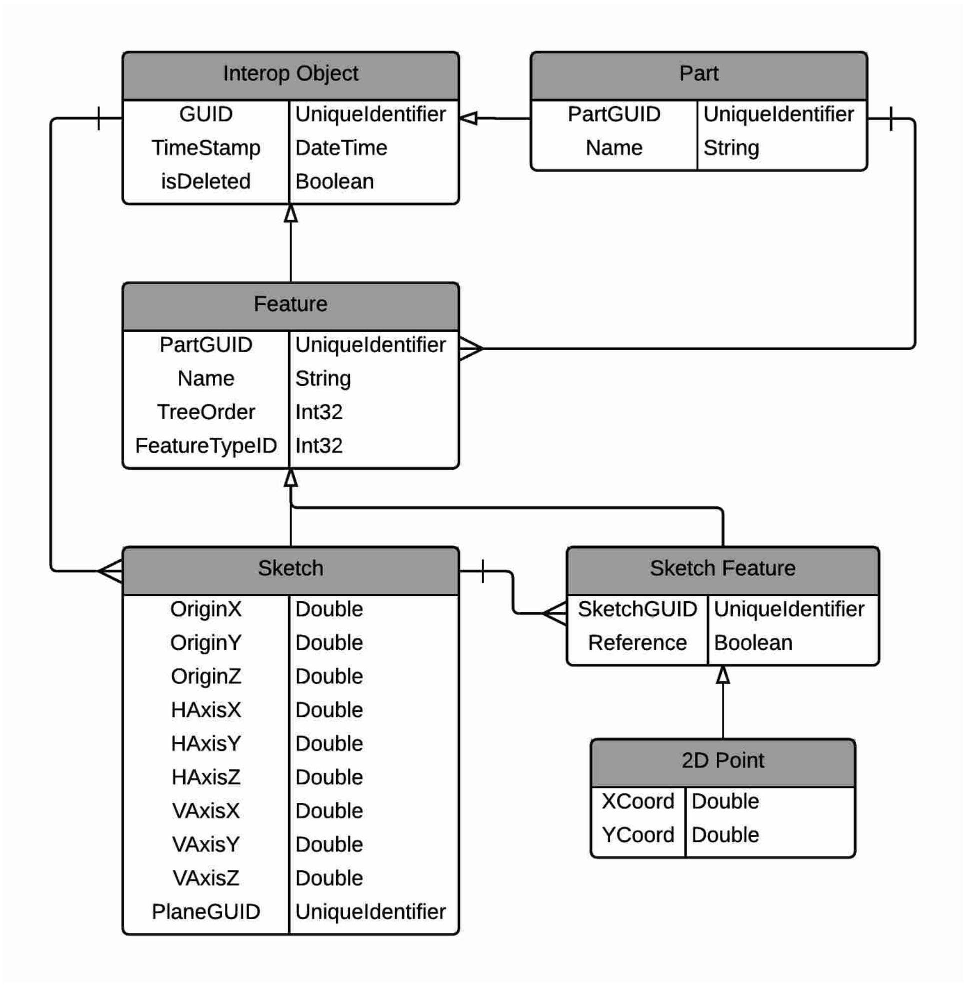


Figure 4.3: NPDB for 2D Point Feature

4.2.4 2D Spline Implementation

A 2D spline allows complex surfaces and shapes to be generated through extrusion or revolution. The NPDB consists of an array of control points and is shown in Figure 4.5. Other important properties such as the GUID and sketch are inherited as was the case for the other sketch features.

Implementing the 2D spline feature was similar to what was done for the 2D line feature but the addition of control points complicated the translation. Instead of passing in the start and end points to the class methods the control points array is passed in.

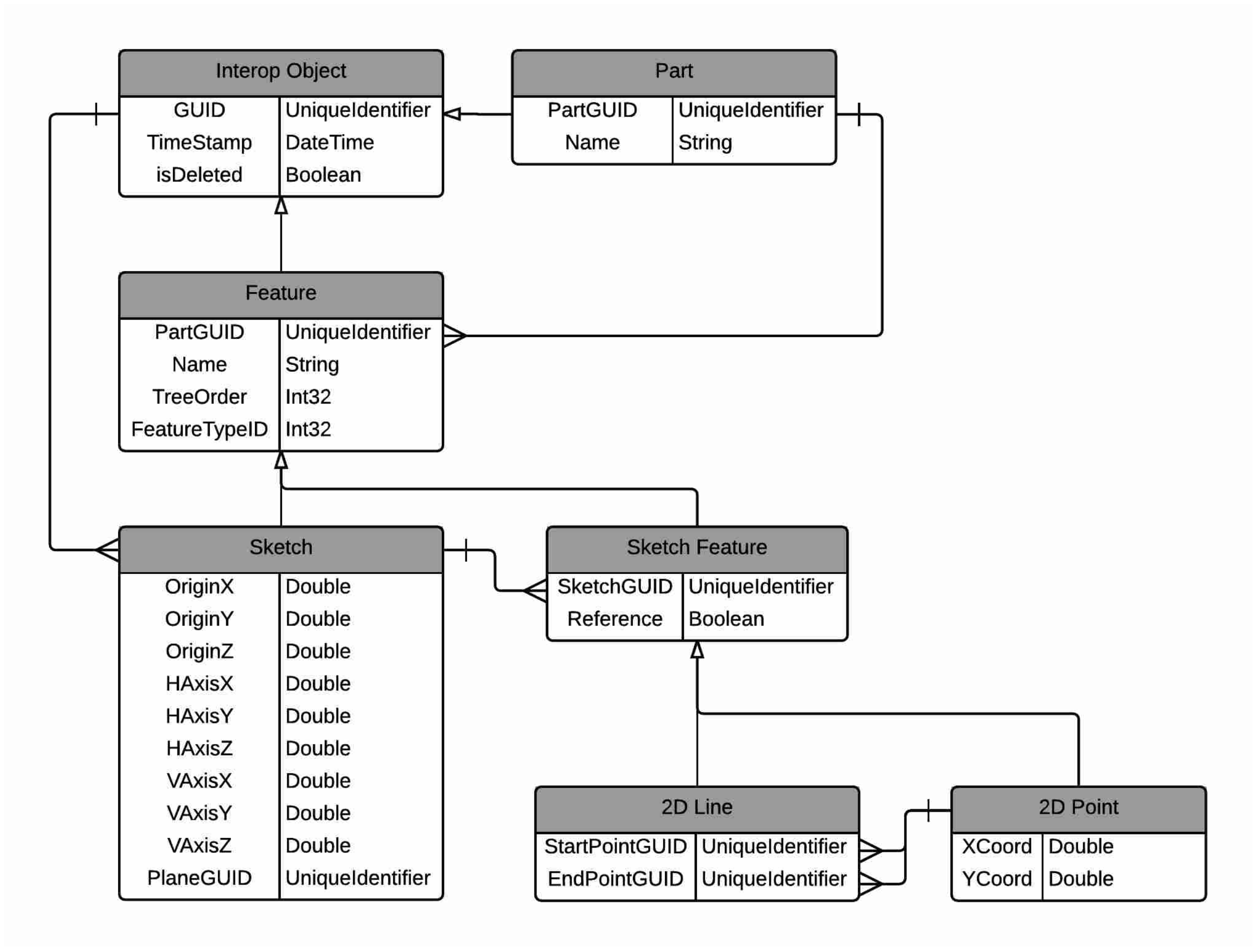


Figure 4.4: NPDB for 2D Line Feature

2D Control Point Implementation

A 2D control point is its own sketch element and a separate multi-engineer class was written for it. The extra data for the NPDB of a control point is a 2D spline GUID and an order number. Figure 4.5 shows how the NPDB for a 2D control point feature relates to the 2D spline feature NPDB. The relationship between a 2D spline and a 2D control point is a one-to-many relationship because one 2D spline has multiple 2D control points. The 2D control point inherits its X and Y coordinates from a 2D point object as shown in the figure. The use of an unspecified number of control points makes implementing the 2D spline more complicated. When translating, the number of control points for the specific 2D spline needs to be extracted using the CAD API's to properly

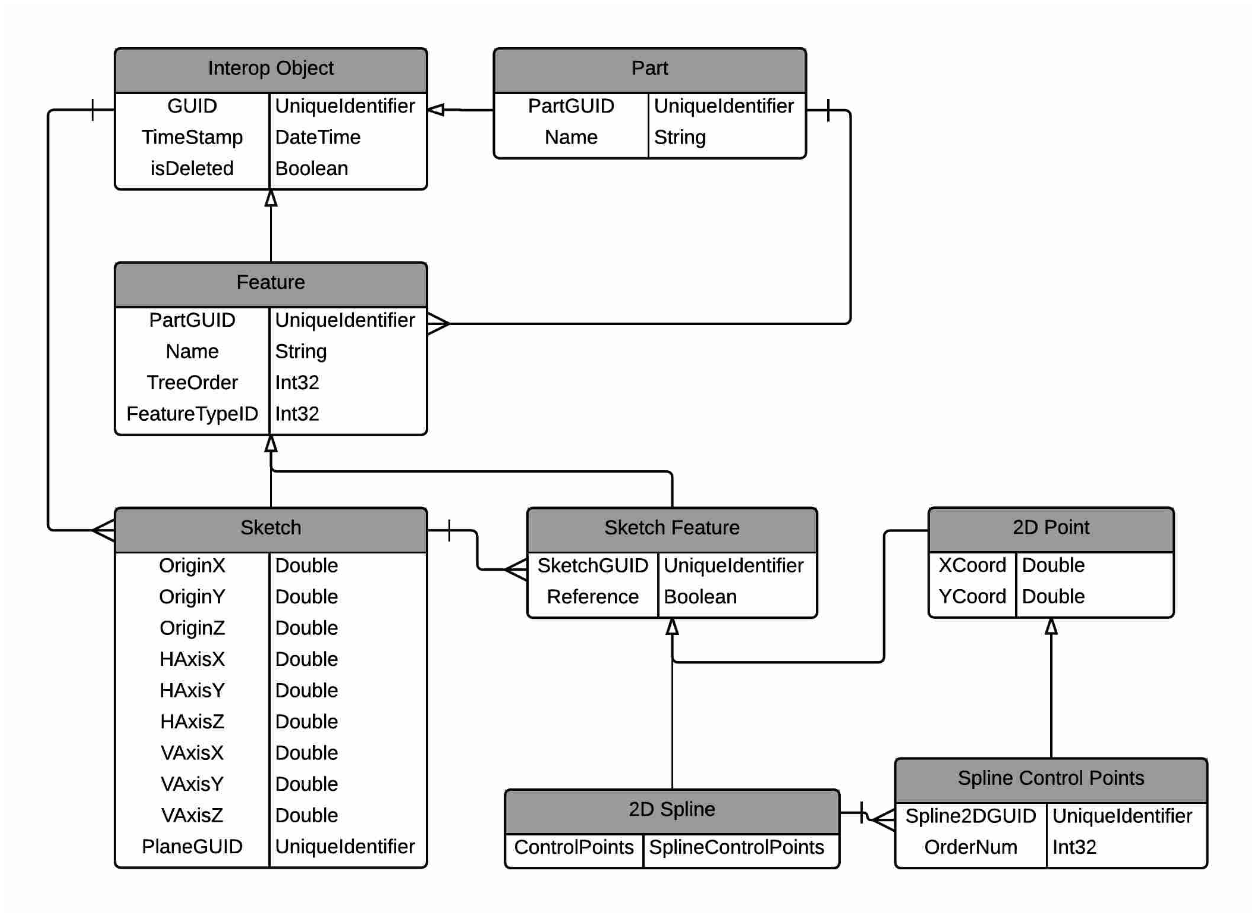


Figure 4.5: NPDB for 2D Spline Feature

fill the array of control points. The order in which the control points are added to the 2D spline is also very important which is why an order number is assigned to each control point.

4.2.5 Feature Capabilities and Limitations

All the features discussed can be created, edited, and deleted in either CAD system during modeling. Multiple engineers using either NX or CATIA can collaborate on models without restrictions. While this research has focused on a limited set of the features, it is not difficult to add more to the database. In fact, the 2D circle, 2D arc, extrude, revolve, datum CSYS, datum plane, 3D line, and 3D spline have all been added since the experimentation described in this dissertation.

Current limitations of the program include, no undo/redo functionality, and no conflict resolution logic. Unexpected bugs also occur during modeling and more work is needed to make the program more stable and robust. While there is edit functionality the undo/redo options are currently not implemented. Finally, if two people attempt to edit the same feature at the same time there is no logic in place to lock a feature being edited by another engineer. This can cause inconsistencies in the model. Ammon Hepworth and Robert Moncur implemented conflict resolution and feature locking in NXConnect and similar ideas could be applied to the interoperability software [30] [36] In spite of these limitations the existing software is effective.

4.2.6 Remarks on NPDB Structured Query Language Model

By implementing a core feature set rather than a complete feature set (Requirement 5) and using industry standard tools (Requirement 6) an architecture has been developed which not only proves the concept of MESH CAD but is also extensible into an fully scaled enterprise environment (Requirement 7). While this and the previous chapter have outlined a theoretical and practical foundation for the NPDB, there is one open issue with the format. The references between features in a parametric model can be difficult to model while avoiding update anomalies. The discussion up to this point has been limited to references that point to one type of CAD feature. This focus has been maintained in order to develop the critical mathematical representations of features including the tree graph that represents their most fundamental dependencies. The next chapter will define a method to deal with the more complex case of references which can be directed to multiple feature types.

CHAPTER 5. MULTI-REFERENCE INTERFACE INHERITANCE

The previous chapters have laid out a theoretical and practical foundation for the NPDB but one that only supports simple inter-feature references. This chapter discusses what is required to support the more common, multi-feature type references which model associative relationships between features. Associativity in CAD systems is one of the most powerful ways for a designer to express design intent. By building a 2D-sketch feature on a datum plane, the designer signifies to others that the sketch, and any child features of the sketch, are associated with the location of the plane. Any neutral format used to translate CAD features from one CAD system to another should retain these associations to preserve the design intent of the designer. This leads to Requirement 8.

Requirement 8 *The NPDB must store inter-feature references in a way that mirrors existing commercial CAD systems.*

That is to say that the NPDB must be able to reproduce any type of reference that can be found in an associative CAD part.

Also due to this highly-relational nature of modern feature-based CAD, it is important that data being sent between clients is accurate. This is especially important for the thin-server, thick-client architecture of MESH CAD implementations because each operation must be performed in the exact same way on each client to ensure model consistency. If any data is corrupt or missing, it is not guaranteed that each client will be viewing and interacting with the same model which provides Requirement 9.

Requirement 9 *Complex inter-feature reference storage must prevent update anomalies.*

To ensure data validity, a number of enhancements have been made to previous work in this area. The first enhancement stores the CAD Neutral data in an object-oriented way which mirror the way CAD programs store their own data. By preserving the object hierarchy, much fewer mistakes are made when incorporating a new CAD system into the interoperability client.

5.1 CAD Object Mirroring

To support real-time collaboration in a multi-engineer CAD environment, a central relational database is used to record and store CAD operations as they are performed and sent to other clients. The NPDB saves the commands used to create the model - alleviating the need to save local copies and ensuring all clients have access to the same version of the model. To support the associative parametric abilities of modern CAD software in a MESH CAD environment which utilizes a database, a CAD-neutral operation must be formatted in a database-friendly manner. To accelerate the process of incorporating new CAD packages, the neutral data should also be stored in a manner consistent with the way the native CAD system represents CAD objects. To accomplish both of these requirements, the database should contain a table for each type of supported CAD object. Database relationships are created to mimic inheritance of objects. For example, the feature table implements a one-to-many relationship with the 2d-sketch table, as seen in Figure 5.1. CAD objects that have multiple supported creation methods are represented with multiple tables. The plane-feature table implements the one-to-many relationship with both offset-plane and fixed-plane. features that reference other features, such as the build plane for a sketch-feature, are represented by a one-to-one relationship. Making a database table for each CAD type helps meet Requirement 8. An Object Relational Map (ORM) is used to represent the database tables as programming objects.

A wrapper class is created in code for each CAD object that is responsible for translating between the CAD-specific version of the object to the neutral representation created by the ORM. The wrapper classes follow the same inheritance hierarchy as the neutral objects, and map one-to-one with the CAD-specific features that can be created by the engineer. As new objects are created on one client, the translation method in the respective CAD operation's wrapper class is called to gather all necessary information to recreate the operation on any other system, and fill out the neutral object created by the ORM. The netural object is then transmitted to the central server to be distributed to the other clients in the model and to the database. Each of the remote clients receives a copy of the neutral object, which is then translated to the CAD-specific version by the corresponding wrapper class, and put in the model.

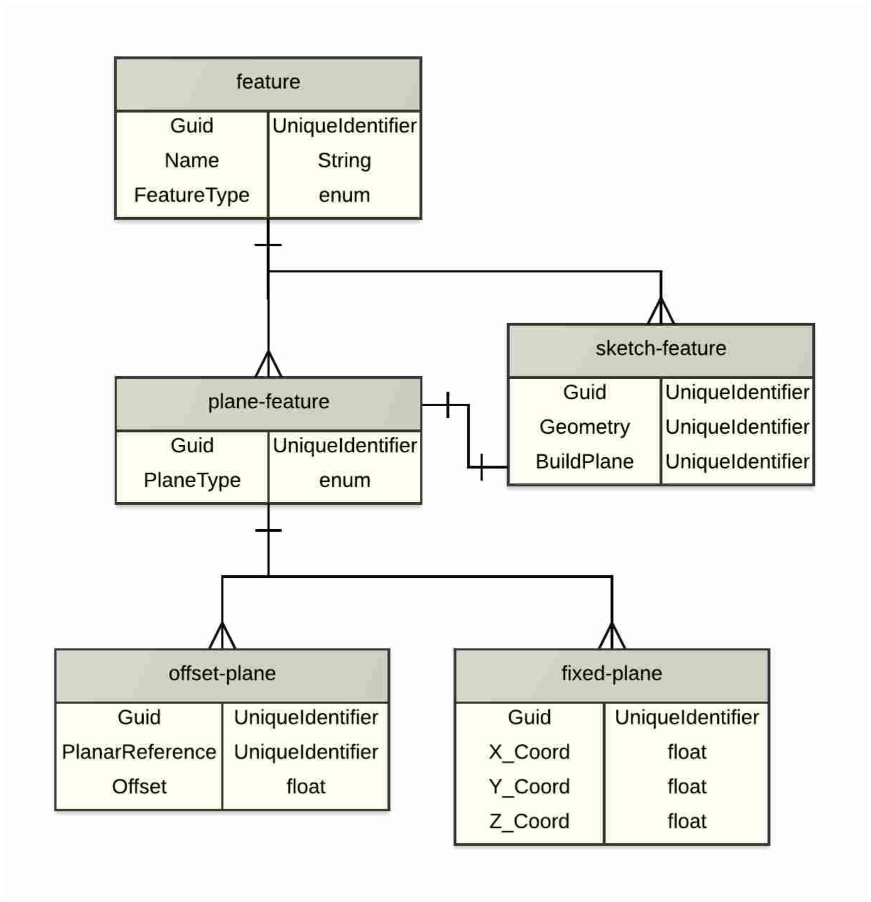


Figure 5.1: Database Schema for Sketch Features and Plane Features.

Besides gathering all required information necessary for recreating a CAD object remotely, the wrapper classes are responsible for applying unique identifiers to each translated object. These identifiers are used to query and either delete or redefine the objects.

5.2 Referential Integrity

The NPDB is integral to facilitating collaboration between multiple engineers by preventing data corruption through referential integrity. This means that each data field in the database can only refer to one other table rather than multiple tables. This means that each reference can only be directed at one other type of feature. Current MES software works around this problem by creating a null-able column for each type of object that could be referenced by a CAD feature. In this case, a sketch-feature table would have a column for a referenced plane as well as a column for a referenced planar BREP face. For a valid sketch, only one of these fields can be filled out

- the other would be null. This does not meet Requirement 9 because with nullable fields of this type multiple or no fields could hold data, resulting in an invalid data state.

To meet this requirement, each property of a CAD feature referencing another object has a foreign key constraint pointing to the most generic table, as seen in Figure 5.2.

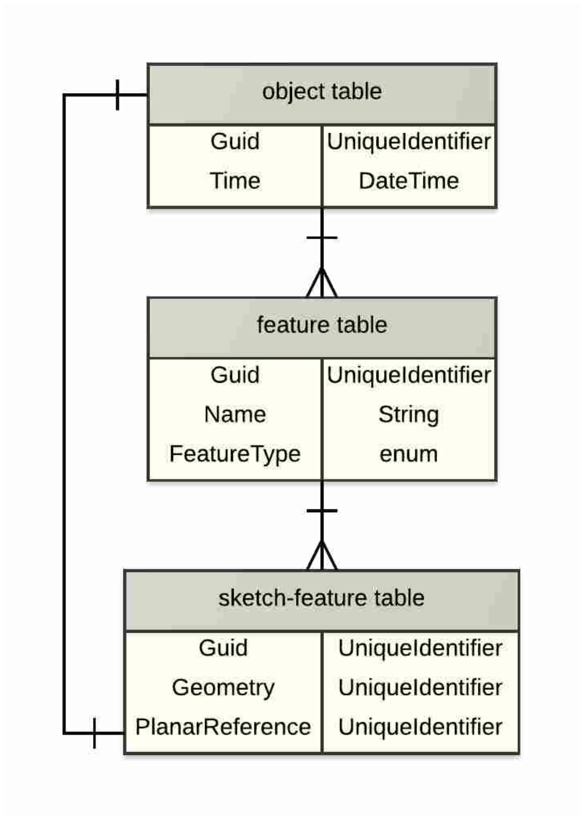


Figure 5.2: The Planar Reference Field Is Associated with the Generic Object Table.

Since each table has an association with the generic table, any object can be referenced by the property. When a new CAD feature is created that has a reference to another CAD object, the unique identifier for the CAD object is stored in the object reference column. This allows references to be stored in the database in such a way that mirrors the CAD clients and so meets Requirement 8.

Because the reference object field is associated with the most generic object table, the database would allow any CAD object be used as a reference - potentially allowing invalid or corrupt data to be saved. Because all read/write methods are abstracted by the ORM, we can ensure that only valid data is written to these fields by enabling ORM objects to implement interfaces.

Neutral CAD objects implement interfaces based on the way they can be referenced. For example, a plane feature can be referenced both as a surface for construction, as well as a direction by its normal vector. In this case, the plane object implements both IPlanarObject and IDirectionalObject interfaces. The planar surface of BREP geometry data also implements IPlanarObject. The attach plane property of the neutral 2d-sketch object is of type IPlanarObject, so only objects that implement this interface can be assigned to this property.

5.3 Implementation

Multi-Reference inheritance was implemented into CAD Interop, a thin-server thick-client, CAD interoperability program developed to integrate the design of part models from Siemens' NX, Dassault's CATIA, and PTC's Creo. Both the client and server programs are written in C#, a .NET programming language. A database, hosted on a SQL server running Microsoft SQL Server 2012, contains a table for each supported CAD feature - with each property represented as a column in the table. As features are created on client computers, data required to create that feature on remote clients is sent in a neutral format through the server to be distributed to the other clients and to the database.

Entity Framework 6 is used as the Object Relational Map (ORM) to represent the database tables as programming objects. Each table on the database implements an association with related tables which, in combination with the ORM, represent an inheritance hierarchy among the objects which mirror the object hierarchy of a typical CAD system. Features with multiple supported creation methods are represented as separate tables which inherit from a more generic table. For example, the DBPlane table represents a plane feature in a CAD system and can be created either as a fixed object by coordinates and a normal, or by an offset from another planar object. Both the DBFixedPlane table and the DBOffsetPlane table implement an association with the DBPlane table which is represented as an inheritance in the ORM object.

The columns for each table on the database are mapped by the ORM into properties for the resulting object. These properties represent the values required to exactly recreate the feature on any supported CAD system, and can include numbers, strings, and references to other features. Entity Framework automatically generates a class for each database table through the use of a template file. As the template file is updated, either from changes made to the database or though changes made to template itself, these classes are automatically overwritten to reflect the change. This in turn updates the ORM object. The database schema is downloaded and, based on the associations set up on the database, a class hierarchy is created with objects representing the database table. Through the use of a graphical engineer interface (GUI), the associations representing an inheritance connection were changed to an inheritance type. By default, associations between tables, such as the association an extrude-feature has with the sketch-feature to be extruded, are preserved and represented as a property of the referenced type in the ORM object. These properties are called navigation properties in entity framework. In the extrude example, the DBExtrude object has a DBSketch property.

To handle properties which can reference more than one type of object, an association on the database between the column representing the reference and the most generic DBInteropObject is created. This allows the unique identifier of any other object to be referenced and stored in the column. By default, the template class translates this navigation property to a DBInteropObject which can accept any object translated by the ORM. To ensure the field allows only valid objects to be referenced, various interfaces representing categories of objects to be referenced were created. For example, the interfaces IPlanarObject and IAxisObject represent objects which can be referenced as a plane or axis respectively. The names of any properties which must reference an interface were changed in the entity framework GUI to a reserved name based on the type of interface they must reference, seen in Table 5.1. The name of the referenced planar object navigation property of an offset-plane-feature, for example, was changed to "ReferencedPlane". In the template file, the method which assigns the navigation properties was modified to include a conditional statement to change the navigation property type when a navigation property name was one of our reserved names. The modified template file then automatically updates the navigation properties for all the class objects with the reserved name to be the correct interface type.

With the navigation properties correctly set to preclude storing invalid data, objects which satisfy the conditions imposed by the interface must implement those interfaces to be referenced by those objects. This maintains referential integrity and so meets Requirement 9. Because classes automatically generated by the template file are automatically overwritten after updates to the template are made, partial classes were written for each object needing to implement one or more interfaces. In C#, a partial class is a continuation of a class declaration which can be written in a

Table 5.1: Reserved Names Associate an Interface with a Navigation Property

Reserved Name	Associated Interface
ReferencedPlane	IPlanarObject
ReferencedAxis	IAxisObject
ReferencedDirection	IDirectionObject
ReferencedPoint	IPointObject
ReferencedLine	ILinearObject

separate file. The partial class then implements the required interfaces. When the template file is used to update the class declaration, the original file will be overwritten but the partial class file, with its interface implementations, will remain unmodified.

5.4 Remarks on Multi-Reference Interface Inheritance

The multi-reference interface inheritance method allows references to mirror the structure of the CAD system rather than allowing multiple vacant data fields (Requirement 8) and maintains referential integrity with these complex references (Requirement 9). With the requirements of the NPDB met for the mathematical and practical foundations including support for complex references between features, the next step is to prove the concepts discussed through the creation of a full implementation of a MESH environment. The development of the remainder of the application had a few unique requirements which will be discussed in the next chapter. Specifically, a pattern to allow the stable use of asynchronous data to generate client-side CAD data tree graphs and a method to enforce proper communication between unknown clients and the server are discussed.

CHAPTER 6. MULTI-USER HETEROGENEOUS SERVER, LOGIC, AND CLIENT LAYERS

With the NPDB fully developed and implemented including complex references, the next step is to create a full application suitable to prove the NPDB concept. In order to extend multi-engineer homogeneous CAD software there are a number of problems that must be solved. Two key problems that must be solved to implement a MESH application effectively are first that a MESH client must be feature-order independent and second that the system be client-agnostic.

One of the issues previously mentioned with respect to the part history representation of CAD data is that it is difficult to determine if CAD feature messages are ordered so that the part can be built. This is particularly important in an asynchronous client-server environment where server messages can arrive at clients out of order. Because of the asynchronous nature of real-time collaboration, a MESH architecture must be able to handle these unordered messages in a stable way, giving Requirement 10.

Requirement 10 *A MESH application must maintain stability despite unordered feature messages.*

Finally, in a MESH architecture the server should behave in exactly the same way regardless of the client it is connected to. If the server must be modified whenever a new client is added to its ecosystem than it is not truly neutral. In fact, to be truly neutral the server should not even be aware what client types are connected to it and the messages to and from heterogeneous clients should be exactly the same. In other words, the MESH application should be client-agnostic as stated in Requirement 11.

Requirement 11 *A MESH application must support clients agnostically.*

6.1 The Psuedo-Singleton Pattern

In order for a system to support parametric CAD models properly, it must represent them as a directed acyclic graph (DAG) where the nodes are features and the edges are parent-child relationships between features as shown in Figure 6.1

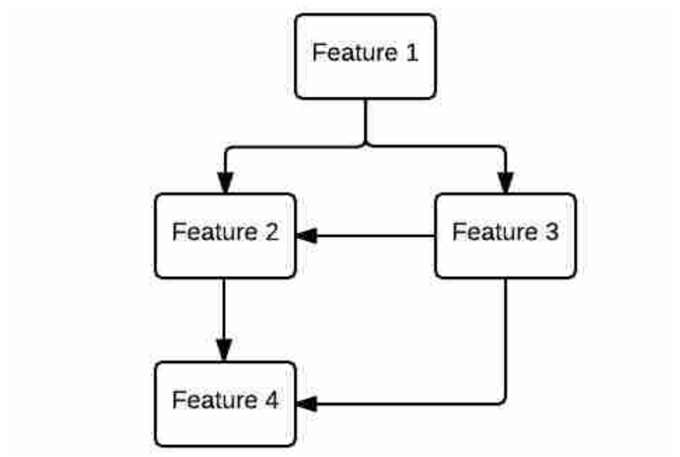


Figure 6.1: CAD Data Should Be Represented as a Directed, Acyclic Graph

This is difficult in a client-server architecture because computer queries typically return a list of features with no regard for their dependencies. The listed features not only do not clearly display dependencies but they can also be returned in a random order. For example, if you queried all of the features from Figure 2 the result might look like the following numbered list:

1. Feature 2

2. Feature 3

3. Feature 4

4. Feature 1

Unfortunately, the part model cannot be built in this order because feature 4 depends on features 2 and 3, feature 3 depends on feature 1, and feature 2 depends on feature 1. A client that simply builds whatever feature messages it receives in order would be unstable and would not meet Requirement 10. So, the order this model must be built in is as follows:

1. Feature 1
2. Feature 3
3. Feature 2
4. Feature 4

Any other build order would result in a fatal application exception. Finding the correct build order for a part is a critically important and difficult task necessary to meet Requirement 10.

The intuitive approach to solve a problem like this is to explicitly find and maintain the correct feature order using consistency managers. Each part could have an overall consistency manager which manages dependencies of all features within the part. This manager could be made more efficient by dividing features into logical groups in a divide-and-conquer style algorithm. All part-level consistency managers in a session would need to be managed by an overall consistency manager. This type of architecture is shown in Figure 6.2

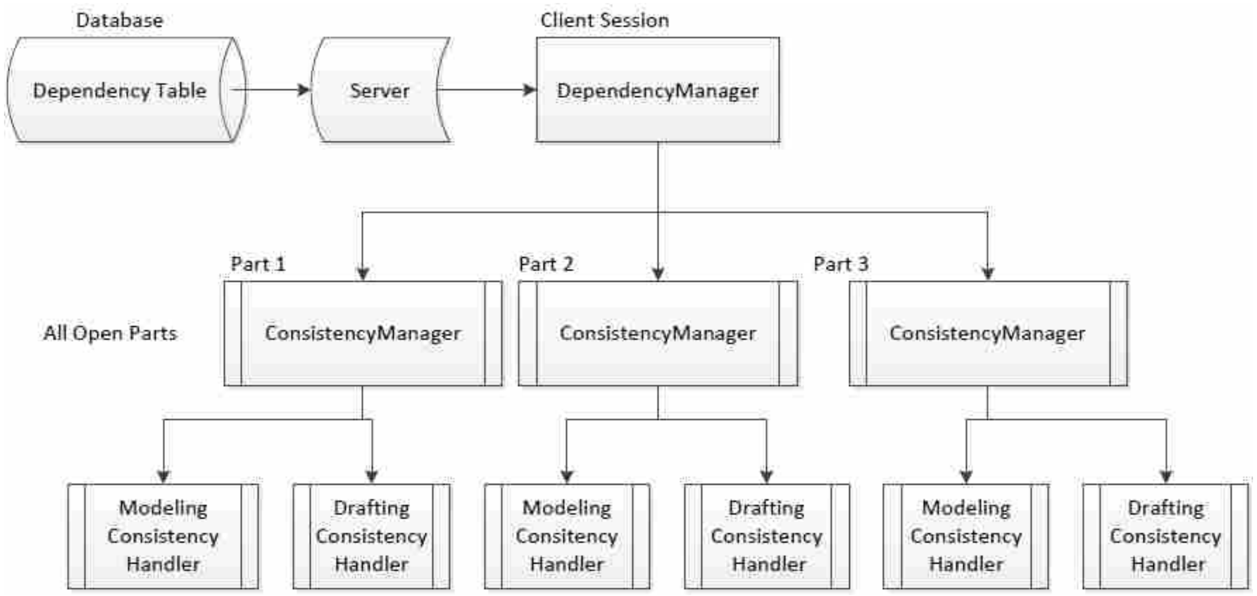

Database
Dependency Table
Server
Client Session
DependencyManager
Part 1
Part 2
Part 3
All Open Parts
ConsistencyManager
ConsistencyManager
ConsistencyManager
Modeling Consistency Handler
Drafting Consistency Handler
Modeling Consitency Handler
Drafting Consistency Handler
Modeling Consistency Handler
Drafting Consistency Handler

Figure 6.2: A Consistency Manager Is the Intuitive Approach to Maintain Model Build Order

This method would function for a time but has several drawbacks. Specifically, the logic for this type of architecture is difficult to develop and must be changed whenever feature definitions are modified. It also requires significant computation time to determine the dependency order and significant memory to store all dependencies for all features of all parts within the assembly. This approach is unlikely to scale to a large assembly with hundreds of thousands of parts and millions of features.

Another way to solve this issue is through the pseudo-singleton pattern which is a development from the singleton pattern shown in Listing 6.1. The pseudo-singleton pattern makes its class constructor private and only allows uniquely identified instances of itself to be queried. If the uniquely identified instance exists it is returned from a static dictionary. If the instance has not been created the class queries the server for the instance, adds it to the static dictionary and returns it. In this way feature instances can be used by a client developer without taking concern for their associations. In other words, any feature built from the pseudo-singleton pattern maintains its own dependencies with no further logic. Sample code that illustrates this pattern is shown in Listing 6.2.

Listing 6.1: The Singleton Design Pattern

```
public class Singleton
{
    private static Singleton Instance {get; set;}

    private Singleton() { }

    public static Singleteon GetInstance()
    {
        if (Instance == null)
            Instance = new Singleton();

        return Instance;
    }
}
```

Listing 6.2: The Pseudo-Singleton Design Pattern

```
public class PseudoSingleton
{
    private static Dictionary<Guid, PseudoSingleton> Instances
    {
        get
        {
            if (instances == null)
                instances = new Dictionary<Guid, PseudoSingleton>();

            return instances
        }
    }
    private static Dictioanry<Guid, PseudoSingleton> instances;

    private PseudoSingleton(Guid Id) { }

    public static PseudoSingleton GetInstance(Guid Id)
    {
        if (!Instances.ContainsKey(Id))
            Instances.Add(Id, new PseudoSingleton(Id));

        return Instances[Id];
    }
}
```

The utility of this pattern can be demonstrated by assigning features to the DAG shown in Figure 6.1. Below in Figure 6.3 is a DAG representation of a part containing a coordinate system (CSYS), a 3D Point built from that CSYS, another point built from the CSYS and relative to the first point, and and a line connecting the two points.

Assigning those features to the queried feature list mentioned above we would get the following list of feature messages:

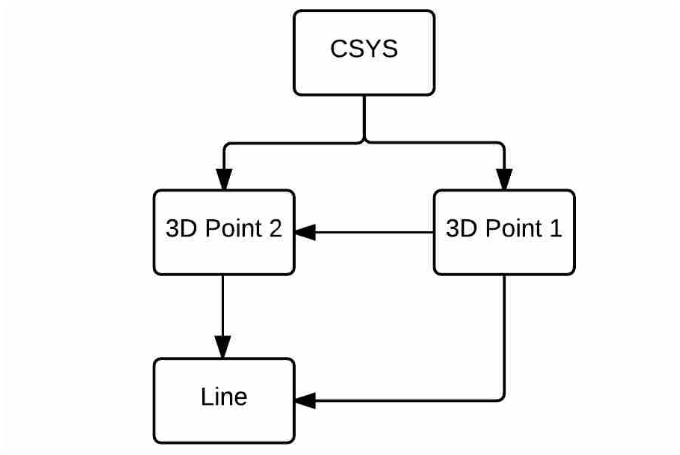


Figure 6.3: A Sample DAG of a CAD Part with a Line Between Two Points

1. 3D Point 2
2. 3D Point 1
3. Line
4. CSYS

The first feature, 3D Point 2 has a dependency on CSYS and a dependency on 3D Point 1. When this message is processed, 3D Point 2 calls the GetInstance method, passing in CSYSs GUID. Since CSYS is not yet in the feature dictionary its constructor is called, it is added to the feature dictionary and it is returned to 3D Point 2. 3D Point 2 then calls the GetInstance method passing in 3D Point 1s GUID Since 3D Point 1 is not yet in the feature dictionary its constructor is called. 3D Point 1 is built off of CSYS so 3D Point 1s constructor calls the GetInstance method passing in CSYSs GUID. Since CSYS is already in the feature dictionary it is simply returned. 3D Point 1 is then added to the feature dictionary and returned. The dependency graph for 3D Point 2 has automatically been assembled. The point is created and added to the feature dictionary.

The second feature, 3D Point 1 has already been placed in the feature dictionary so when the loading algorithm reaches its message it simply verifies that it is already in the feature dictionary and moves on.

The third feature, Line has a dependency on 3D Point 1 and 3D Point 2. When this message is processed, Line calls the GetInstance method, passing in 3D Point 1s GUID. Since 3D Point 1 is already in the feature dictionary it is simply returned. The same happens for 3D Point 2 since it is also in the feature dictionary. The dependency graph for Line has automatically been assembled. The line is created and added to the feature dictionary.

The final feature, CSYS, has already been placed in the feature dictionary so when the loading algorithm reaches its message it simply verifies that it is already in the feature dictionary and moves on.

To implement this pattern, first all CAD feature classes should inherit from a base class which contains a dictionary of all features. As shown in Listing 6.3.

Listing 6.3: Base Object Implementation using the Pseudo-Singleton Pattern

```
public abstract class MUObject
{
    //The GUID of this Object
    public Guid GUID...
    private Guid guid;

    prublic static Dictionary<Guid, MUObject> MUFeatures {get; set;}

    //Sends a Message to the server to delete the server version of
        this MUObject
    public void DeleteFromClient()...

    //Upon recieving a message from the server,
    //this method shall remove the deleted MUObject from the client
    public abstract void DeleteFromServer(MUObject ObjectToDelete,
        CADTypes CADType);
}
```

Since all features will inherit from this base class they have access to this dictionary. Note also the utility of using inheritance to ensure that every feature will have a GUID to uniquely identify it which matches the GUID primary key in the NPDB. There is also a DeleteFromClient method that is fully implemented in this base class and a DeleteFromServer method signature that all sub-types must implement. Pertinent portions of the 3D Point class from the CATIA client are shown in Listing 6.4.

Listing 6.4: 3D Point Implementation

```
public class MUPoint : MUObject
{
    //properties
    public override DBInteropObject serverObject
    {
        get { retukrn ServerPoint; }
    }
    public DBPoint ServerPoint { get; private set; }
    public HybridShapePointCoord CATIAPoint { get; private set; }

    //Public
    //Creates CAD specific point from the server point
    public static MUPoint GetInstanceFromServer(DBPoint Point, CADTypes
        CADType)...

    //Checks to see if the CATIA point is already in the dictionary
    //If it is it returns it. If it is not it creates it
    public statice GetInstanceFromCATIA(HybridShapePointCoord
       clientPoint, Guid partGuid)...

    //Use this method when the server version has changed
    public void UpdateFromServer(ConnectData.DBPoint serverPoint,
       CADTypes CADType)...

    //Use this method when the CATIA version has changed
    public void UpdateFromCATIA(HybridShapePointCoord ClientPoint)...

    public override void DeleteFromServer(MUObject.ObjectToDelete,
       CADDypes CADType)...
}
```

The pseudo-singleton pattern is implemented in the GetInstanceFromCATIA and the GetInstance-FromServer mothods. The GetInstanceFromServer method is shown in Figure 6.5.

Listing 6.5: Implementation of the Pseudo-Singleton Pattern

```
//Creates CAD specific point from the server point and CAD system
public static MUPoint GetInstanceFromServer(DBPoint Point, CADTypes
    CADType)
{
    //If the server point already exists in the dictionary return it
    if (MUFeatures.ContainsKey(Point.GUID))
        return (MUPoint)MUFeatures[Point.GUID]

    //If the point doesn't exist it is created
    MUPoint result = new MUPoint();
    result.CreateCATIAPoint(Point);

    //The newly created MUPoint object is added to the dictionary
    MUFeatures.add(result.GUID, result);

    return result;
}
```

Any other place the application needs to reference a point, it must do so through this method rather than creating the point. This can be seen in the Sphere implementation shown in Figure 6.6.

Listing 6.6: CreateCATIASphere Method Demonstrating Pseudo-Singleton Pattern Use

```
\\Creates a CATIA Sphere
void CreateCATIASphere(DBSphere serverSphere)
{
    HybridShapeSphere sphere;
    GUID = serverSphere.GUID;

    HybridShapePointSphere center = (MUPoint.GetInstanceFromServer(
       serverSphere.CenterPoint, CADTypes.CATIA)).CATIAPoint

    PartDocument partDoc = CurrentPart.CATIAPart;

    HybridShapeFactory hybridShapeFactory = (HybridShapeFactory)partDoc
       .Part.ShapeFactory;
    partDoc.Part.InWorkObject = MUObject.currentBody;

    INFITF.Reference centerRef = partDoc.Part.CreateReferenceFromObject
       (center);

    sphere = hybridShapeFactory.AddNewSphere(centerRef, null,
       serverSphere.Radius, 0, 90, 0, 90);
    sphere.Limitation = 1;
    sphere.set_Name(serverSphere.Name);
    Utilities.SetGuid(sphere, GUID);

    MUObject.currentBody.InsertHybridShape(sphere);

    partDoc.Part.Update();

    CATIASphere = sphere;
    ServerSphere = serverSphere;
}
```

Note that when the sphere center is used, rather than perform any logic checks or take concern whether the center point has been received yet or not, the code simply grabs the point of interest. If the point has been created already it will simply be returned from the dictionary. If the point has not been created it will be automatically created in the correct dependency order. Because of this pattern it does not matter if the point message or the sphere message is sent from the server first, the sphere will assemble its own dependency graph in the correct order.

6.2 Agnostic Business Layer

In addition to working with unordered feature data, a heterogeneous server should work the same irrespective of the client that is connected to it as stated in Requirement 11. From a data storage perspective this has been addressed by creating a neutral parametric database (NPDB) which maintains referential integrity, thus preventing data corruption. From the server perspective this is further enforced by requiring all clients to use a standard set of messages. From a client perspective there must be a standard, documented architecture that any developer can use to integrate their client with the heterogeneous server.

In this case the NPDB was mapped to server-side classes which enforce full referential integrity of the database. Once those classes were made, a standard set of methods was created which performs the standard create, read, update and delete (CRUD) operations both from the client to the server and from the server to the client. The standard methods use the mapped classes as arguments and the server. Inheritance can be used to enforce that any client implement the exact set of methods with the correct arguments. Since data integrity is enforced by those server objects, any client that uses the standard set of methods can fully interact with the heterogeneous server architecture and Requirement 11 is met. The standard set of methods can be seen in Figure 6.4 and the delete methods are enforced in the proposed way as shown in Figure 6.3

With the architecture of a MESH CAD client developed, discussed and implemented the final step to demonstrate that requirements have been met is to demonstrate the software system. This chapter will first review a demonstration of the system, then review the research that has been done and finally make recommendations for the continuation of the research.

6.3 Modeling Demonstration

A sample rocket assembly was modeled by seven engineers on seven different computers. Users worked with a mixture of NX, CATIA and Creo clients. All engineers were given instructions for modeling their individual parts of the assembly. All the features implemented in this research were used in the modeling of the various rocket parts. This project was selected because it has enough complexity to demonstrate feasibility of the MESH architecture while avoiding heterogeneous modeling conflicts between users and the need for heterogeneous undo or redo capability. While methods for homogeneous modeling conflict resolution, undo, and redo have been developed by Red et al. [31,36], there are enough differences between a homogeneous and heterogeneous environment that these methods must be adapted to the heterogeneous environment before they can be used.

All engineers work at the same time creating features on their own CAD client. Unlike other technologies where engineers can all view but only one can edit, all engineers can view and edit the model collaboratively. This allows for a truly parallel work flow where multiple engineers are all editing simultaneously rather than a work flow where one engineer is editing and other engineers are watching or where multiple engineer are editing but cannot see what their team mates are doing.A close-up of the finished assembly in NX, CATIA and Creo is shown in Figure 6.4. The parts are not only the same geometry but are editable, parametric models. A CATIA, NX or Creo engineer can edit the model and the other engineer can see what is being edited in real-time

This architecture not only allows for collaboration within a company, but also real-time collaboration across supply chains. It simplifies the workflow in Figure 1.3 even more than the multi-engineer homogeneous environment. A sample workflow can be seen in Figure 6.5.

Note that not only the conflict resolution tasks within a company have been eliminated, but also the conflict resolution tasks that span multiple companies as well as all data conversion and check workflows and feedback loops. By reducing design-time conflicts engineers will be able to collaborate on large projects effectively. It is also worth noting that because the NPDB has been implemented using a SQL database it can store very large amounts of data effecively.

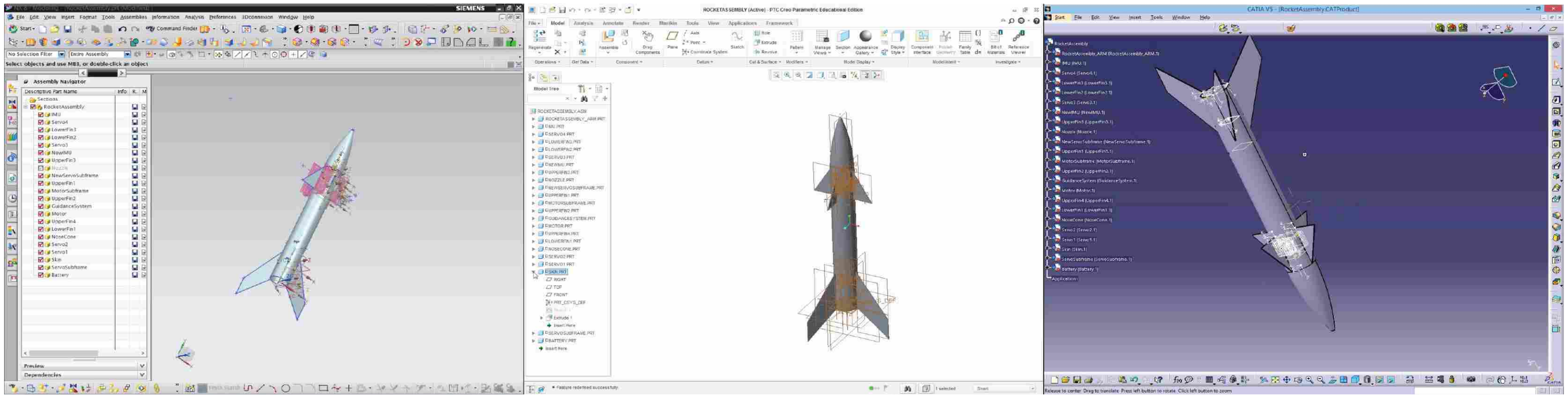


Figure 6.4: Multiple Operators Collaborating in Real Time in a Heterogeneous CAD Environment

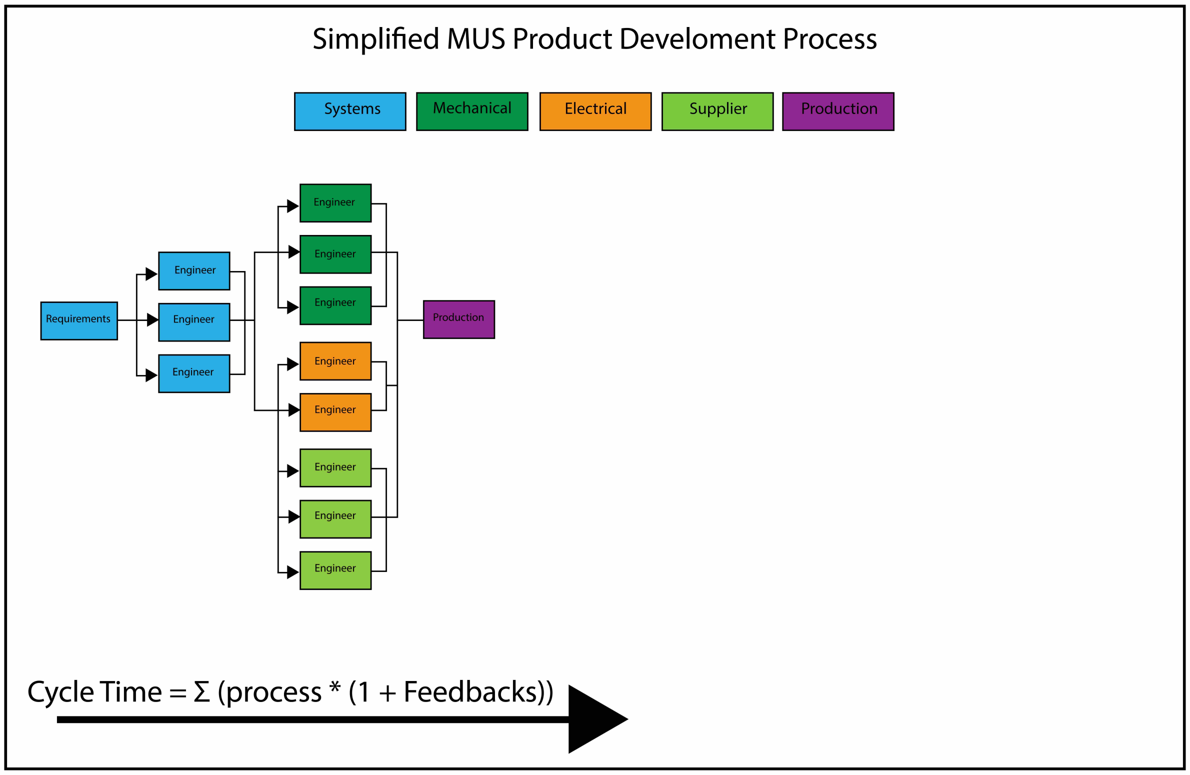

Simplified MUS Product Develoment Process
Systems
Mechanical
Electrical
Supplier
Production
Requirements
Engineer
Production
Cycle Time = Σ (process * (1 + Feedbacks))

Figure 6.5: Sample Workflow Using a MES Heterogeneous CAD Environment

6.4 Remarks on Multi-User Heterogeneous Server, Logic, and Client Layers

This architecture can be used to create an application that has a stable, easy to use data storage method and a client server architecture which can not only maintain stability despite unordered data (Requirement 10) but also work irrespective of client (Requirement 11). An implementation of this architecture will prove the concept of a MESH CAD architecture. The next chapter will discuss a few other needs which have been expressed by our industry partners and then the actual results of implementing the MESH CAD client will be discussed.

CHAPTER 7. CONCLUSIONS AND FUTURE WORK

7.1 Dissertation Review

In this dissertation a software architecture that facilitates MESH CAD has been developed and discussed in such a way that key system requirements have been met. The next four subsections review these requirements and how they were met.

7.1.1 Neutral Parametric Database Mathematical Model

Chapter 3 set four requirements:

1. MESH CAD is only obtainable through the development, implementation and acceptance of a Neutral Paramatric Database.
2. The NPDB must be based on fundamental mathematical representations of CAD features.
3. The NPDB must prevent update anomalies.
4. The NPDB must be compatible with an object oriented class structure.

A MESH CAD system must be built on a sound foundation that represents CAD features generically and accurately, maintains data integrity and maintains clear relationships. In order to achieve this, a neutral parametric database or NPDB was created. In order to ensure that CAD features are well represented, the fundamental mathematical definition of the CAD elements was used rather than simply using API function arguments. This ensures that the CAD features can be read from the database and recreated in any CAD system for any length of time into the future. Update anomalies are prevented by normalizing the database. This is critical because it ensures that no mater how many clients interact with the database or how poorly written they are, they cannot corrupt the CAD data. Finally, by mapping the NPDB database tables to an object oriented class structure the intent of the storage model is clear and the elements are straight forward to work with. While these requirements are necessary in order to create a MESH CAD system they are not sufficient and so the next section will review additional requirements.

7.1.2 Neutral Parametric Database Structured Query Language Model

Chapter 4 set three additional requirements:

5. The Nirvana Fallacy must be avoided by proving the concept with a partial feature set.
6. The NPDB standard must be implemented using industry best practice tools and methods.
7. MESH software must be scalable to an enterprise-scale design environment

It is a common error when developing engineering software to fall victim to scope creep and because of it never complete a development project. The way to avoid this is to acknowledge the Nirvana fallacy and choose a clear, succinct scope. This research accomplished this by working with a central set of the most used sketch and solid features rather than the full set of CAD features. Another common mistake is to try to use the newest, most exciting tools and methods rather than the most established ones. This mistake would result in a system that would not be open to further development because of obscure coding language and tool usage. To avoid this pitfall industry-standard tools such as SQL, C#, and Entity Framework were used. Not only is using a standard tool set important to ensure a smooth development project, but these standard tools are commonly used by industry members and so are designed to scale to their needs. By meeting these three requirements, the architecture not only has a sound foundation but also has a highly pragmatic implementation that can be built and expanded upon.

7.1.3 Multi-Reference Interface Inheritance

Chapter 5 set forth two additional requirement:

8. The NPDB must store inter-feature references in a way that mirrors existing commercial CAD systems.
9. Complex inter-feature reference storage must prevent update anomalies.

One very important and often overlooked aspect of CAD data storage are the relationships between the CAD features. Research typically focuses on the features themselves while neglecting how they associate with each other. It is critical that a system meant to store CAD data represents all aspects of the data. For this reason all references between features must be accurately stored in the database. Not only must they be stored, but they must be stored in such a way that update anomalies are prevented. For this reason the NPDB was designed to support mutable inter-feature references accurately, succinctly, and in a way that only allows the appropriate data elements to be written to the database. Without meeting these requirements, a MESH CAD system would have low data integrity and thus be highly unstable. One mistake could corrupt an entire assembly structure, rendering it unusable.

7.1.4 Multi-User Synchronous Heterogeneous Server, Logic, and Client

Chapter 6 lays out two more requirements:

10. A MESH application must maintain stability despite unordered feature messages.
11. A MESH application must support clients agnostically.

Finally, a MESH CAD system's logic and client layers must be as stable and ordered as the data layer they are built upon. For this reason it is important that a logic layer be written which maintains the proper order and relationships between features received from the server. While it is possible to maintain these relationships through complex algorithms, it is far superior to do so through an elegant design pattern and so the pseudo-singleton pattern was developed to automatically maintain these relationships regardless of how data is received from the server. Additionally, standard function methods were developed so that even a novice programmer can write a new MESH CAD client which interacts with the NPDB. This will allow future students to continue to improve and work with the MESH CAD architecture developed for this dissertation. These eleven requirements are necessary to build a MESH CAD environment and they are sufficient to prove the conceptual architecture.

7.2 Future Recommendations

While this dissertation represents a strong start at creating a MESH CAD system, there is still much to be done. With a project of this nature where many doubt the possibility of the approach, the first step is to create a proof of concept which was done in this case. The next step is to identify key research questions and resolve them in order to prepare the technology for commercialization. In this spirit a few particular problems that are currently under research by students of the BYU CADLab are:

- Model History
- Version Control
- Variant Branching
- Variant Merging
- Consistent BREP Naming

A model history is an important part of the data integrity of a CAD system and is a foundational step towards supporting heterogeneous undo and redo operation. With a model history in place, if designers make mistakes in their work those mistakes can be reversed, either at a local or a global level as described for a homogeneous environment by Red et al. [36] Version control allows engineers to flag important milestones in their design project. This will allow them to prepare models for a design review and then continue working uninterrupted. Variant branching and merging will allow engineers to explore the design space without corrupting the production product. Defining branch and merge operations for the part graphs within this MESH CAD system is also foundationally important in order to support heterogeneous optimistic conflict resolution, similar to what was researched for a homogeneous enviornment by Red et al. [31] Finally, Consistent BREP naming is a critical piece of a MESH CAD system because it will allow the storage of items like edges, vertices and faces which do not exist as independent features but rather as the result of features. Once these research projects are completed there is still more work to be done. A few items to start with are:

- Heterogeneous Global Undo and Redo Operations

- Heterogeneous Local Undo and Redo Operations
- Heterogeneous Optimistic Conflict Resolution
- Complex Curves and Surfaces With Proprietary Interpolation Algorithms
- Features Exclusive to One CAD System
- Multi-disciplinary View Management

Engineers frequently make mistakes as they work and so undo and redo operations on both a global and local scale are critical for a smooth work flow in a heterogeneous environment. With multiple engineers working together in close quarters operational conflicts are a regular occurrence. Because of this automated conflict resolution operations will be necessary for MESH CAD system stability. As was mentioned in the dissertation, some elements like curves and surfaces have different mathematical and API definitions. A robust method to store these elements and allow them to be both accurately viewed and easily edited must be developed. Similarly, features that are exclusive to one CAD system or are missing from one CAD system must be represented in the system so that it can work with the union of all CAD features rather than their intersection. Finally, an important step in engineering design will be to take the MESH CAD system developed in this dissertation to become a MESH CAx system which can store and work with all types of engineering data.

REFERENCES

[1] Brunnermeier, S. B., and Martin, S. a., 1999. Interoperability cost analysis of the US automotive supply chain: Final report Tech. Rep. 7007. 1, 25

[2] Amdahl, G. M., 1967. "Validity of the single processor approach to achieving large scale computing capabilities." In *AFIPS spring joint computer conference*, pp. 187–196. 1

[3] Tupper, E. C., 2013. *Introduction to Naval Architecture, 5th Edition*. Elsevier. 2

[4] Red, E., Jensen, G., French, D., and Weerakoon, P., 2011. "Multi-user architectures for computer-aided engineering collaboration." *2011 17th International Conference on Concurrent Enterprising*(Ice), pp. 1–10. 7, 16, 19

[5] Basu, D., and Kumar, S. S., 1995. "Importing mesh entities through IGES/PDES." *Advances in Engineering Software,* **23**(3), pp. 151–161. 10

[6] Gu, H., Chase, T. R., Cheney, D. C., Bailey, T. ., and Johnson, D., 2001. "Identifying, Correcting, and Avoiding Errors in Computer-Aided Design Models Which Affect Interoperability." *Journal of Computing and Information Science in Engineering,* **1**(June 2001), p. 156. 11

[7] Tien-Chien Chang, Richard A. Wysk, H.-P. W., 2006. *Computer-Aided Manufacturing*., 3rd ed. Prentice Hall. 11

[8] Marjudi, S., Amran, M. F. M., Abdullah, K. A., Widyarto, S., Majid, N. A. A. M., and Sulaiman, R., 2010. "A Review and Comparison of IGES and STEP." *Proceedings Of World Academy Of Science, Engineering And Technology*(January 2016), pp. 1013–1017. 11

[9] Ranyak, P., 1994. "Application Interface Specification (AIS), Version 2.1." *Consortium for Advanced Manufacturing International (CAM-I). Integrity Systems, USA*. 11

[10] Choi, G., Mun, D., and Han, S., 2002. "Exchange of CAD part models based on the macro-parametric approach." *International Journal of CAD/CAM,* **2**(1), pp. 13–21. 11, 12, 19, 20, 22

[11] Iyer, G. R., 2001. "Development of API-Based Interfaces to Enable Interoperability Between CAD Systems During Design Collaboration." PhD thesis, The University of Texas at Arlington. 11

[12] Ganapathi, S., 2002. "A Software Model for Interoperability." PhD thesis, The University of Texas at Arlington. 11

[13] Aspire3D. 11

[14] Mun, D., Han, S., Kim, J., and Oh, Y., 2003. "A set of standard modeling commands for the history-based parametric approach." *CAD Computer Aided Design,* **35**, pp. 1171–1179. 12, 40

[15] Li, W., Ong, S., Fuh, J., Wong, Y., Lu, Y., and a.Y.C. Nee, 2004. "Feature-based design in a distributed and collaborative environment." *Computer-Aided Design,* **36**, pp. 775–797. 12, 19, 40

[16] Li, M., Yang, Y., Li, J., and Gao, S., 2004. "A preliminary study on synchronized collaborative design based on heterogeneous CAD systems." *... Cooperative Work in Design, 2004. ...* (310027). 12, 19, 20, 22

[17] Li, M., Gao, S., and Wang, C. C. L., 2007. "Real-Time Collaborative Design With Heterogeneous CAD Systems Based on Neutral Modeling Commands." *Journal of Computing and Information Science in Engineering,* **7**(June), p. 113. 12, 19, 20, 22

[18] Dou, W., Song, X., and Zhang, X., 2009. "A language of neutral modeling command for synchronized collaborative design among heterogeneous CAD systems." *2009 1st International Conference on Information Science and Engineering, ICISE 2009*, pp. 12–15. 12

[19] Dou, W., and Song, X., 2013. "Operation Command Transformation of Synchronized Collaborative Design Upon Heterogeneous CAD Systems." *Journal of Algorithms & Computational Technology,* **7**(4), pp. 423–448. 12

[20] Song, X., Dou, W., and Zhu, J., 2010. "Implementation of collaborative design system upon heterogeneous CAD systems using a feature-based mapping set." *Proceedings of the 2010 14th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2010*, pp. 510–515. 12

[21] J. Chen, Y. M. C. W. C. A., 2005. "Collaborative Design Environment with Multiple CAD Systems." pp. 367–376. 12

[22] Zhang, X., and Dou, W., 2009. "An approach of constructing neutral modeling command set of synchronized collaborative design upon heterogeneous CAD systems." *Proceedings - International Conference on Management and Service Science, MASS 2009*, pp. 0–3. 12

[23] Rappoport, A., 2003. "An architecture for universal CAD data exchange." *Proceedings of the eighth ACM symposium on Solid modeling and applications - SM '03*, p. 266. 13, 20, 22

[24] Proficiency. 13

[25] Tae-Sul Seo Sang-Uk Cheon Soonhung Han Lalit Patil Debasish Dutta, Y. L., 2005. "Sharing CAD models based on feature ontology of commands history." *International Journal of CAD/CAM,* **Vol 5, No**(0). 13

[26] Sun, L. J., and Ding, B., 2009. "Heterogeneous CAD data exchange based on cellular ontology model." *2009 WRI World Congress on Software Engineering, WCSE 2009,* **1**, pp. 46–50. 13

[27] Tessier, S., 2011. "Ontology-Based Approach To Enable Feature Interoperability Between Cad Systems." PhD thesis, Georgia Institute of Technology. 13

[28] Hepworth, A., Tew, K., Trent, M., Ricks, D., Jensen, C. G., and Red, W. E., 2014. "Model Consistency and Conflict Resolution With Data Preservation in Multi-User Computer Aided Design." *Journal of Computing and Information Science in Engineering,* **14**(June), p. 021008. 16

[29] Hepworth, A. I., Tew, K., Nysetvold, T., Bennett, M., and Greg Jensen, C., 2014. "Automated Conflict Avoidance in Multi-user CAD." *Computer-Aided Design and Applications,* **11**(May 2014), pp. 141–152. 16

[30] Hepworth, A. I., Nysetvold, T., Bennett, J., Phelps, G., and Jensen, C. G., 2014. "Scalable Integration of Commercial File Types in Multi-User CAD." *Computer-Aided Design and Applications,* **11**(May), pp. 459–467. 16, 49

[31] Red, E., French, D., Jensen, G., Walker, S. S., and Madsen, P., 2013. "Emerging Design Methods and Tools in Collaborative Product Development." *Journal of Computing and Information Science in Engineering,* **13**(September), p. 031001. 16, 68, 74

[32] Red, E., Holyoak, V., Jensen, C. G., Marshall, F., Ryskamp, J., and Xu, Y., 2010. "V-CAx: A research agenda for collaborative Computer-Aided Applications." *Computer-Aided Design and Applications,* **7**(3), pp. 387–404. 16

[33] Cai, X., Li, X., He, F., Han, S., and Chen, X., 2012. "Flexible Concurrency Control for Legacy CAD to Construct Collaborative CAD Environment." *Journal of Advanced Mechanical Design, Systems, and Manufacturing,* **6**(3), pp. 324–339. 16

[34] Maher, M. L., and Rutherford, J. H., 1997. "A model for synchronous collaborative design using CAD and database management." *Research in Engineering Design,* **9**, pp. 85–98. 16

[35] Sederberg, T. W., 2011. *Computer Aided Geometric Design Course Notes*. 31

[36] Moncur, R. a., Greg Jensen, C., Teng, C. C., and Red, E., 2013. "Data consistency and conflict avoidance in a multi-user CAx environment." *Computer-Aided Design and Applications,* **10**(May 2014), pp. 727–744. 49, 68, 74