2006-03-23

# The Storage of Parametric Data in Product Lifecycle Management Systems

Jonathan Gary Lund
*Brigham Young University - Provo*

# THE STORAGE OF PARAMETRIC DATA IN PRODUCT LIFECYCLE MANAGEMENT SYSTEMS

by

Jonathan Gary Lund

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

Brigham Young University

April 2006

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a dissertation submitted by

Jonathan Gary Lund

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

| | |
|---|---|
| Date | C. Greg Jensen, Chair |
| Date | Spencer P. Magleby |
| Date | Joseph J. Ekstrom |

BRIGHAM YOUNG UNIVERSITY

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

Date

C. Greg Jensen
Chair, Graduate Committee

Accepted for the Department

Matthew R. Jones
Graduate Coordinator

Accepted for the College

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

ABSTRACT

THE STORAGE OF PARAMETRIC DATA IN

PRODUCT LIFECYCLE MANAGEMENT

SYSTEMS

Jonathan Gary Lund

Department of Mechanical Engineering

Master of Science

Product development companies are continually seeking methods to increase efficiency while maintaining quality. Distributed development is also more important than ever before as industries globalize. These forces have driven firms to adopt formal data management practices that allow groups and individuals to work from singular, centralized data source that are secure, reliable, and support collaboration. This thesis proposes a methodology to leverage globalized infrastructures for the efficient storage of product variations.

The methodology is proved through a working prototype using the market leader in Product Lifecycle Management (PLM) systems, Teamcenter Engineering. First, paradigms are set forth for the storage of various types of engineering documents in PLM

systems in parametric formats. Then the use of these paradigms is exemplified by various programs retrieving and storing document variations in the form of PLM metadata. Finally, the results show that this methodology produces drastic increases in system performance as well as the enabling of PLM-compatible automation and optimization. The impacts of these findings have significant implications for industry and has generated interest from several global engineering firms and academic journals.

ACKNOWLEDGEMENTS

My wife, Jennifer, children, Natalie, Amy, and Adam.
Nathaniel Fife, Brett Black, Paul Nelson, and Dr. Jensen

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1: INTRODUCTION

Product development firms are deploying product lifecycle management (PLM) systems to improve efficiency, product quality, and reduce the time to market. These systems' overarching objectives are the management, maintainance, and eventual retirement of the company's portfolio of products. Despite these advantages and objectives, PLM systems inhibit their ability to integrate knowledge-based methods used to automate and optimize tasks. This thesis proposes a method by which companies can store parametric data in PLM systems in a flexible, automation-friendly manner.

Engineering today is heavily dependent on computers and their ability to manipulate data. From design, analysis, and manufacturing to product support and recycling, a product's data undergoes a host of operations that transform a concept into a usable product. The product's data gets pre- and post-processed many times and appears in the form of emails, word-processing documents, spreadsheets, CAD models, measured data, analyses, tool paths and customer feedback—to name a few. The work done by a company in the form of these documents to develop a product quickly generates a seemingly unmanageable volume of data and supporting data, and demands a clear method of organization and document control. This has led to the development of Product Lifecycle Management (PLM), a framework for controlling documents and revisions more closely than regular file systems.

Object reusability is the basis for knowledge-based engineering (KBE) and mass customization. In many design situations a computer program can be created to produce a number of product variants in a short amount of time by iterating the design process with different parameters. This helps create more effective product designs and often produces iterations that may otherwise not be possible. Reusable or "parametric" documents are becoming more commonplace and are helping companies save time and money because they allow valuable documents to be reused many times, eliminating the need for costly repetition of work.

## 1.1 Problem Statement

Although PLM and KBE are both important to engineering, there is no commercially-available solution to using them together. Automation and parametric technologies are not found in out-of-the box PLM systems and knowledge-based engineering frameworks lack ties into PLM systems. This may be due to the fact that the methods required to produce such functionality are non-conventional and require a working knowledge in many areas to properly plan and integrate the various systems. Some companies have benefited from storing design parameters in databases that can quickly provide shared data to multiple programs, but such circumstances are highly customized and case-specific due to the lack of commercially-available solutions.

In typical PLM situations, companies are required to have a full models stored on the server. Design variations that differ even by small amounts still require the same amount of PLM storage as any other model. This includes part files, documents, spreadsheets, etc. Storage requirements are then greatly amplified with the storage of revisions and older

versions. This presents the following problems that can inhibit productivity and discourage proper PLM usage:

- Large storage and backup requirements
- Model variability in topology and parameterization
- High network bandwidth
- Model representation in proprietary formats
- Nearly impossible to search models for internal attributes

## 1.2 Thesis Objective

The objective of this thesis is to present a design paradigm for storing model variations as basic PLM metadata. The paradigm allows master parametric models to be saved in PLM systems normally, while design variations are stored completely inside the PLM system's database. The integration is seamless and independent of PLM system make. The problems presented above could potentially be minimized by achieving:

- 1-3 orders of magnitude smaller file sizes
- Model standardization and modularization
- Open (non-proprietary) storage of key design parameters
- Database-searchable data format

In designing such a paradigm, the following questions arise:

- Can variant data be efficiently stored in PLM systems?
- Can PLM systems be used as a basis for automation?
- Can parametric data storage improve PLM system benefits?

To test the proposed paradigm, a plug-in to a commercial PLM system will be developed and tested. The results of this work will serve as a guide for those planning collaborative, knowledge-based data management systems.

## 1.3 Delimitation of the Problem

While this opens the door to many knowledge-based engineering PLM integrations possibilities including ERP, optimization, and multi-site collaboration, this research is limited to the development of the storage framework and schema. There are many possibilities to explore once a framework is established—the implementation of these will provide some exciting future research.

Considerable research has already been implemented using this work as its foundation. It includes advanced topics such as workflow automation and transactional optimization. These topics will only be discussed in the context of how they might be accomplished, and interested readers should refer to Fife's work. [56]

# CHAPTER 2: BACKGROUND

The development of a parametric PLM framework involves the usage of various data communications tools. This chapter discusses some fundamentals of Knowledge-Based Engineering (KBE), network communications and the technology behind PLM systems and their databases.

## 2.1 Knowledge Based Engineering

KBE is a rapidly growing trend in engineering due to the gains it provides in performance, efficiency, and quality. The term is somewhat loosely used and basically means that development efforts should build on existing company knowledge. While this has obviously been used for years, recent tools are being built with specific capabilities for leveraging previous design knowledge.

The most common example of KBE is parametric solid and surface models, or CAD models that can morph with changes of design parameters. This thesis uses a broad, abstract view of KBE by including tools such as: spreadsheets; legacy codes and scripts; analysis configurations; and essentially any other design task.

### 2.1.1 Parametric Models

Parametrics CAD models revolutionized the capabilities of CAD systems. It allowed the system to be programmed and models to be reused for multiple purposes. CAD models parameterization now happens automatically in the background as designers model. Parameters can then be modified and given "knowledge" by assigning parameters to be the results of equations or functions (see Figure 1).

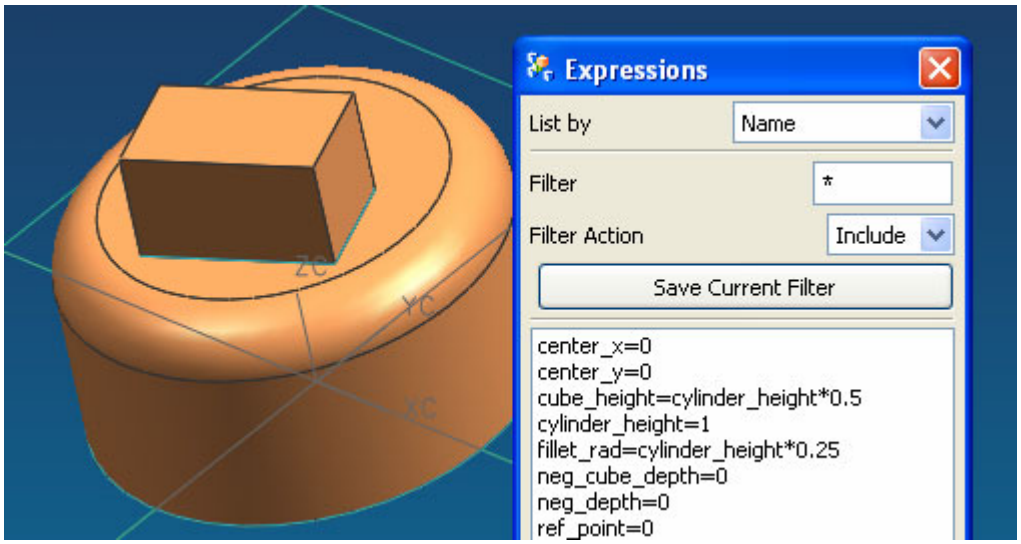


**Figure 1- Parametric variables (set by equations) controlling a CAD model.**

### 2.1.2 A Broader View of Parametrics

The term "parametric" refers to objects being controlled by parameters. A classic example of this is the parametric equation of a line.

$$P(t) = P_0(1-t) + P_1 t$$

While this may be quite fundamental, an analysis of the components can be quite useful in broadening the perspective from just CAD models. First, the left side, ***P(t)*** is the

expected output—we know what form to expect the output to be in—that is, of ***P***. For geometric points, this would be a vector with three values. Then, on the right side of the equation, there is some degree of data (or knowledge) built into the equation by specifying the points $\boldsymbol{P}_0$ and $\boldsymbol{P}_1$. Finally, the value for *t* is left out, meaning it is specified as the input parameter that will vary.

If we analyze a spreadsheet in the same way, we can see that it can be treated and used in a similar fashion. The spreadsheet's output is defined by the formatting and location of certain cells of interest. The built-in knowledge is composed of all the formulae, formatting, macros, etc. that are used to compute the cells of interest. Finally, the cells that one would vary when using the spreadsheet are the input, just as the *t* variable in the parametric equation (see Figure 2).

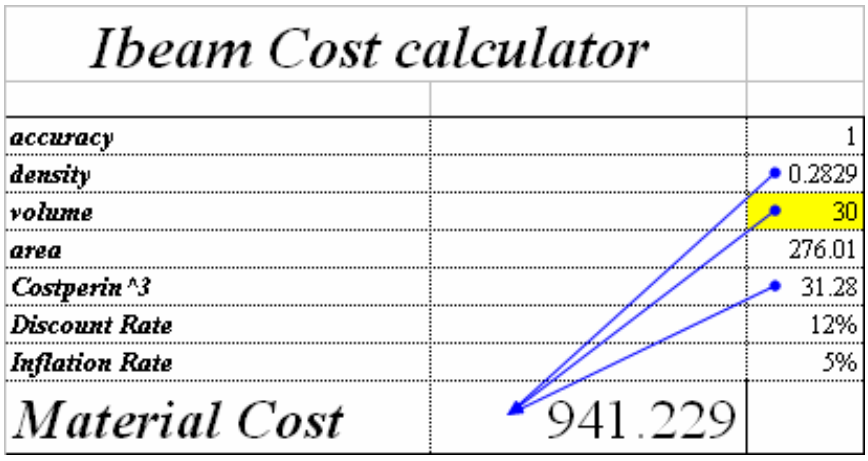


**Figure 2- Spreadsheet viewed as parametric object: input (shaded) combines with existing knowledge to compute output (cost)**

This method of breaking a spreadsheet down into a parametric object can be applied to virtually any other engineering task or activity. It is, essentially, a free-body diagram of the task and its data. Some examples are shown in Figure 3.

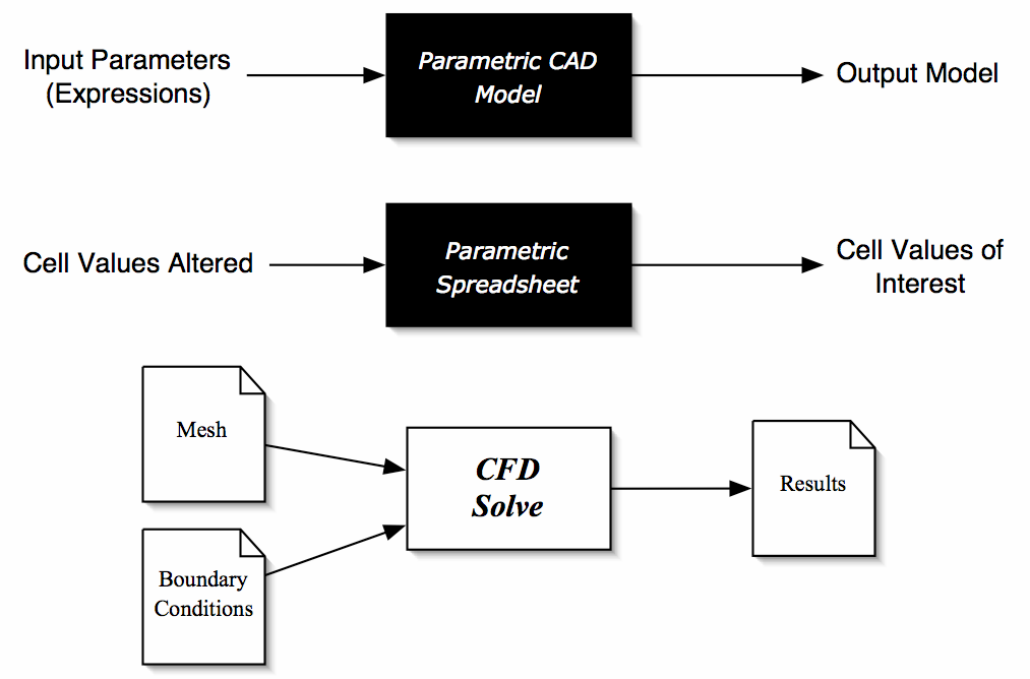


**Figure 3- Free-body flow diagrams of various engineering tasks**

Figure 3 is referred to as a free-body flow diagram because it isolates a single task and indicates all inputs and outputs to the system. This method can be applied to any product development operation to define the steps in development.

## 2.2 Network Communications

Network communications is the enabler of global business. It allows companies to share their data not only within their organizations, but with other plants, suppliers, and customers. In order to properly plan to leverage collaborative systems, it is important to have a basic understanding of how these systems operate. Their strengths and weaknesses

ultimately drive the capabilities of global product development. The most recent collaborative environments operate with one of the two main architectures, client-server and peer-to-peer.

### 2.2.1 Client-Server Systems

A client-server system refers to systems where there is a central server that can provide service to more than one client simultaneously. Systems that use application servers are another alternative for collaborative solutions. In this configuration, the server is the repository for all knowledge and communication designated to be public. Most of these systems come with an industrial application server made by specialized companies, however, some PLM systems are configured with their own. Engineering frameworks such as PLM systems generally use application servers as only a sub-component rather than a full system [91].

Areas where these systems generally outperform peer-to-peer configurations are B2B networking, process and error logging, security and universal access. Again, this is a generalization; the intention here is to state the natural capabilities of the architectures.

While application servers perform well, they do not provide a complete solution to companies' needs. Therefore, adding application servers is redundant if its features can be integrated with PLM software; and redundant IT infrastructures can equate to significantly higher costs not only for physical architectures but training and support as well.

### 2.2.2 Peer-to-Peer Systems

Some emerging collaborative frameworks are built on peer-to-peer technology. These systems act as their own servers and clients so computers can interact and work together directly to solve problems. Systems such as these can run independently as well as serve and make requests with other computers. One such architecture can be seen in Figure 4.

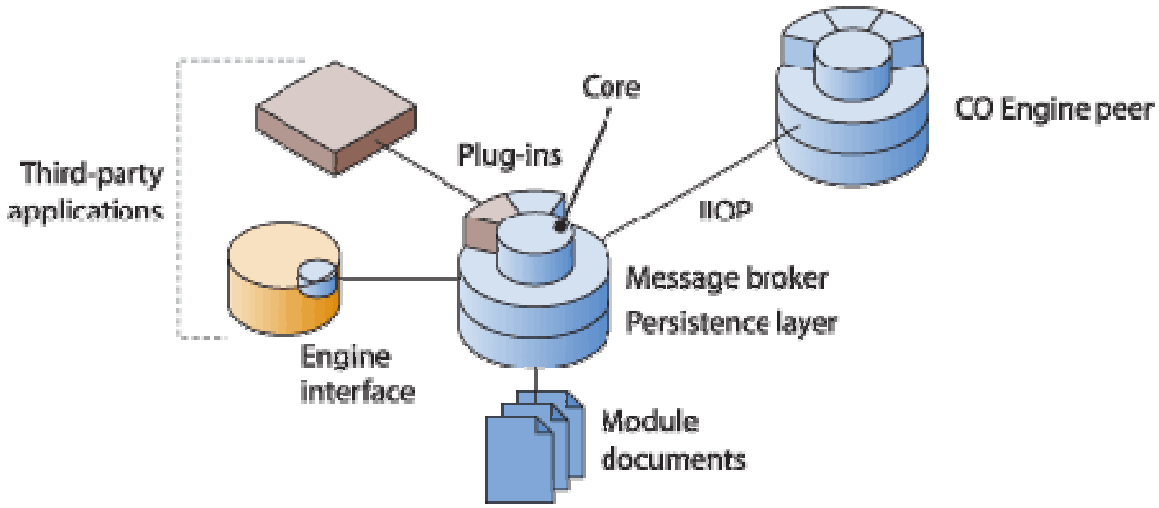


**Figure 4- Peer-to-peer architecture of Oculus CO**

Speed is one of the main advantages of the peer-to-peer architecture. Requests are made through direct socket connections and without any intermediaries, allowing very fast processing and real-time communication. However, this direct connection also has many disadvantages due to the lack of an overarching control system. Version control is highly disorganized due to the independence of each computer. Security is a major concern as well, and although work has been done to make individual connections secure, individual connections between companies are not allowed across firewalls without lowering security in other areas.

**Table 1- Challenges inherent to peer-to-peer systems**

| *Limitation* | *Description* |
|---|---|
| Version control | Models are generally configured to automatically update based on other computers' data. Models can get damaged by bad data. |
| Security | Free communication must be allowed between all federated computers. This can be difficult to monitor and unusual firewall configurations are often required. |
| Business-to-business | Because of the peer-to-peer connections, individual nodes must be given public addresses in order to serve outside requests. For companies with sensitive information this is generally out of the question. |
| Error handling | Direct connections are less stable than transactional systems because requests and run-time memory can be lost when errors occur. |
| Universal Access | Because projects and data are spread among many "peers," there is no one place to go to access project data. Connection to web servers for external monitoring is also made difficult. |

Some of the concepts from peer-to-peer networks will be used for this thesis. One example of this is where programs such as ANSYS require that they be run by themselves and therefore do not have a link back into the PLM database. In order for this link to be made ANSYS will be set up as its own agent, much like agents in peer-to-peer networks, which will have the ability to connect back through a socket to the PLM software. More on this and its implications such as security will be discussed in the method section.

## 2.3 PLM Systems

A PLM system is a software/hardware configuration that provides collaborative access and highly sophisticated management of documents. PLM is a term that encompasses many other techniques including engineering data management (EDM), document management, product information management (PIM), technical data management (TDM), technical information management (TIM), image management and others [5]. Many consider PLM to be a simple data repository or "vault" that has little more than a file-saving role. However,

PLM is providing much more than this, and has thus begun to be called product lifecycle management (PLM). According to Philpotts, it also "integrates and manages processes, applications and information…" [5]. PLM offers benefits in many different ways and to many people as shown in Table 2.

**Table 2- PLM users Philpotts, Mike (1996)**

| *Individuals* | *Industry Segments* |
|---|---|
| Chief executives | Aerospace |
| Technical directors | Automotive |
| Chief engineers | General mechanical manufacturing |
| Engineering managers | Electrical and electronic products |
| Engineers of all disciplines | Computer manufacturing |
| Heads of IT | Defense industries |
| Design managers | Petroleum industries |
| CAD/CAM/CAE managers | Chemical and process engineering |
| Production engineers | Food and drinks industries |
| Project managers | Pharmaceuticals |
| Operations and maintenance managers | Power generation |
| | Construction companies |
| Estimators | Transport operators |
| Purchasing officers | Utilities: electricity, water, telecom |
| Marketing and sales managers | Design and management consultants |

PLM systems allow many users to access a single database full of documents and company data. As stated above PLM can store all types of electronic documents and related information such as: product configurations, part definitions, CAD models and drawings, images, engineering analysis models and results, NC programs, spreadsheets, presentations, web pages, videos, and documents. Besides this electronic vaulting capability, other important features of PLM systems include: workflow and process management, and product structure management. These features are often overlooked and comprise some of the more powerful tools offered by modern PLM systems.

### 2.3.1 Configuration

PLM system configurations are based on client-server technology; where all the documents reside on a central server and are controlled by software libraries and an underlying database. Individual workstations access and contribute data by running a client application that communicates with the central server through one or more networking protocols, such as TCP/IP, FTP, or SMB. Figure 5 displays a typical three-tier architecture where a database server communicates with a second tier, the application server, which in turn communicates with user workstations. The application servers often reside on the individual workstations as well, making it a typical two-tier client-server configuration much like a typical web server-browser configuration.

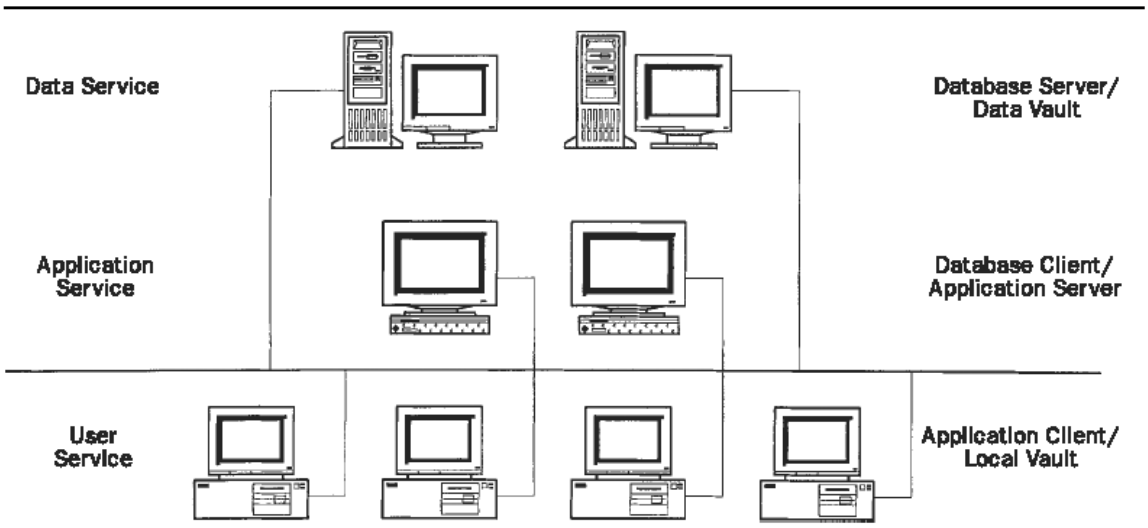


**Figure 5- Three-tier PLM architecture**

### 2.3.2 PLM Storage Architecture

PLM systems are made of two components—a relational database management system (RDBMS) and a file server (Figure 6**Error! Reference source not found.**). These components may be on separate machines or even spread across multiple systems. The database contains metadata about the files in the file system as well as data related to other PLM modules not discussed here. The application service is the component that allows both of these systems to be accessed simultaneously and work together in a sensible manner.

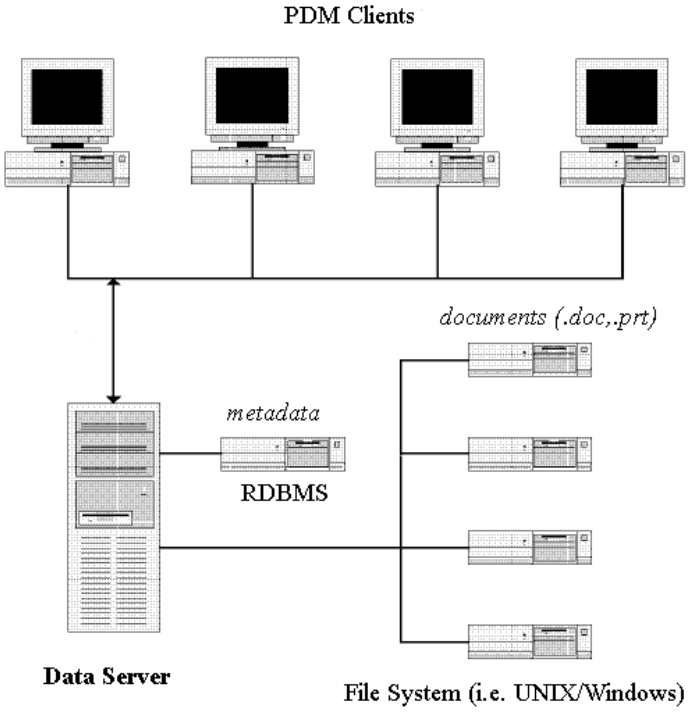


**Figure 6- Data storage on the server (adapted from Xu & Liu 2003 [1])**

The database server and data vault act together as an extremely advanced file system. In a typical file system, information such as owner, creation date, permissions, modification date, comments, keywords, and size is stored as metadata along with a document's real data.

In contrast, when a PLM system saves a file, the data service stores the file itself in a standard file system, but also stores additional metadata in a relational database management system. This additional metadata storage enables a PLM system to track virtually any information about the file a company would want. Some of this may include revision information, user ID, group ID, owner, additional permissions, release status, last modifying user, and notify-on-change users. Because of the nature of the underlying database and its ease of customization the possibilities are virtually without limit. Because of this, document management can be tailored for any situation, data can be distributed any way desirable, and it creates the ability to perform extremely fast searches and retrieval on related metadata.

#### 2.3.2.1 Relational Databases

Database management systems (DBMS) are highly evolved servers that provide and store data upon requests of clients. Careful attention has been and is given to their design to provide extremely fast and reliable access to data to multiple simultaneous clients. They have been relied on for years in data-critical industries such as banking and have proven their ability to be a solid starting point for collaborative environments.

Databases function as a series of tables analogous to spreadsheets. Each table has a defined number of columns that expands by adding rows. Columns are defined as types such as floats, doubles, strings, and integers much like variables in a C++ class. Each row in a table can then be thought of as an instance of the class.

Modern databases have evolved and have taken the name of relational database management systems (RDBMS). In these databases, often one of the columns will represent a relationship to a row in another table. This setup is similar to a pointer in a C++ class that

refers to an instance of another class. This ability of one instance to “point to” another allows a great deal of flexibility, including polymorphism and inheritance. These abilities will be key to allowing the proposed system to perform the tasks industry needs.

#### 2.3.2.2 Shared File Services

The file services component of the PLM systems is typically made up a samba-like (SMB) file server and an access controller. The file services may reside on any type of operating system, including UNIX, Linux, or Microsoft Windows ®. The access controller is a service residing on the file server machine that grants access to the PLM files only to clients connecting with a recognized application server. This is critical to ensure security and data control since all clients must be granted access to the file services and would otherwise be able to circumvent the security by using basic file sharing methods.

### 2.3.3 Additional PLM Components

PLM applications such as workflow and product structure management do not run the same as typical applications. All inputs and outputs are transactional—referring to how changes by a user are sent through the network to the PLM system where they are stored as data in the database; this is vital and will be referred to in later sections. Information such as view preferences and folder hierarchies are stored in database tables rather than in memory. Because the input and output in a PLM system differs from that of a conventional local file system, programs must either be PLM-wrapped or be custom-written to interface with the PLM data storage. A general lack of familiarity with this system and a lack of standards have resulted in slow development of PLM plug-ins and third-party software integration.

According to Chu and Fan, this customization can often be more time-intensive and arduous than the development of the PLM system itself [3].

# CHAPTER 3: LITERATURE REVIEW

In recent years, usage of parametrics has become widely available in CAD systems and other engineering tools. It has also moved progressively towards networked solutions as complex design and analysis often require distributed development. As such, data management has been a major consideration and authors have engineered many approaches to this problem. This chapter discusses the approaches and considerations discussed by peers in this field of study.

The relevant research t can be divided into the following topics:

- Data handling in automation
- Data handling for collaborative environments
- PLM variance management
- Important design considerations

## 3.1 Data Handling in Automation

A vital component to any automation is the handling of the data involved. The manner in which data is stored and communicated between programs defines the interface between the processes. The input of one process must be the deliverable of its

parent process(es). Because of this, the mechanisms for data transport and storage define limitations for the efficiency and capability of the system.

Database management systems have proven themselves ideal for a wide range of applications because of their performance and standardized interfaces. Their use as a central point for design automation is not new—it has been used for years as a solution to advanced engineering problems. It has only been in recent years, however, that RDBMS systems have been used in real engineering design automation. Several related projects will be reviewed and compared to the usage proposed by this thesis.

One of the most important steps when working with databases is the design of the data schema. Cochran and Hong did extensive work in designing a generic data structure for manufacturing simulations that would allow various classifications of experimental conditions to be defined in database tables [15]. Their approach focuses on a generic system of schema design using the Extended Semantic Hierarchy Model (SMH+) database modeling methodology of Brodie and Ridjanovic [17]. This concept of generic schema design becomes an important part of this thesis, as a schema must be designed to handle all variations of parametric CAD data. This is discussed further in the name-value portion of the method section..

Although RDBMS systems have been useful to various disciplines, including simulations as mentioned above, their history with CAD systems is limited and begs the question of plausibility. Leong et al. highly favored modern database management systems for use in distributed PLM environments and advanced CAD/CAM/CAE [2]. This supports the concept of enabling the RDBMS of the PLM system to be the leverage point for parametrics as well. Barbosa et al. also proposed that for ideal collaborative

design, the distributed CAD object model "…should allow teams to work cooperatively, accessing and changing information in distributed engineering data systems at run time." [16] This is a primary objective of this thesis—to provide the functionality of parametrics and knowledge-based engineering to a distributed audience requiring data concurrence.

One important area driving parametrics is the need to associate non-geometric with geometric data. This would be highly possible if data were stored generically in a RDBMS, as proposed. In order to overcome this classic problem of geometric/non-geometric linked data storage, Peng et al. developed a custom solution using databases for rapid spring design [18]. Their primary achievements included the ability to achieve simultaneous data manipulation and the ease of locating necessary data. Baker addressed the same issue in a generic manner to apply to any design situation needing non-geometric data [19]. He shows that applications such as thermal analysis can use data defined from within the CAD system. This greatly reduces the amount of effort in design changes and provides a viable solution to design/analysis automation.

Bowland et al. [20] created a tool that uses the database of a PLM system to define assemblies for powerful features not offered by the CAD system. Using the schema depicted in Figure 7, assembly definitions were created and defined entirely external of the CAD system in the PLM database tables. This proves that the PLM system can contain additional data not provided by the CAD system in order to achieve custom or extended functionality, a vital assumption in the formation of this thesis.

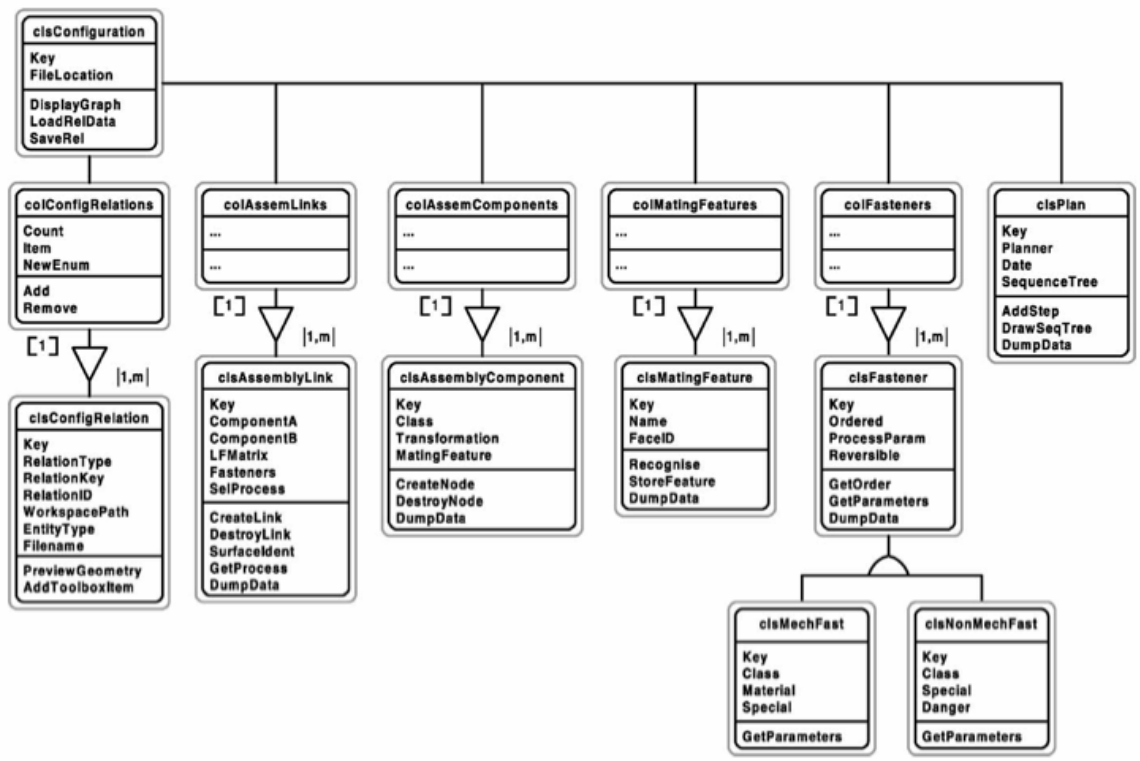


**Figure 7- Assembly definition data schema**

The powerful external assembling shown in Figure 7 allows customizable assembly operations such as extremely rapid part freedom assessment. Because the assembly data is modularized, any component's local freedom can be derived with respect to each of its related components without loading the entire data model for the assembly.

. Besides rapid freedom assessment, other benefits of basing assembly definition in databases rather that CAD models include automation of upstream and downstream tasks, simplified planning, and added modularity. Bowman et al. describes all of these as factors currently limiting computer-aided assembly process planning (CAAPP) [20].

## 3.2 Data Handling for Collaborative Environments

As data communication has become increasingly important to engineering, numerous technologies have emerged to attempt to satisfy these needs. These technologies address the issues of viewing design data from multiple programs and formats, distributing it across networks, storing relevant knowledge, and the search and reuse of existing company knowledge. One manager stated: “In today’s distributed manufacturing environment, how to manage the product data and distributing it to the right people is critical to the success of a company. [5]” The demands for reliable and secure data sharing have sparked a frenzy for new methods, tools, and standards in product lifecycle management and design collaboration. Many of these new technologies overlap in many areas and there is some disagreement as to which technology is appropriate for individual tasks. A discussion will be given to the most relevant of these frameworks along with recent research in each area.

### 3.2.1 Web Frameworks

With the increasing count of web developers and web-based tools, many have found ways to implement PLM functionalities with common web software found on most computers. These systems evolve fast and are known for rapid deployment. They are very important to this research because they are often the first place new concepts emerge, and their best practices can be leveraged into the proposed system. The main needs of web technology in this thesis include:

- Global capability is mandatory to follow global product development trends

- Clients must be able to interact with and modify PLM objects, therefore, data must be accessible and editable at client stations
- The architecture should support this functionality natively in order to standardize interfaces and prevent excess customization requirements

This section searches out technologies that can be used to achieve these objectives.

In *WebBlow: A Web/agent-based multidisciplinary design optimization* [6] Wang et al. created a design optimization environment using a series of agents. Their data transport mechanism was XML over the Internet. This is important to this research because the same methods to read and interpret XML can be used with relational database models. In fact, they stated that the use of an RDBMS system could have achieved higher performance. An important statement of theirs is that "environments should not only automate individual tasks, but should also allow members to 'share information and knowledge', 'collaborate' and 'coordinate' their activities within the context of a design project." Their approach is impressive and accomplishes the objectives notably, while remaining a typical example of custom, case-specific software. Each component is independently developed and leverages very little existing work. This approach of starting from scratch is simply not feasible for many companies that require solid security, reliability, access control, and a host of other features to be in their framework. The work in this thesis attempts to use the existing technology in PLM systems to accomplish many of these same objectives.

Some collaborative systems have emerged strictly as hypertext (HTTP) systems. Liu and Xu suggested that a web browser should be used as the ultimate client for PLM

systems because of its standardized and globalized nature [1]. Web server configurations already closely match the tiered configurations of PLM systems.

Xu and Liu, in a later article, describe the implementation of such a configuration in the conference address, “Web-Enabled PLM System in a Collaborative Design Environment” [1]. Their system is fully custom and developed with six main technologies, namely: Common object request broker architecture (CORBA), common gateway interface (CGI), ActiveX, standard for the exchange of the product model data (STEP), VRML, and object-oriented database management system (OODBMS). Xu and Liu essentially re-created a PLM system with an ActiveX-enabled web browser front end. The purpose of this was to enable PLM collaboration without solely relying on a local network. They state that “a stand-alone, non-Web-based PLM system is barely sufficient for supporting and managing various stages of product development in an organization, mainly due to its lack of much needed global communication infrastructure, which normally leads to the ineffectiveness of design intent exchange.” These tools shipped (and still ship) with a JAVA front-end vs. the Microsoft-dependent ActiveX and perform all the functions named by Xu and Liu. The architecture of their system demonstrates how CAD systems might interface with the PLM system, which is of particular interest to this thesis in order to define data relationships (Figure 8).

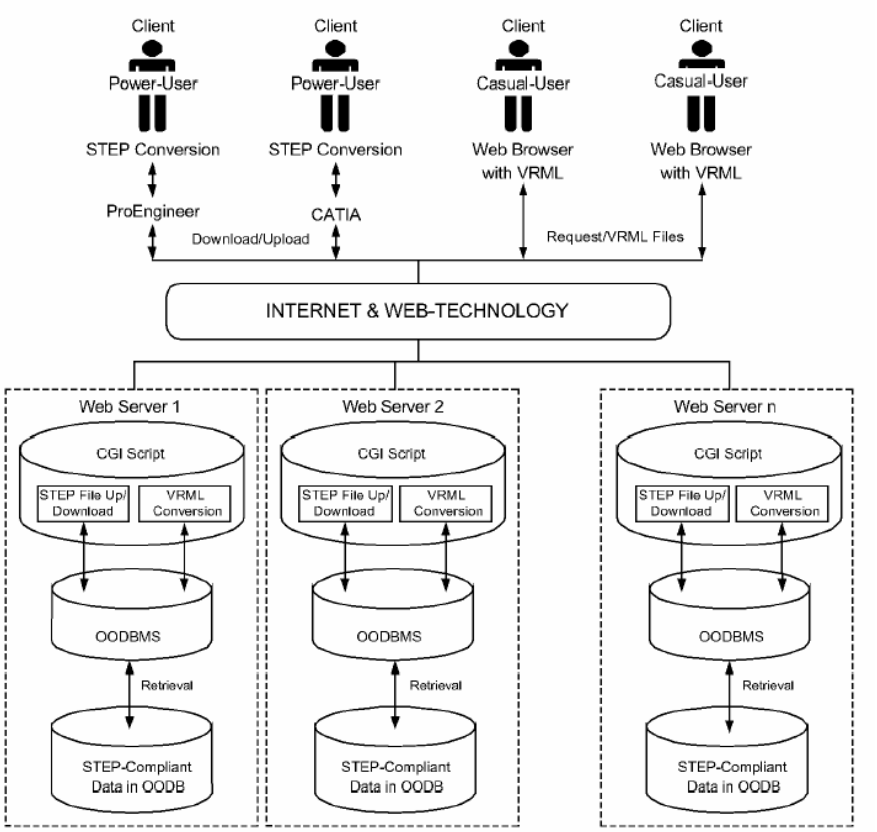


**Figure 8- Xu and Liu's web-based PLM architecture**

Chu and Fan also argue that PLM system functionality can be built using web technology [3]. This makes the system faster to implement and has a more comfortable feel. They also stated that there can be a combination of the two using components, namely databases and web technology, through the use of Data Access Objects (DAO). They state that browser plug-ins can essentially be the applications running on the remote computers. They suggest that this can be accomplished with ActiveX technology (by Microsoft) or Java. Their comparison of the two architectures can be found in Figure 9.

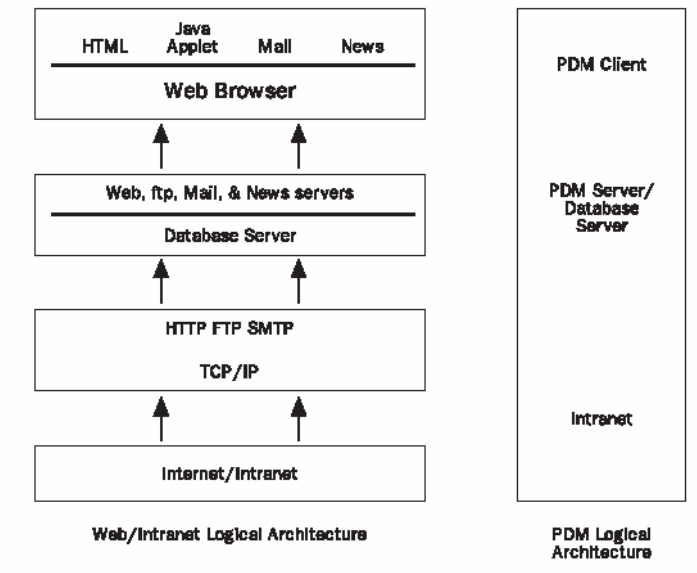


**Figure 9- Comparison of web and PLM architectures**

It has been observed that many of Chu and Fan's concepts (such as JAVA clients) are in use in commercial PLM systems such as Teamcenter Engineering and Dassault Systemes's Enovia. This is one of the stronger points of these PLM systems because it enables them to run on most operating systems, just as Chu and Fan advocate. However, the notion that the web is a good foundation for PLM deployment, especially when used to gain faster deployment, can certainly have drawbacks. Any use of ActiveX would also limit use to Microsoft operating systems. In the case of the named commercial PLM systems, a web browser interface is an additional module rather than a foundational component. For example UGS has created an optional *Web Access* PLM component that has similar appearance and many of the functionalities of their "workspace" tool. It is described as follows:

> Web Access provides a streamlined browser-based view of product information stored in a Teamcenter Engineering database.

> Just as with Workspace [Teamcenter's standard PLM client tool], graphical icons represent various types of product information. However, instead of using a "tree style" representation (as found in normal Teamcenter Engineering Workspace), product information and relationships are represented as a sequential series of HTML pages. Web Access pages contain menus for accessing various features (e.g., InBox, Search, Options, etc.) and links to other objects (e.g., folders, Items, datasets, BOMs, etc.). [3]

The notion of simple web access to PLM data and applications is an extremely powerful notion because it does not require any software on the client other than a standard web browser. It is made possible because of the transactional nature of the system I/O as stated earlier. This is very attractive in the situations many companies face where B2B networking is required. It is limited, however, to data viewing, downloading (checking-out), and uploading (checking-in); it does not support integrated CAD environments and some other niceties of the Java thin client. This is highly attributable to common security features on the internet. However, the internet is not limited to web browsers and FTP, and in fact, modern PLM relies heavily upon web transfer protocols such as SMB, SSL, and CORBA. Therefore, a good understanding of these relationships is useful to this thesis.

Carroll and Hawkins [21] created a system to perform analysis runs on multiple computers using a three-tier web system. They used XML as the definition mechanism, which is highly similar to relational databases and thus of use to this thesis. XML can be defined in databases and often database queries can be configured to return results in XML. Table of contents, geometry data, and FEA data were the three categories of the XML schema used (abbreviated versions shown below).

**Table of Contents Node**

```
<TOC>
<TOCITEM name="Asm1" id="1"  type="assembly"/>
```

```
<TOCITEM name="Part1" id="2" type="part" />
<TOCITEM name="Part2" id="3" type="part" />
<TOCITEM name="Mesh1" id="4" type="mesh" />
<TOCITEM name="Mesh2" id="5" type="mesh" />
<TOCITEM name="Displacements1" id="6"  type="nodalresults" />
<TOCITEM name="Displacements2" id="7"  type="nodalresults" />
<TOCITEM name="Param1" id="8"  type="parameter" />
<TOCITEM name="Param2" id="9"  type="parameter" />
</TOC>
```

**Geometric Data**

```
<CAEDOC>
        <ASSEMBLY id="1">
                <PART id=2>
                        <BREPID/>
                        <XFORM/>
                        <NAME/>
                        <BOUNDINGBOX/>
                </PART>
                <PART id=3>
                </PART>
                <BREP id="id1">
                …
                </BREP>
                <BREP id="id1">
                …
                </BREP>
        </ASSEMBLY>
</CAEDOC>
FEA Data

<CAEDOC>
        <FEAMODEL>
                <TABLES>
                        <ELEMENTTYPE ID="1" topology="brick" order="2" …/>
                </TABLES>
                <MESH>
                        <NODE id="1">
                        <NODE id="2"/>
                        <ELEMENT id="1" type="1"/>
                        <ELEMENT id="2" type="1"/>
                </MESH>
        </FEAMODEL>
</CAEDOC>
<CAEDOC>
        <FEARESULTS>
                <NODALRESULTS lab="uxyz" />
        </FEARESULTS>
</CAEDOC>
```

Their data schemas are conceptually very close to what is to be used in the PLM database for this research. This thesis research expects to see the same advantageous properties of Carroll and Hawkins's implementation, namely:

> Asynchronous - When submitting a simulation for execution, the system does not require that the client remain connected throughout the duration of the simulation.

Multi-user - Simultaneous client access is synchronized via the web-server.

Robust – Failure of network resources or servers results in a job being rescheduled at a later date when the system is back up. No loss of data will occur. Catalog updates are transactional in nature at the web server level.

Extensible – The definition of a 'job' is general enough that the system can be used to host a wide range of batch-style engineering computations. Indeed, our implementation can be viewed as a layered architecture. The base layer is the infrastructure to support generic application and execution and synchronization. The next layer is engineering application specific. The definition of 'data location' is also general. It is a Uniform Resource Locator (URL), and the actual storage location of any server-accessible data can be on a local file, in a database, on another machine, or even generated on-demand via another web-server.

While their research is very similar to that of this thesis, it differs dramatically in the environment where it resides. It is proposed in this thesis to use these techniques in a PLM system where there is much less experience when compared to web programming. However, it should be well worth the effort due to the benefits of tighter collaboration with CAD systems, ERP systems, and bills of materials, workflow, and all the company operations already in PLM systems will be of great importance to industry.

### 3.2.2 Database Frameworks

Baker [19] created a system of storing CAD file metadata in an external RDBMS, much as proposed in this thesis. He was able to flag specific geometries in the CAD system with extra attributes such as mesh density, loads, and other parameters needed by downstream applications. Analysis software (ANSYS) was then able to load the CAD part, read the extra attributes from the RDBMS, and then use them for analysis purposes.

He found that this enabled the system to use a CAD-centric approach, which then created persistency with downstream attributes that normally have to be regenerated upon iteration. The performance of his system was notable and the system was easy to use.

His research is the closest to that of this thesis, and serves as a substantial foundation for this research. He has proved that the concept is viable on a smaller scale, and the notable difference in this thesis is that system proposed in this thesis on a much broader scale. It includes incorporation into PLM systems, and extends to all engineering tools and models. It extends the purpose from just inter-process communication to inter-departmental and inter-organizational collaboration.

## 3.3 PLM Variance Management

PLM customization is important to this research because it is the vehicle by which parametrics will be put into PLM systems. PLM systems are usually highly customizable through the use of Application Programming Interfaces (APIs). This is a common requirement due to the many varying ways companies operate [4]. Knowledge-based engineering can be integrated into PLM systems through customization; however, industry's current solutions are mostly case-specific. Programming is challenging and starts with a steep learning curve. Most programmers are accustomed to writing files or storing things in memory and have to learn how to change their way of thinking to a transactional nature as spoken of by Carroll and Hawkins [21].

Oh et al. [22] presented a good example of knowledge integration into PLM systems using the unified modeling language (UML). The format of the data exchange is STEP and an interface was provided to import the data into Metaphase (Figure 10).

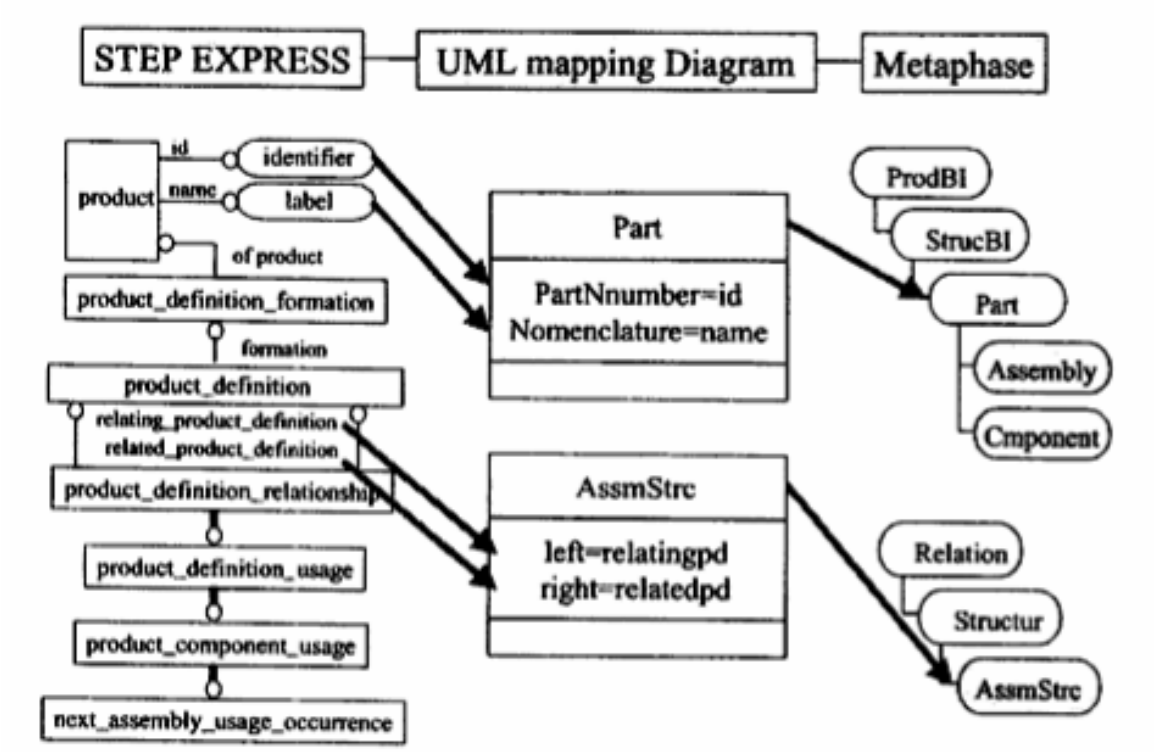


**Figure 10- STEP data mapping to PLM system**

One of the major drawbacks of using PLM systems is the performance effects of heavy network traffic. One large aerospace firm nearly abandoned PLM after a major launch effort because designers were frustrated by its poor speed. This speed issue has made the data distribution model and related performance measurements major PLM design considerations when dealing with multi-site systems. Leong and Lee [2] determined four main objectives for implementing an efficiently distributed PLM system:

- Minimize remote object request rate
- Minimize DPLM system retrieval and update operation cost
- Minimize DPLM system response time

- Reduce storage cost

Leong and Lee also quantified the operation cost (consisting of local retrieval, local update, remote retrieval, remote update, and storage cost) as follows:

$$\sum_{i=1}^{m}(x_{ij}^{a} r_{ij} C_{local} + x_{ij}^{a} s_{ij}) = \sum_{i=1}^{m}[x_{ij}^{a}(r_{ij} C_{local} + u_{ij} C_{local} + s_{ij})] \quad \textbf{(1)}$$

where *i* represents a data object being retrieved or updated, $x_{ij}^{a} r_{ij} C_{local}$ represents the local retrieval cost, $x_{ij}^{a} u_{ij} C_{local}$ represents the local update cost, and the local storage cost of object *i* in site *j* is represented by $x_{ij}^{a} s_{ij}$. Similarly, when an object *i* is updated via sites *j* and *k*, the remote retrieval, update and storage cost is:

$$\sum_{i=1}^{m}\left[\sum_{k\neq j,k=1}^{n}\left(x_{ijk}^{r} r_{ij} C_{remote} + x_{ijk}^{4} u_{ij} C_{remote} + x_{ijk}^{4} s_{ik}\right)\right] = \sum_{i=1}^{m}\left[\sum_{k\neq j,k=1}^{n} x_{ijk}^{r}\left(r_{ij} C_{remote} + u_{ij} C_{remote} + s_{ik}\right)\right] ($$

**2)**

Their research shows that by lowering the values of these variables the bandwidth requirements (cost) decreases exponentially. This heavily supports the idea of this thesis because the transfer of the parameter data rather than the full documents reduces the load from around one thousand to one.

## 3.4 Literature Conclusions

The literature has presented valuable research and addressed many questions that were needed by this thesis. They were able to establish the validity of the use of database for the storage of CAD parameters. Architectures have been proven that will support the

needs of such a system, making the concept viable. Finally, based on the results presented by others, the concept presented in this thesis should be able to provide the desired improvements in standardization, modularization, performance while still providing tighter collaboration and globalized automation capability.

# CHAPTER 4: METHOD

This thesis proposes a methodology for storing model parametrics in the relational database component of PLM system in order to facilitate various aspects of product development. This chapter discusses the steps involved in creating this type of system.

## 4.1 Objectives

The objective is to store the parameters used to control models in relational database tables of the PLM system in order to improve the following high-level ideals:

- Decreased time to market through higher efficiency and performance
- Improved quality through repeatable, standardized processes that can be used for product optimization
- Enhanced globalization

Some specific objectives are targeted. This methodology should be able to provide:

- An open (non-proprietary) storage of key design parameters
- Automation-friendly data storage and access
- Increased PLM system performance
- Standardization of data and communication
- Decreased development costs

## 4.2 Generate and Standardize Reusable Objects

The first element of the system is the existence of parametric models. These can be any reusable objects such as spreadsheets, CAD part files, meshing scripts, and optimization configurations. These objects should be standardized and version-controlled, and thereafter treated as a complete class or application much like a compiled software program. These objects or applications then become process "black boxes" with predefined inputs and outputs. Each black box has its own set of parameters and methods. The parameters are the data that will be passed in and out of the system and the methods are the operations that this particular model will be able to perform.

### 4.2.1 Parameters

For a given a set of parameters, a version of a parametric solid model will generate a version, or instance, of a design (Figure 11). Each design's parameters are stored in PLM system's database tables instead of storing an entirely new full part file. Multiple designs can be compared quickly by viewing design parameters using database queries rather than opening multiple models.

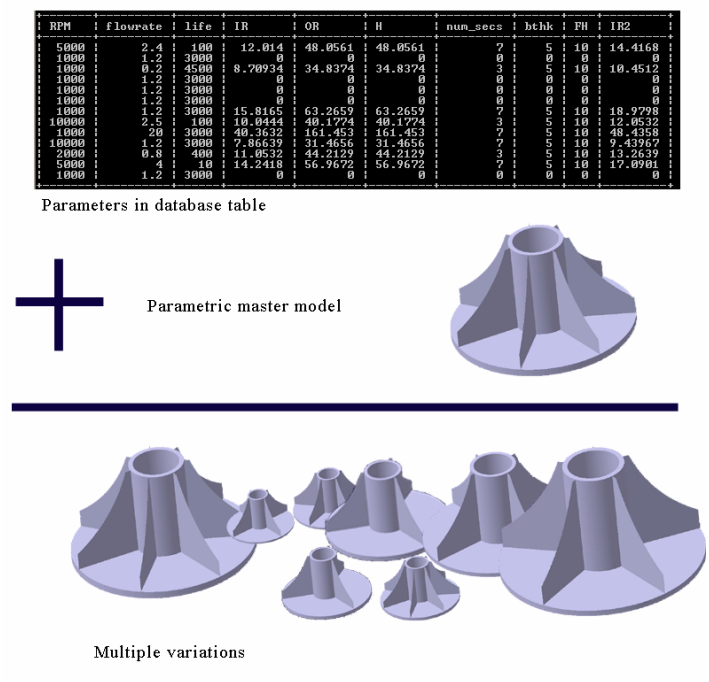


**Figure 11- Database instantiation**

In Figure 11, the single master model along with a series of database-housed design parameters can be used to reproduce multiple model instances. This requires only the smallest possible amount of information to represent an entire part's family. This is the concept of the proposed methodology and a general view of the use of this approach shows that the objectives will be satisfied.

The parameters must be extracted from the models in order to accomplish the database storage. RDBMS systems define storage columns as generic types such as integers, floats, doubles, strings, etc. By adhering to these types of fields, any program is able to retrieve any values, regardless of the associated model. This allows data sharing and inheritance—important features for concurrent development.

For example, if the driving parameters of a spreadsheet also refer to the dimensions of a CAD model, these parameters can be stored in a single object to be retrieved by both

models. Similarly, a single model may draw upon multiple datasets for its parameter set. This can be leveraged to provide inheritance in an object-oriented fashion—downstream models can simply use the output parameters of upstream models as their input.

This open standard also allows B2B sharing of data without distributing proprietary knowledge. Companies may publish the input and output parameters of a program or model without the need to reveal the original model.

CAD system interoperability is an example of the potential use of this type of product. If two master models are created with the same parameterizations, they can both be used to represent the same set of models. The only need inherit the same parameter set and they will be the same model. This could be useful when third-parties require data in specific formats.

### 4.2.2 Methods

The various tasks a model may perform should also be evaluated. Most models will have only one task, but some will have the capability to perform multiple. In the example of the CAD model, it is assumed that a reshaped version of the model is desired, and the output will be the new geometry. However, the output could be a mass and centroidal analysis instead. It could also be an IGES file or any combination of those mentioned. These different uses are referred to as the object's methods, and they should be determined before continuing into further stages of the system design.

## 4.3 Design the Database Schema

A data model is then created inside the PLM system that can represent the parameters for each of these objects. Depending on the PLM system, this can be done is various ways, but in order to accomplish the objectives, it must be assured that the schema will provide parameter storage inside the database portion of the PLM system as opposed to the file system.

The design of a good data storage schema is critical to accomplishing the goals of the system. It determines the performance and flexibility limitations as well as the complexity of accessing the data. Careful consideration of not only existing parameters, but potential parameters should be designed into the schema as well.

### 4.3.1 Schema Design Considerations

A good schema design would satisfy all possible uses of the data involved. In large scale databases, which most PLM systems are, a good schema is critical to achieving acceptable performance, functionality, and flexibility. In the long run, the performance of the PLM system will be greatly influenced by the design of the schema. Some PLM systems have been abandoned entirely because their performance seems to users to inhibit their productivity. It is important that data can not only be stored quickly, but searched and retrieved quickly as well. Summary and reporting requirements are the most demanding of interactions as they require accessing the data contained in many objects at the same time.

In order to design the system to be catered to the needs of KBE, the type of storage used in the database should imitate as nearly as possible the way parameters are stored in the programs needing integration. This will minimize workarounds and improve the accuracy of the data transmission and storage if translation can be bypassed. Fortunately, most engineering models follow a common parameter method; namely, there is a unique variable name that corresponds to each parameter.

### 4.3.2 Functionality

The data should be able to be accessed interactively as well as programmatically. In order to achieve this, the PLM system's methods should be inspected to determine the available interactivity methods. If data is accessed through the PLM system's API, it should be inherent. If not, portal server and client programming will be required to display data and their properties. Interactivity will provide most of the usage in the cases of distributed access. Although interactive access is not entirely required, if not present, it would severely limit the usefulness of the system.

### 4.3.3 Schema Design Options

For most engineering applications, the schema design is fairly simple—it must be able to store a variable number of name-value pairs. To satisfy this requirement, this research has found three main options as table schema types:

1. **Field-named**—each variable to be used is pre-defined in the database table
2. **Name-value**—each object contains two variable length arrays (VLA), one for the names and the other for the associated values

3. **Name-value-ref**—a spin off of the previous method, name-value-ref also contains a VLA for both the names and the values, but an additional VLA of unspecified-type pointers allows the values to contain different object types.

The first two types were the first to be experimented with and the third was invented out of necessity during the prototype stage. Each has its advantages and disadvantages, so a discussion of each follows.

#### 4.3.3.1 Field-Named

The field-named storage type is the most familiar to database users. It is the most efficient, but is also the most rigid. With this type, a table is created that identifies each variable column by name (Figure 12).

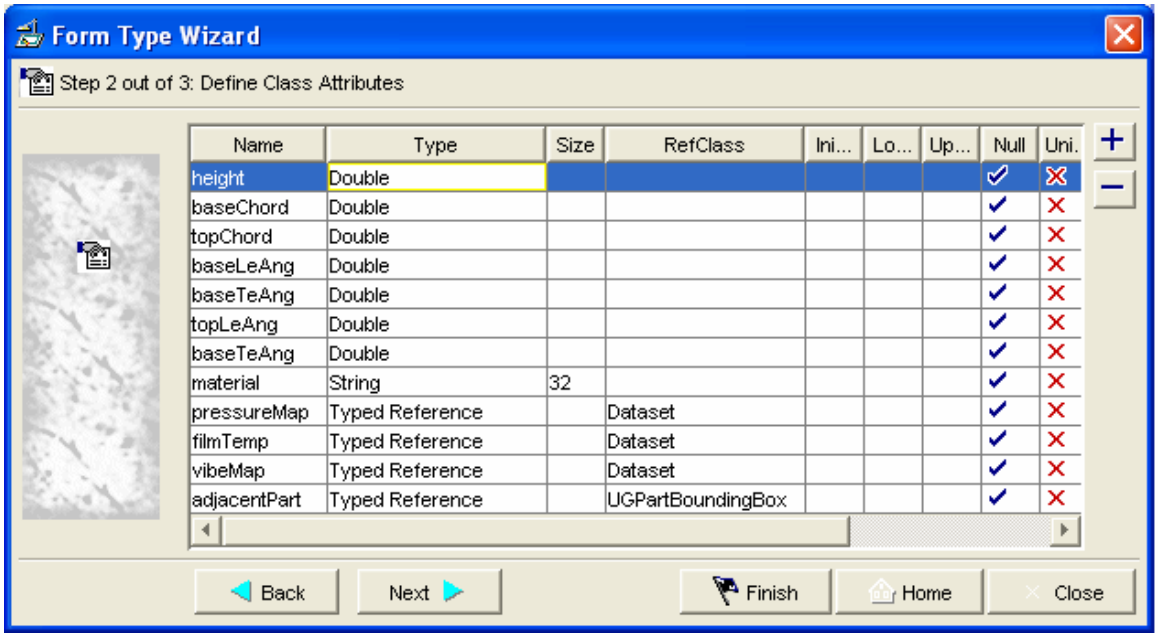


**Figure 12- Field-named table design**

Field-named tables present various advantages. They are very efficient when retrieving and storing information. They are especially valuable when searches and summarization is required. Because the database itself knows exactly where each value is stored, it is able to perform searches based on the value of one of the fields. For example, if an optimization were performed with the data being stored in a field-named table, after the optimization was finished, a single query could return the Pareto front nearly instantly (Figure 13).

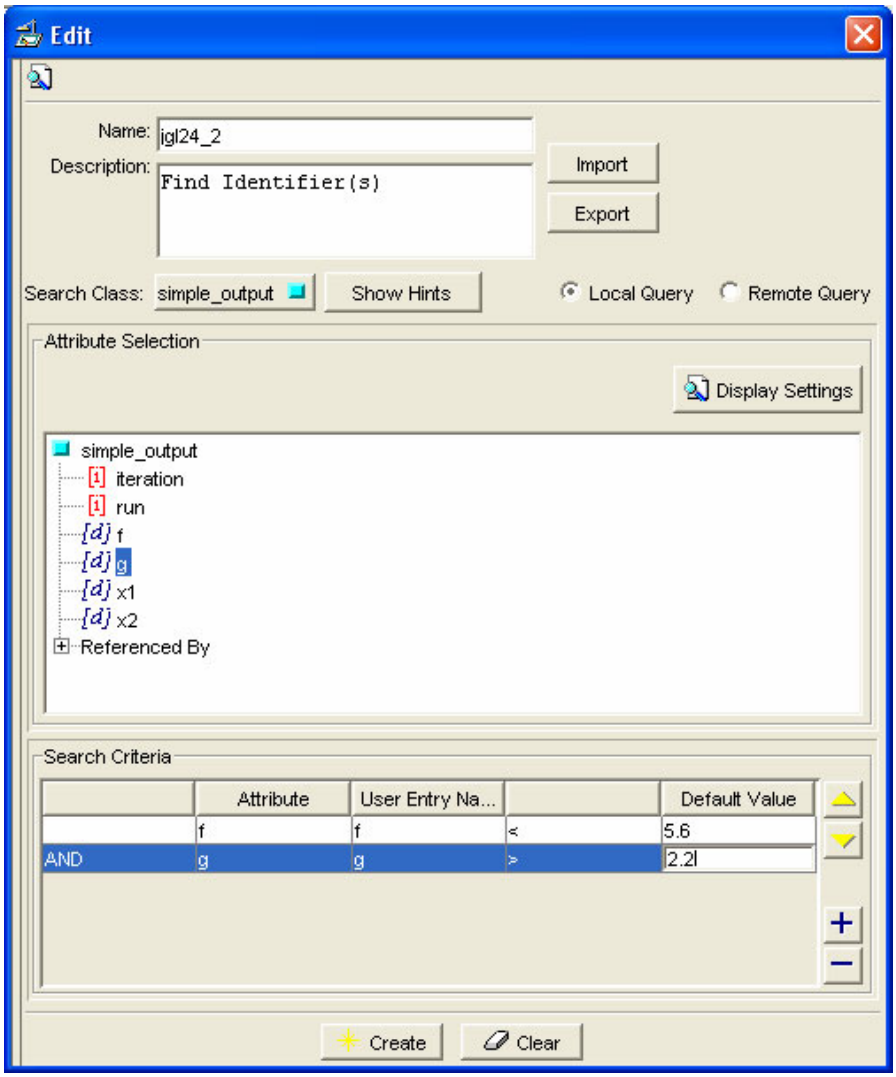


**Figure 13- Query for objects meeting specific criteria**

Another advantage to a field-named schema is standardization. As shown in Figure 12 it is required that the value type be specified. This can be selected from a number of options, including integer, boolean, string, reference, arrays of each of these, and more. This forces values to be stored exactly as prescribed, which also reduces confusion and mishaps. The precision of doubles is maintained and the exactness of integers remains intact as well. It also forces business units to determine exactly what their formats will be and then to abide by their decisions.

Field-named tables can be useful when attempting to extract information from models. Often, a model contains much more information than needed to pass on to downstream applications. With Excel, for example, the spreadsheet would typically be opened and a few select cells, summarizing the results of the computation would be the only values of interest to other processes. Some type of indication of which cells these are must be known to the user or program performing the operation and this is one case where field-named tables are useful. The names of the fields themselves can be the variables expected to be found in the spreadsheet. The desired values can then be retrieved in a loop such as:

```
name = iargs->getColumnName(i);
while(name[0] !=  ' ' && name[0] != '\0') {
        VARIANT a = iargs->getVariant(i);
        excel.putCellValue(name,a);
        VariantClear(&a);
        delete [] name;
        i++;
        name = iargs-> getColumnName (i);
}
```

While standardization improves accuracy and confusion, it is also a double-edged sword. It freezes simulation designs by requiring administrative access to make modifications to the table. On production databases, administrative access is closely held

among very few because of the damage one could cause. Also, after a table contains data, it is very dangerous to the health of the system to attempt to make modifications—if possible at all. Also, most engineering tools allow for a varying number of design variables, which obviously does not fit well with the concept of a field-named table design.

Another disadvantage is that companies may be using hundreds of reusable processes and models. If they are to be modularized and stored parametrically this method would require a huge amount of modification to the database structure—one table for every parametric object type. This also says nothing of the additional tables required to handle inevitable variations to schema designs.

#### 4.3.3.2 Name-Value

The Name-value is called such because it consists of (at least) two fields- one is an array of the variable names and the other is an array of the variable values. This presents an opposite set of advantages and disadvantages from the field-named alternative. The main advantage is its flexibility—the variable names can be changed without administrative privileges. This allows programs to use a variable number of variable names into a form at runtime. It also allows users to modify their own variable scheme, which is crucial in most product development simulations.

A major drawback of this type of storage is that summarization queries cannot be executed by the database alone. Queries can still be made to locate objects that contain specific variables and numbers, but it is not possible to construct a query with standard SQL to tell the database to search for an object with a specific value who's index

corresponds to the index of a specific variable in a sister field, or in other words, to search for a name-value-stored value. Another potential drawback is the number of the values that could be stored in the variable-length array (VLA) table. This is a problem will be discussed in the test case section.

Although the name-value has some serious drawbacks in the searching and summarization category, the importance of non-administrative control dominates the more structured field-named approach.

Although the name-value storage type is favored, there remains one more limitation—the lack of type specification for the values. The type must be specified as a string to be able to store integers, floats, and strings with the same data type. As such, the length of the string must be decided upon install and then adhered to with all further usage. This creates both limitation and surplus in storage lengths—integers will only need a few bytes to represent their values, but complex equations demand several hundred. Because of these limitations, the name-value-ref schema type was created.

#### 4.3.3.3 Name-Value-Ref

The name-value-ref schema is essentially the same as the name-value type but with an additional VLA that contains “untyped references”. The untyped reference enables a variable to refer to any other table in the database. This is analogous to a struct that contains a name, value and an untyped pointer:

```
struct name-value-ref {
       string name
       string value
       void* ref
};
```

Once created in Teamcenter, each field corresponds to one another by their index. If a reference is used, the value is simply left blank and vice versa.

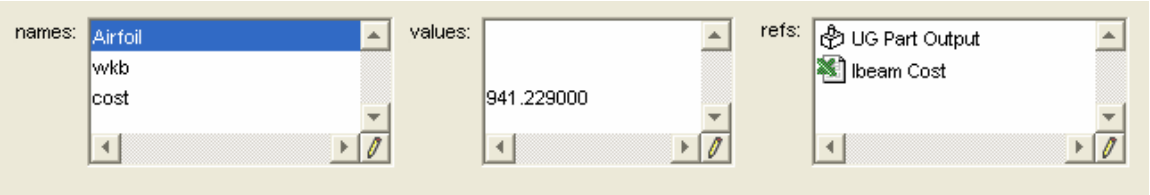


**Figure 14- Name-value-ref in Teamcenter Engineering**

Figure 14 shows three variables stored in a single form. Two of which actually refer to files and the third refers to a number stored in the value field. The object named "UG Part Output" corresponds to the variable "Airfoil" because they are both on an index of 0, or they are the first in the list.

Once a suitable schema has been determined, it should be inserted into the PLM system via the PLM system's own methods. For some, it will be easy-to-use graphical user interfaces (GUIs), for others, it may be Structured Query Language (SQL) statements that should only be done by experts.

## 4.4 Determine the Data Access Method

With an analysis of the storage requirements in place, the methodology of how data will be accessed and stored is necessary. The data should be able to be accessed interactively as well as programmatically. In order to achieve this, the PLM system's methods should be inspected to determine the available interactivity methods. If data is accessed through the PLM system's API, it should be inherent. If not, portal server and client programming will be required to display data and their properties. Interactive

access will provide most of the usage in the cases of distributed access. Although interactive access is not entirely required, if not present, it would severely limit the usefulness of the system.

There are two methods by which the data can be programmatically accessed—through the API of the PLM system or that of the RDBMS. This is one of the major questions this research addresses. Which of these is used is an important decision because it affects nearly every aspect of the implementation, from security to accessibility.

In order to determine which is desired, the following factors should be considered:

- Security
- Data integrity
- Speed
- Simplicity

The implications of both methods are discussed further in the results section.

## 4.5 Determine Implementation Categories for the Model Types

Each model type will usually require its own development and/or configuration. Fortunately, model categories can be fairly broad, thus enabling generic method development. Depending on the nature of model, the required amount of development can vary significantly.

Some models will be able to be run from within the PLM system and others will be required to run externally. How the program will run and access its data determines how

it will be developed. The research determined three main classifications of how the programs may run:

- Internally
- Externally without database access (typical for legacy programs)
- Externally with database access

Other models will not be programs at all and may only be human interaction. PLM systems generally provide input forms and other methods to collect input from users. Although this is not the focus of thesis, this capability to have human interaction is vital to the concept, as very few processes can be standardized enough to have complete hands-off automation. Although these types of models can be either of the three categories, they will typically be internal functions. In fact, most PLM systems already provide many of these for use within workflow modules.

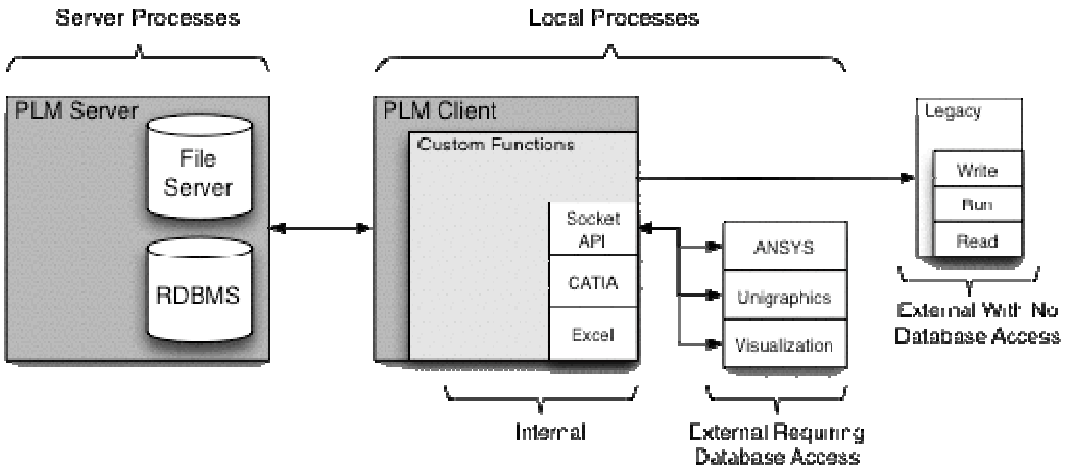


**Figure 15- Method execution location**

Figure 15 is a diagram of the area where each of the function categories execute along with the categorization of some common tools. In the case of computer programs, if the functionality is encapsulated in an executable file other than the PLM system's it

will be external with the exception of those capable of external automation. Excel and CATIA are examples of this because they can be controlled using Common Object Model (COM) interfaces.

### 4.5.1 Internal Functions

Internal functions are those object methods that can be executed from within *and by* the PLM system itself. If this is the case, the implementation must be done with the PLM system's API in order to embed the new functionality directly into the system. This is typically accomplished with the C or C++ programming language and the added functions (usually called "user functions") get packaged into a .dll. The specifics of the implementation and use will vary among PLM systems. The next chapter includes the implementation within a specific PLM system.

### 4.5.2 External Functions Without Database Access

For external programs that do not require database access, the implementation may already be complete, and the model needs to only be set up within the PLM as a standard process. If the program takes command line arguments or reads control files, the execution can be set up within the GUI of the PLM system itself, and the runtime files only need to be placed in a location accessible by the PLM system.

This category typically comprises legacy programs. They will be the simplest for companies to use and those most readily available. This will include FORTRAN programs and standalone executables. It can also be manual processes that only require someone to sign off on a process after he as seen specific files.

Although their simplicity is advantageous, their lack of database access defeats many of the ideals set up in this thesis. Again, examples of these will be provided in the next chapter.

### 4.5.3 External Functions Requiring Database Access

Externally executed programs are those that need to be their own separate processes because of any reason cannot be owned by the PLM system. These programs are generally those that require special runtime environments such as JAVA and software packages with complicated licensing or interfaces. The results section discusses some specific applications and their problems with internal execution.

In order to provide these programs access to the database, an internal function can be written that acts as an intermediate data source to the external program. This can be accomplished by acting as a server and providing data to the external applications using sockets. Details on this approach can be found in the implementation chapter.

## 4.6 Implement Methods for each Model Type

Once the model parameters and methods have been determined, the schema has been designed and put in place, and the method of data access has been elected, the implementation of each model's methods is the final step toward a fully-integrated system. The implementation details will depend upon the specifics of the models and the PLM system in use.

# CHAPTER 5: PROTOTYPE IMPLEMENTATION

This chapter reviews the specifics of the implementation of a prototype following the method outlined previously. The prototype was implemented with Teamcenter Engineering as the PLM system and Oracle 9i as the underlying RDBMS. Models included a spreadsheet, a CAD model, and an analysis script. The prototype system was developed to test the distinguishing feature of the methodology: store model parametrics in a PLM system's RDBMS.

## 5.1 Analysis of KBE Data Requirements

A name-value string pair was found to be a common storage method among most KBE tools. Four major tools were examined to determine variant data storage strategies. The specific tools examined include UGS Unigraphics, Microsoft Excel, Dassault Systemes CATIA, and ANSYS.

Unigraphics stores parameters as expressions, and more recently as variables in *KnowledgeFusion* (Unigraphics proprietary scripting language) scripts. The most predominant is the expression as nearly every parametric operation performed generates expressions by which the operation can be modified. An expression list is essentially a simple name-value pair, seemingly stored as text. Expressions themselves can contain some degree of knowledge as they can contain not just numerical values, but equations

that return a numerical value based on other expressions or other environment information such as geometric relationships. Figure 16 shows the expression list for the expressions driving a turbine airfoil and attachment.

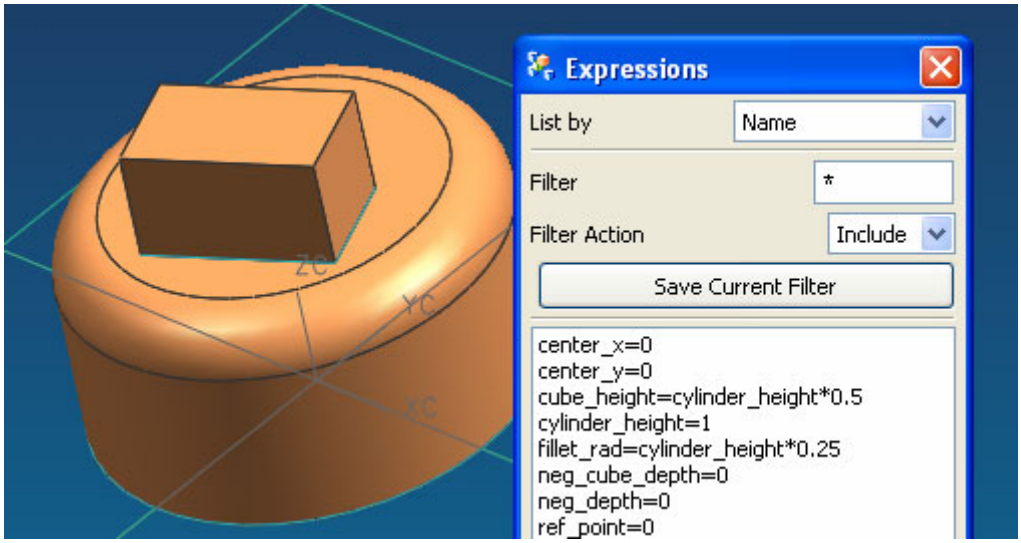


**Figure 16- Parameter storage in UGS Unigraphics (name-value method)**

Expressions can also be exported and saved in a standard ASCII text file (with the extension ".exp"), with the following format:

```
center_x=0
center_y=0
cube_height=cylinder_height*0.5
cylinder_height=1
fillet_rad=cylinder_height*0.25
neg_cube_depth=0
neg_depth=0
ref_point=0
```

The second KBE tool examined is the widely used spreadsheet tool, Microsoft Excel. Excel stores its values in various ways, but the values themselves are handled as VARIANTs, which are an abstract variable type containing either an integer, a double, a float, a string, arrays, or one of a few other types. Referencing these values in Excel happens in various ways; the first is by the sheet grid itself, with each cell name being the combination of a column and row name, vis. "A3". While this can be useful and certainly

functional, engineering automators typically employ the second referencing mechanism discussed here, which is that of named cells (Figure 17). The way data is referenced in Excel is by name-value pairs as well, again, with names associated with strings that contain either equations or values.

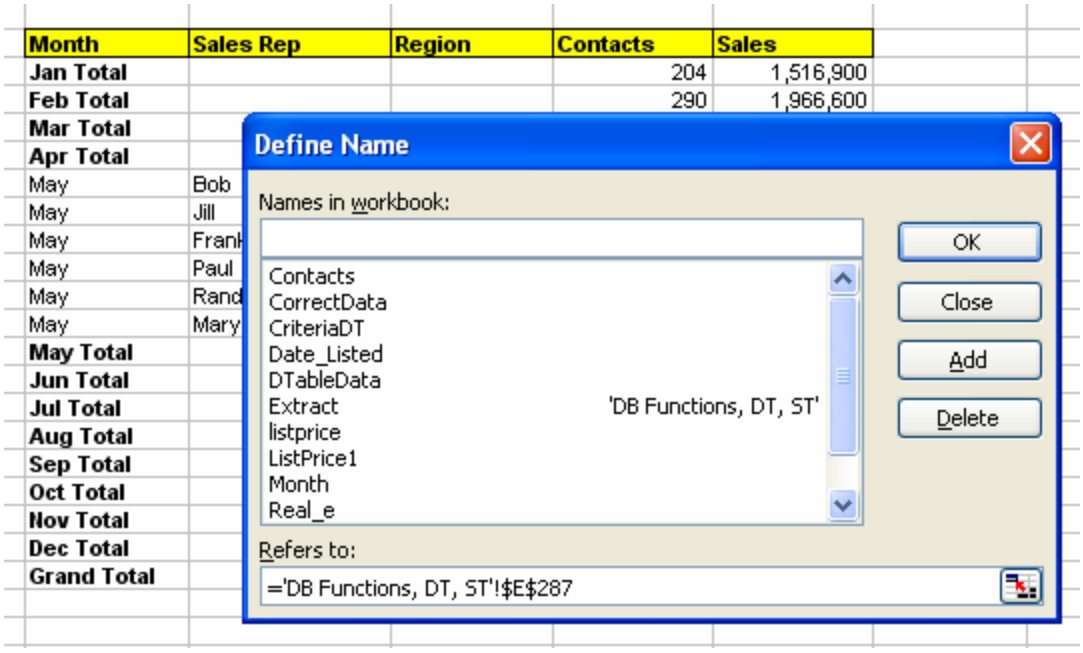


**Figure 17- Parameter storage in Microsoft Excel (also uses the name-value method)**

CATIA, another CAD system, was found to use a similar approach to Microsoft Excel. It provides a broader range of variable types than Unigraphics and variables can be stored and retrieved as VARIANTs (Figure 18). Although the actual storage technique differs, the fundamental concept of a single name corresponding to a single value is maintained and is common to all tools examined thus far.

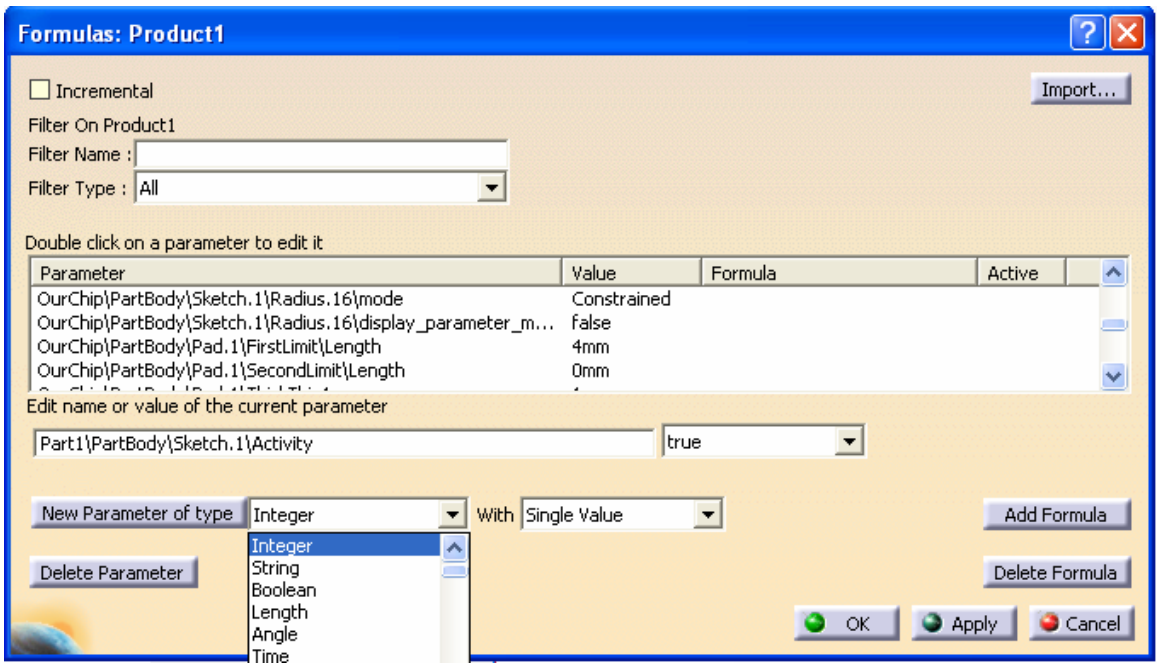


**Figure 18- Parameter Storage in Dassault Systemes CATIA**

Finally, the popular analysis software, ANSYS was investigated and found to follow the trend of storing parameters as name-value pairs.

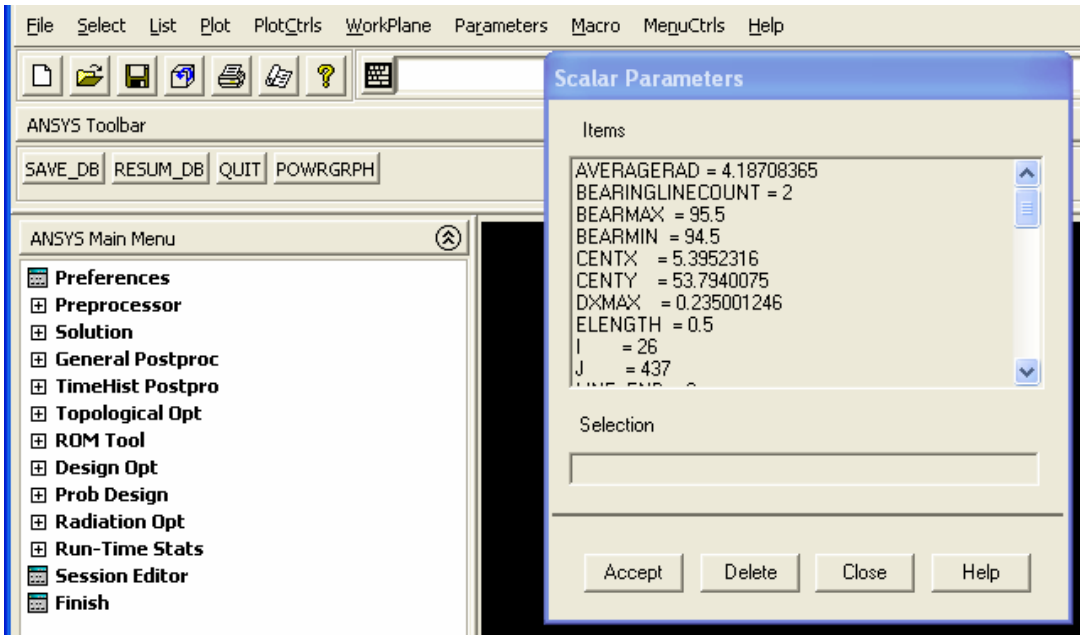


**Figure 19- Parameter storage in ANSYS**

## 5.2 Schema Design

All three categories of the parameter storage were used because this was a prototype system and they needed to be benchmarked. The results would be benchmarked in order to make recommendations. These can be found in the next chapter.

For the spreadsheet, a field-named approach was used because spreadsheets have many names (A1, A2, etc.) so it is best to know beforehand which variables to retrieve. At first, the name-value type was used for the other tools, but after the name-value-ref was invented it made the normal name-value type obsolete.

## 5.3 Determined Access Method

The access method chosen for this prototype was through the API of the PLM system rather than that of the RDBMS. This was chosen after extensive testing of the two methods.

### 5.3.1 Analysis of Direct Database Access

As discussed in the method section, an RDBMS is an autonomous server that simply serves data requests. It resides as a daemon process with no user interface. Because of this autonomy, any access could be considered API usage. Because of this, a wide range of flexible APIs is available for well-developed RDMBS systems. For example, Oracle data can be accessed directly from the programming languages Visual Basic, C/C++, TCL, PERL, PHP, and over 70 others. This makes the RDBMS highly

attractive to developers familiar with databases and, particularly, the standard language to access databases, SQL (structured query language).

#### 5.3.1.1 Direct Access through SQL

Because PLM systems use these databases, they can be bypassed and the data can be accessed directly (Figure 20).

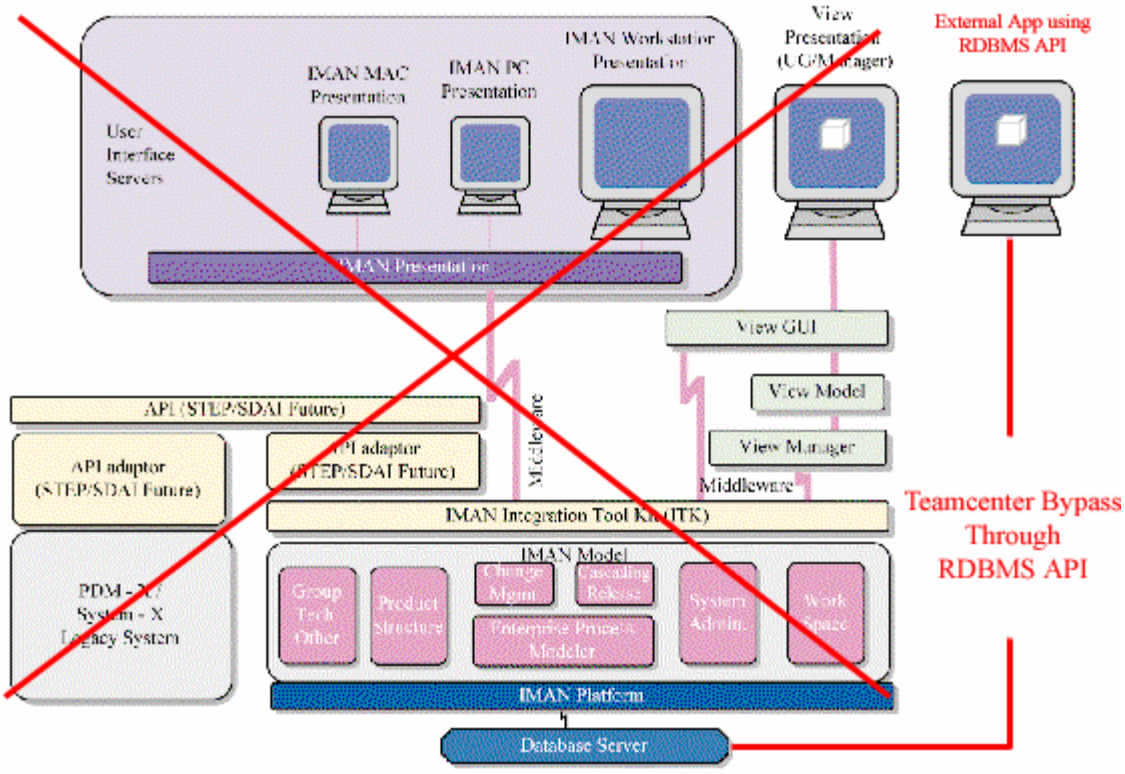


**Figure 20- Data access through RDBMS API**

The first test to prove this bypass of Teamcenter, the SQL Plus utility supplied with Oracle was used. After logging in with the same administrative password supplied to Teamcenter's install utility, a simple SQL statement shows all the tables created by Teamcenter:

```
SQL> SELECT table_name FROM all_tables;
TABLE_NAME
------------------------------
PIMANGRM
PNAMECOUNTER
…
PPATTERN_0
PCOUNTER_TAGS_0
PNAMERULES
PNAMEFIELDS

1094 rows selected.
```

Standard SQL can be used to create new tables, delete tables, add data, remove data, etc. Because of this simplicity and it being so standard, a large library of API calls can be used to rapidly implement tasks. For example, the following script connects to oracle, reads all the Teamcenter tables and their row count, and inserts them into rows in a new Excel spreadsheet.

```
<?PHP

dl ( "php_oracle.dll");
$row = 0;
$excel = new COM("Excel.Application");
$excel->Visible = true;
$nwb = $excel->Workbooks->Add();
$conn = ora_logon( "infodba", "infodba")
        or die;
$cursor = ora_open($conn);
$query = "SELECT table_name, num_rows FROM all_tables
                                WHERE owner LIKE 'INFODBA'";
ora_parse($cursor, $query) or die;
ora_exec($cursor);
$numcols = ora_numcols($cursor);

        $row += 1;
$sheet = $nwb->WorkSheets(1);

while(ora_fetch($cursor)){
        $row++;
        $cell1 = $sheet->Cells($row, 1);
        $cell2 = $sheet->Cells($row, 2);
        $cell1->Value = ora_getcolumn($cursor,0);
        $cell2->Value = ora_getcolumn($cursor,1);
}
//$excel->Quit();
unset($excel);

?>
```

This is called only after the table statistics are generated with the following SQL:

```
SQL>spool analyze.sql;
SQL>select 'analyze table '||table_name||' compute statistics;' from user_tables;
SQL>spool off;
SQL>@analyze.sql
SQL>select table_name, num_rows from user_tables;
```

Similarly, the data from Oracle can be used to update a CATIA part. The language used is PHP, and it simply uses COM interfaces to interact with CATIA and update all the radii in a sketch:

```
$catia = new COM("CATIA.Application");
$docs = $catia->Documents;
$doc  = $docs->Open("C:\\energymodels\\Panel.CATPart");
$part = $doc->Part;
$bodz = $part->Bodies;
$body = $bodz->Item("PartBody");
$skchz= $body->Sketches;
$skch = $skchz->Item("Sketch.1");
$cnst = $skch->Constraints;

while(ora_fetch($cursor)) {
        $row++;
        $param = $cnst->Item("Radius.$row");
        $len = $param->Dimension;
        $len->Value = ora_getcolumn($cursor,0);
}

$part->Update
$doc->Save();
unset($catia);
```

#### 5.3.1.2 Sharing Data with the User Interface

Programmatic access to the data is obviously not the problem with the RDBMS API scenario. A less obvious question though is whether or not it is possible to interact with the data manipulated externally, or through direct RDBMS APIs, through the portal's graphical user interface (GUI).

This required first an understanding of what the PLM system was doing each time it inserted data into the database. With this, any GUI action could be simulated by inserting rows in a similar fashion in the database.

In order to determine where the changes made by Teamcenter were necessary, a form type was created and the tables were analyzed right before and after to determine what interactions were happening within Teamcenter Engineering. The first PHP script

listed above was used to perform this by viewing which tables were modified during the GUI transaction.

Once the changes were analyzed, the changes were reverse-engineered and imitated using an external script. The reverse engineering was somewhat incomplete because the database changes were complicated. The result was catastrophic and eventually resulted in Teamcenter Engineering reporting that its database was corrupt and needed to be wiped clean. When this attempt was mentioned to UGS, the makers of Teamcenter Engineering, they were not surprised and said that in many cases it could be illegal to even attempt doing so.

Since the PLM schema was not published, scripts were created to inspect changes in the database after Teamcenter operations were performed interactively. This way, tables could be inspected and used externally, and hopefully the results would appear in the interactive tool. Table analysis was a five-step process:

4. Store table information
5. Make a change interactively (create table or instantiate row)
6. Re-analyze and store table information
7. Compare stored table data to find changes
8. View details of each modified table

The simplest operation was instantiating data in an existing table. This was done in Teamcenter by creating a new form. The form would be attached to the user's home folder, which is an important step. After the form was instantiated, it was found that this made a number of changes in the database—many more than expected. The form only had three fields: a blank array, one integer, and one string. The changed tables and contents are shown below (excerpt taken from research log 4/27 2:30 PM):

```
PWORKSPACEOBJECT:
    PUID:                   wkLBobflVJfqVA          (jonid_00)
    POBJECT_NAME:           change_comparison
    POBJECT_DESC:           Just checking this out
    POBJECT_TYPE:           jon_form
    POBJECT_APPLICATION:    Form
    VLA_34_6:               0
    VLA_34_7:               0
    VLA_34_9:               0
    PREVISION_LIMIT:        1
    PDATE_REL:              NULL

PCONTENTS:
    PUID:                   1xOBm24pVJfqVA          (jonid_02)
    PSEQ:                   2
    PVALU_0:                wkLBobflVJfqVA          (jonid_00)
    PVALC_0:                50

PFORM:
    PUID:                   wkLBobflVJfqVA          (jonid_00)
    PFORM_FILE:             n/a
    RDATA_FILEU:            woFBobflVJfqVA          (jonid_01)
    RDATA_FILEC:            322                     NOTE: two previous were 322 as well

POM_BACKPOINTER:
    TO_UID:                 wkLBobflVJfqVA              woFBobflVJfqV
    TO_CLASS:               -1                          -1
    FROM_UID:               1xOBm24pVJfqVA              wkLBobflVJfqVA
    FROM_CLASS:             -1                          -1
    BP_COUNT:               1                           1

PPOM_APPLICATION_OBJECT:
    PUID:                   wkCBm_TWVJfqVA
    PARCHIVE_INFO:          1pNBm24pVJfqVA
    ROWNING_GROUPU:         31
    PACL_BITS:              0
    RLAST_MOD_USERU:        1xIBm24pVJfqVA
    RLAST_MOD_USERC:        37
    ROWNING_USERU:          wgMBm_TWVJfqVA
    ROWNING_USERC:          37
    PCREATION_DATE:         31-MAR-04
    PARCHIVE_DATE:
    PLAST_MOD_DATE:         31-MAR-04
    PBACKUP_DATE:

PPOM_OBJECT (1)(2):
    PUID:                   AIJBobflVJfqVA                  woFBobflVJfqVA  (jonid_01)
    PTIMESTAMP:             AILBobflVJfqVA                  woHBobflVJfqVA
    PPID:                   12                              322
    ROWNING_SITEU:
    ROWNING_SITEC:          -1                              -1
    PIS_FROZEN:             0                               0
```

For the amount of data that was stored, this was much more complicated than expected. Field values that are of the form of a 14 character ASCII string are actually references or “pointers” to other tables. Some of these were tracked and are noted as to what they point to in the right-hand column notes (i.e. `jonid_02`). When the most

important table was located to be “PWORKSPACEOBJECT,” an attempt was made to instantiate the form without Teamcenter. Teamcenter was then checked and the object was not there. After other attempts it was clear that this was causing problems and eventually the entire database had to be erased and created anew, a situation that would be entirely unacceptable under normal circumstances.

Although these results were somewhat anticipated, they provided valuable information leading to the conclusion that it is better to use the API of the PLM tool than to circumvent it altogether. The main disadvantages found include:

- Security has to be bypassed as external data-accessing programs require database passwords.
- Integrity of the PDM schema can easily be violated, causing erred links and ultimately PDM system failure.
- Complexity makes development time.

### 5.3.2 Analysis of Database Access Through the PLM Tool API

PLM systems typically provide a wide variety of methods for customization due to the diverse needs of companies. This API is usually in the C programming language and is integrated into the PLM server. Teamcenter Engineering’s API is called the Integration Toolkit (ITK). It allows companies to customize various aspects of the behavior of the system. These functionalities can either be run as external programs or as part of the normal portal servers.

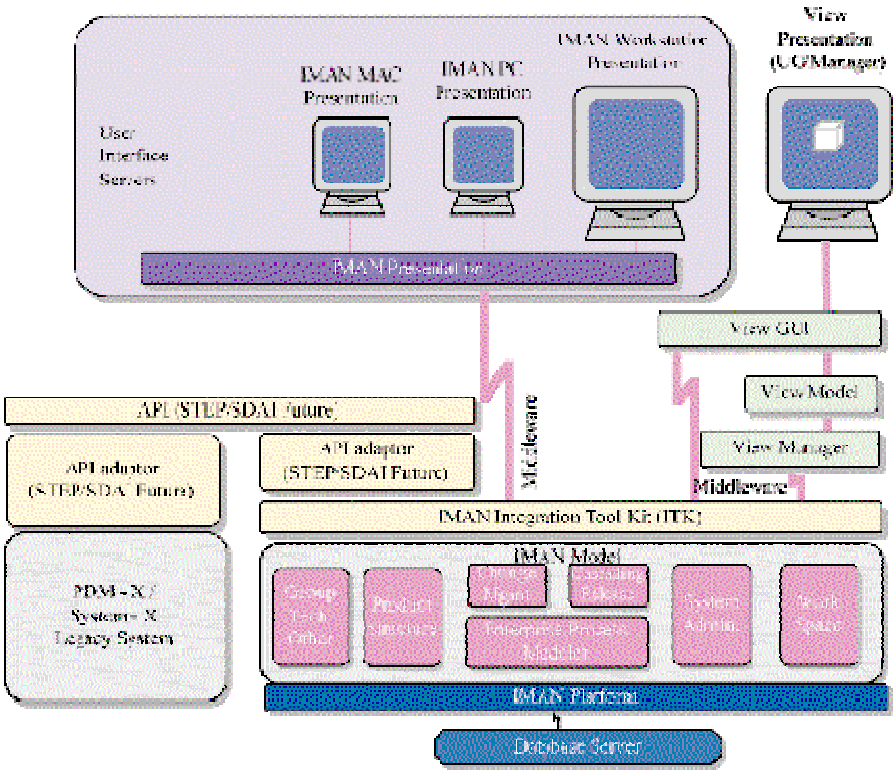


**Figure 21- Sample PLM system's programming hierarchy**

Figure 21 shows the programming hierarchy of the typical PLM system. It shows that the ITK layer is fundamental, as all connections and actions related to the database must pass through it. This shows the great amount of flexibility put into the PLM system, as such a core module is customizable. As such, there is a great deal of power if some development investment is given.

As mentioned earlier, ITK programs can be run independently or custom functions can be built directly into the PLM system. When functions are integrated directly, they are referred to as user functions, and they provide custom functionality even when using the portal GUI.

### 5.3.2.1 Defining the Data Table

ITK was used to first method tested for simple data access. At first, it was found that it was easier to create data in the GUI using the standard form interface mentioned earlier. The class was defined using the *Admin->Type* component. Then, a custom form was created using the form wizard and the *"Define new sub-class"* option was selected. This means that the form's data component would contain data as specified.

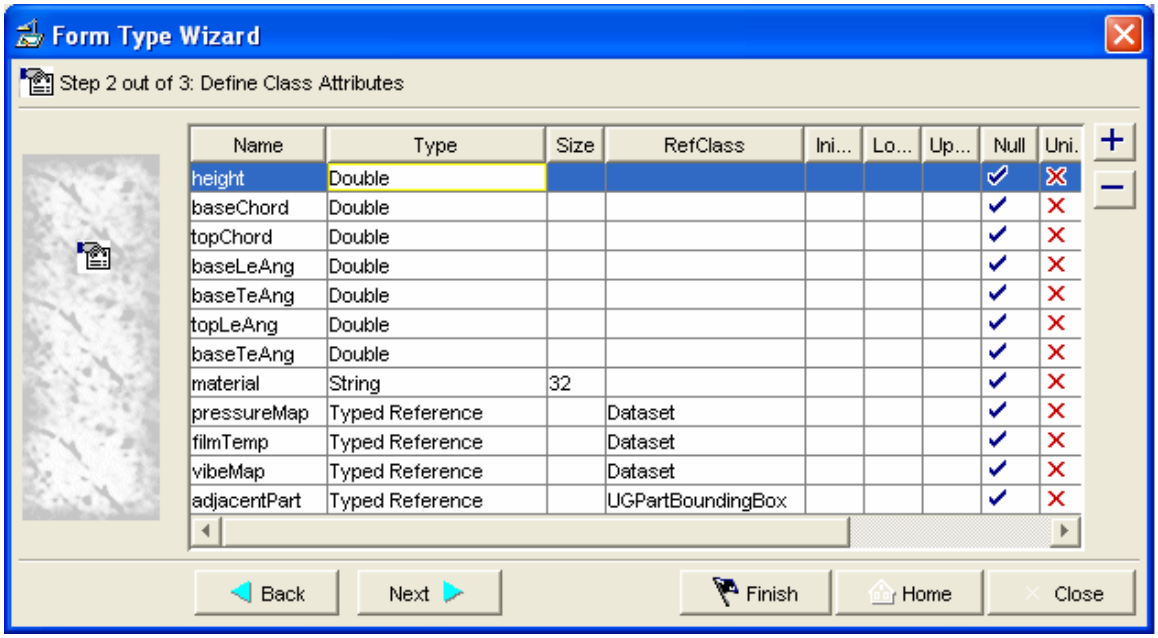


**Figure 22- Interactive specification of data table**

It is important to note that the portion of the wizard in Figure 22 is not actually defining a form itself, but rather the Persistent Object Manger (POM) class (or database table) that will hold the form's data. The form type itself is not a new database table, it is simply a definition of a certain form type—specifically containing a name, display type, and a reference to a POM class. Therefore, the form instance is the component the portal uses to display the data in a POM class, and the POM class itself is the data object.

#### 5.3.2.2 Programmatic Data Access

With a data table created, it could then be accessed programmatically through ITK functions. The first step in doing this was to get a handle to the POM class and instance. Although two methods could be used to accomplish this, only one maintains any type of link to interactive component.

The first and recommended method is to locate the form that is linked to the POM instance. Because the form is the graphical component used by the portal, it is useful to maintain all data linked to a form. Forms can be located by their name and even more specifically by what type of POM_object they are *(form)*. Once the form was located, the link to the POM instance of the data could be read.

The second approach to obtaining a handle to a data collection is to perform an enquiry based on the POM_class and some portion of the data in the data instance. While this will also locate the data, it is not common that only one object can be located with such a query unless some unique identifier is also stored within the table. Regardless, for cases such as just reading static data, it functions well and could be useful.

```
POM_class_id_of_class  ( "PPM_Test", &PPM_class_id) ;
POM_attr_id_of_attr( "numstages", "PPM_Test", &PPM_form_int_id);
POM_attr_id_of_attr( "radius",        "PPM_Test", &PPM_form_double_id) ;
printf("Enter The Id of Sample Form : ");
scanf("%s", buf);

cmp_str = (char *)strdup (buf);

POM_create_enquiry_on_string (PPM_class_id, PPM_form_int_id,
    POM_is_equal_to, &cmp_str, &enq_id);
```

Regardless of the approach used to locate the data, with a handle to the data, the data within that instance could be accessed using ITK's `POM_ask_attr*` functions such as:

```
for(i=0; i<n_instances; i++)
{
        POM_ask_attr_double( *(instances + i), PPM_form_double_id,
                &my_double, &is_it_null, &is_it_empty );
        printf("\nWE FOUND A FORM THE VALUE: radius = %lf\n\n", my_double);
}
SM_free(instances);
```

This prints the values of all the “radius” field in the data instances returned from the enquiry. Obviously, this is more work than the equivalent SQL required to perform the same task:

```
SELECT radius FROM PPM_Test;
```

The use of the PLM tool’s API was found to be viable due to its self-maintenance of data consistency with the PDM software. Through this approach, the risk of corrupting the database is minimal and compatibility among software versions would be maintained.

In order to analyze the performance of this API Figure 23 shows Teamcenter Engineering’s persistent object model (POM) hierarchy. This hierarchy has been normalized and is known to perform very well. It is object-oriented in nature, with each data class (with the exception of the highest level, the POM_object) inheriting methods and attributes from a parent class.

By adding another table to the existing schema, Teamcenter will use it as one of its own. It will be rapid in development and usage, and there are simple C methods for accomplishing this.

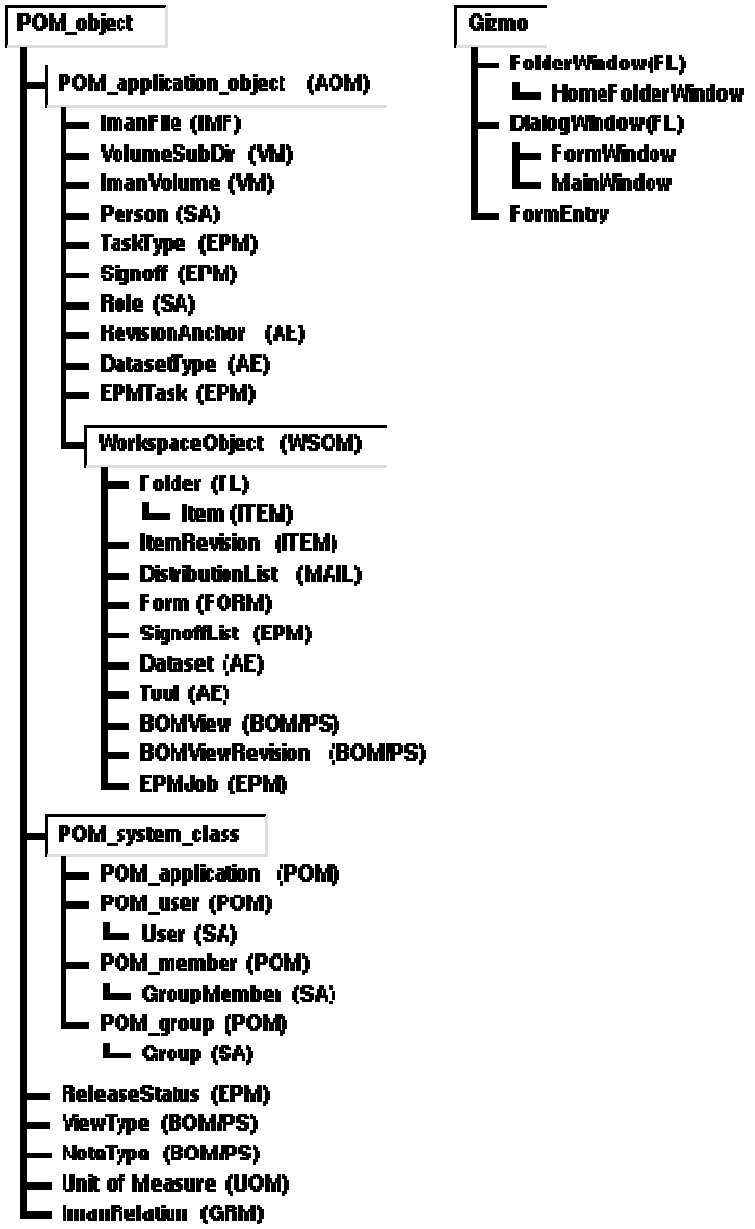


**Figure 23- Partial POM diagram of Teamcenter Engineering**

The most familiar objects will be those found under the POM_application_object class, and more specifically, the WorkspaceObject (WSOM) class. Any object created in under the WSOM superclass inherits parameters such as date created, owner, permissions, etc. automatically. This greatly reduces the work required for customization

because these attributes will be automatically handled by the existing methods. Layered approaches such as these also help in error trapping and debugging.

Upon creation of these objects, the API also allows the storage and manipulation of arrays. However, a single field in a database table can only contain one value, so one might question how this is accomplished. The answer is that for every entry that is defined as an “array”, another table is actually designated to hold values for that particular array and the value in the original object’s array field is actually a pointer (in a sense) to the new table. By using the API of the PLM system, this work is greatly reduced because it is encapsulated within simple functions.

### 5.3.3 Comparison of the Two Methods

In comparison, there were advantages and disadvantages to each method. For example, searching and summarization is best performed by the RDBMS itself. If the desired summary information can be formulated into a structured query language (SQL) query then the power of the RDBMS will be leveraged to retrieve only the desired information after it has been filtered. The alternative is to narrow the possibilities as much as possible with a well-formulated query and then to perform the filtering after the results have been returned. This is much less desirable, but may be required if arrays are to be used.

For example, if a database table contains no arrays and just standard fields, an SQL statement can be constructed that allows the RDBMS to filter the data before it is returned. Statement:

```
SELECT * FROM articles WHERE MATCH (article_title,abstract,keywords) AGAINST ('plm')
ORDER BY MATCH (article_title,abstract,keywords) AGAINST ('plm') ASC LIMIT 0,35
```

This tells the RDBMS to return only the first thirty objects that have the word 'PLM' in the article_title, abstract, and keywords fields. On the other hand, if the data is not filtered, then the client machine must perform the filtering itself. Example:

```
$result = mysql_query("SELECT * FROM articles");
$filtered = array();
while($arr=mysql_fetch_array($result)) {
        if( preg_match("/PLM/", $arr["article_title"]) ) {
                array_push($filtered, $arr);
        }
}
```

The performance differences are substantial in favor of the RDBMS API, not only in CPU client and server requirements but also network bandwidth usage. However, the corruption of the databases was unacceptable, therefore the PLM API was chosen.

## 5.4 Determine Implementation Categories

Each engineering tool in question was analyzed to determine the type of implementation. From above, the possibilities are 1) internal functions, 2) external functions without database access, and 3) external functions with database access. These have been listed in order of complexity and thus desirability.

### 5.4.1 Categorizations

The internal functions were Excel, CATIA, and some legacy programs. The legacy programs that can be used in this way are those where the source is available and can be compiled directly into the PLM customization. Excel and CATIA were referred to as internal (although technically they have their own processes) because they can be

controlled from within the PLM system customization through common object model (COM) interfaces.

Next, the external functions without database access were legacy programs that just require input files and some other programs that can run with macros. Tecplot Visualization is one example of this. The customization had to include writing a macro file, writing the data file, and then launching Tecplot with a command to run the macro.

Finally, the most complex programs were those that were required to run externally but still maintain access to PLM data during runtime. These included Unigraphics and ANSYS. This posed the greatest challenge, and it was worked around using sockets, as discussed in following.

### 5.4.2 External Access Method—Sockets

The drawbacks of using the ITK API make certain tasks significantly more difficult. For example, Teamcenter login is required at the outset of standalone programs. This requires that ITK implements the main symbol, which is the entry point for executables. In simple programs this is not a problem, but it becomes quite a challenge when Teamcenter function calls are needed to be called from other customized applications. This is the case with any application that requires that customizations be made inside a dynamic link library (.dll) or shared object (.so). One example of this is the user function feature of Unigraphics. Many companies develop their own parametric customizations through these user functions. Their entry point is defined by an exported function called ufusr. Because ITK does not own this entry point, it does not allow this approach to be used.

If other programs were not able to access the parametric data, there would be significantly less incentive for the work of this thesis. Therefore, a method was devised to provide data access to external applications with their own security domains while still using the ITK API. This way, the PLM system did not have to be bypassed in any way, maintaining all the data control, security and business rules inside the PLM system.

Because the problem is a matter of entry points and security domains, the solution was to provide a mini server that would serve connections to data-desiring clients. This was simply to bridge the gap between the need for two programs to use the same data. This architecture can be seen in Figure 24, with the PPLM acronym representing Parametric Product Lifecycle Management.

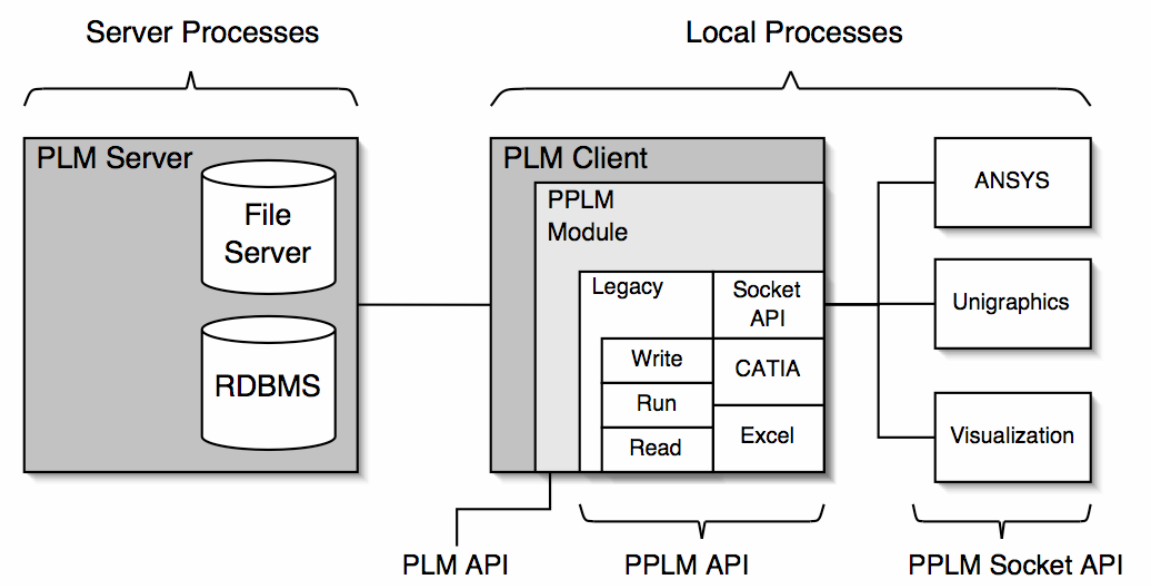


**Figure 24- Security domains and the PPLM socket API**

Figure 24 shows the architecture various domains where these functions can reside. The boxes on the far right, or the PPLM Socket API boxes represent programs that cannot run in the context of the PLM client. For example, ANSYS and Unigraphics are

both complicated programs that require their own sets of environment variables. Visualization software such as Ensight or Tecplot may run with their own client-server architectures. These context-sensitive programs are the ones requiring the PPLM socket API.

The way this socket API works is a function is launched from the PLM client that has data access to certain objects. In order to forward this access to others, it stands as an intermediate proxy that communicates with the external programs. These external programs request data transactions of the socket API in a language modeled after SQL, but specifically for parametric management.

First, the socket function launches the external process and tracks its process id number (PID). Then, as in other TCP/IP communications, there is a three-way handshake where the client identifies itself and provides its PID. This is checked by the socket server with the one stored when the program was spawned and if it matches it proceeds to serve transactions (Appendix A: PPLM Socket example—ANSYS ). For simplicity a language was created and limited to basic functions (Appendix B: Socket API Syntax). Clients can use this language without actually handing the socket functions themselves using yet another API written in C++. This can be included into the project and linked dynamically using the PPLM .dll. This API provides the following class for socket access:

```
class DFM_Data
{

        public:
                DFM_Data() {dfm_connect();}
                ~DFM_Data() {dfm_close();}
                setValue(char* name, char* value);
                setValue(int index, char* value);

                getValue(char* name, char** value);
                getValue(int index, char** value);

                int sendsomething(char* message);
                int dfm_close(void);
```

```
        int dfm_connect(void);

private:

        WSADATA wsaData;
        SOCKET m_socket;
        sockaddr_in service;
        PHOSTENT localhostinfo;
        char name[256];
        char* my_ip;
        char sendbuf[CUSTOMBUFSIZE];
        char recvbuf[CUSTOMBUFSIZE];
```

The implementation of this class can be found in Appendix C:. This class then simplifies the responsibilities of the programming within the external programs. For example, a program was created as a user function for Unigraphics that leverages this class. First, at compile time it links to the class and its functions through the .dll. Then, at run time it calls the functions simple function getValue() to retrieve the expressions needed to update the Unigraphics model. Finally, if any values were needed to send back to the database, it could use the setValue function (not implemented in the Unigraphics program.) The code for this is can be found in Appendix D:Client usage of the C++ socket API library—Unigraphics.

For programs that do not use C++ as their API the connection can still be made. As long as the API of the external program can access socket functionality, the implementation is still possible. For example, ANSYS makes use of a proprietary scripting language, ANSYS Parametric Design Language (ADPL). This language has methods that can make calls to Tool Command Language (TCL) functions. Since TCL has the ability to access sockets, it was able to interface with the same socket API created for C++. This language abstraction was handy in standardizing the usage and language of the API. The following TCL/ADPL code was used in an ANSYS macro () to retrieve values from Teamcenter at run-time.

```
~eui,'source [file join C:/IMAN0900/bin/DFM_ansys.tcl]'
```

```
~eui,'DFM::getValue Length'
~eui,'DFM::getValue Height'
~eui,'DFM::getValue Width'
~eui,'DFM::getValue Web_th'
~eui,'DFM::getValue Flange_th'
~eui,'DFM::getValue Load'
```

The first line of code retrieves the TCL API from the Teamcenter bin directory where all common Teamcenter libraries are located. The code for this API can be seen in Appendix E:.

# CHAPTER 6: RESULTS AND DISCUSSION OF RESULTS

This chapter discusses the outcome of the prototype system developed to test the methodology prescribed in the previous chapter. The prototype was implemented with Teamcenter Engineering as the PLM system and Oracle 9i as the underlying RDBMS. The prototype system was developed to test the important features of the methodology: store model parametrics in a PLM system's RDBMS.

## 6.1 Specific Objectives

The specific objectives targeted were analyzed and the assumptions mentioned previously were proven. The objectives were:

- An open (non-proprietary) storage of key design parameters
- Automation-friendly data storage and access
- Increased PLM system performance
- Standardization of data and communication
- Decreased development time

## 6.2 Approach and Result

The approach was executed as prescribed in the implementation chapter. The tools used were Teamcenter Engineering, Oracle 9i, Integration Toolkit, C++, and TCL. Data extracted from engineering KBE-related tools were stored in the database of PLM system's database.

The prototype was successful. It was capable of representing design iterations as metadata in a commercial PLM system, and met all the objectives of the thesis. The system is robust, and if there are any failures, it recovers from them because of the transaction-based framework of the database. For example, if an action item does not complete, it does not get flagged as complete, and therefore if the client disconnects at any time or an operation fails, it will be restarted upon the next login.

The prototype has been demonstrated on numerous to academic and industrial audiences with good responses. It has also provided the basis for other successful work. Some of the lessons learned provide areas of future work and recommendations for implementation. It also demonstrates the key advantages that motivated this research, as follows.

### 6.2.1 Automation-Friendly Data Storage and Access

As experienced by various authors mentioned in the literature review [1] [2] [16] [18] [19] [21], an RDMBS was seen as an effective tool for enabling automation. Inter-process communication is possible because of the ability of the RDBMS to serve

rmultiple clients simultaneously. Not only does this enforce data concurrency but it simplifies the task of data formatting, file handling, and security.

Another advantage worthy of note is how databases can quickly provide summary information. The RDBMS can quickly search data based on specific criteria, and can rapidly summarize information. This can be extremely useful in optimization, where thousands of result fields may be generated. As an example, the Pareto front of a multiple-objective optimization could be retrieved with a single query, saving time while still providing access to distributed clients.

### 6.2.2 Increased PLM System Performance

In the simple impeller example discussed previously, parameter storage and network requirements for saving a CAD model are 1,000 times greater than saving its parameter set. In other cases, it could commonly be six to eight orders of magnitude greater [2]. As shown in the literature review chapter, this becomes especially important in situations involving large numbers of iterations such as a large assembly optimization and design space characterization.

### 6.2.3 Standardization of Data and Communication

By requiring the storage in a database, one must make the decision ahead of time what parameters will be used. Also, if the parameters for one model are to be used again, they must be in a well-defined format. This is handled by the requirement of storing them in the database, which requires that the parameters be specific and precise.

Also, data formats and standards are automatically communicated to other individuals by providing access through the PLM system. Others are able to view the schema definition and thus the types of variables used are apparent. This simplifies some of the communication tasks in collaborative scenarios.

### 6.2.4 Decreased Development Time

The amount of development time consumed will depend on the type of project. The situations for which this methodology is recommended are those involving high number of object modifications and knowledge reuse. In these situations, not only will development time improve from better hardware speeds, but the ability to automate tasks from various locations. This reduces the need for file transfers, emails, phone calls, etc. to communicate data from one department or company to the next.

## 6.3 Schema Design Comments

Various schema configurations were devised in order to meet the needs of the KBE tools. In doing this, various possibilities were encountered, such as the field-named, name-value, name-value-ref schema types.

The ability to store a variable as a persistent pointer, or one that maintains its linkage even after the program terminates, is novel in the field of engineering parametrics. This is made possible by the name-value-ref schema type. This is not possible with normal programming because memory addresses are under the control of the operating system and therefore change upon program termination. Although the

possibilities this creates are out of the scope of this work and deserve extensive research in themselves, a few important implications are mentioned here:

- Discrete optimization is made possible by simply swapping the value of the reference field.
- Reports can be generated that actually contain the objects (or references to) that they came from
- CAD assemblies can be created simply from a variable list, with each variable representing a part
- CAx data can be stored side-by-side inside the same object

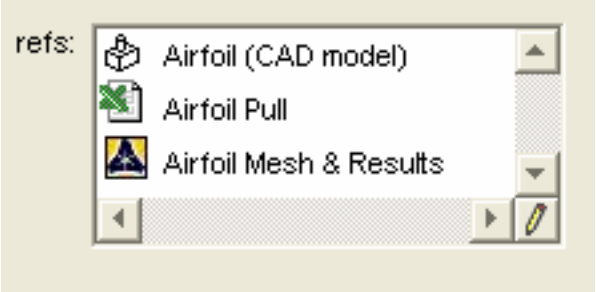


**Figure 25- Side-by-side CAx storage in single object**

## 6.4 Database Access Method Results

One of the major questions this research encountered was about how the data stored in the PLM system's database should be accessed. Because the RDBMS is a separate program in itself, it has its own API through which data can be accessed. However, highly developed PLM systems provide APIs themselves that can also be used to access RDBMS data. Both methods were tested for this research and the PLM systems' API is

the recommended method. However, this could vary according to the complexity of the PLM system used. For complex systems, direct access is discouraged.

Although the SQL approach is much simpler and flexible due to its more extensive APIs, the advantages of using ITK make the latter the recommended choice. The most important advantage is the fact that the use of ITK removes most of the responsibility of data corruption from the customizer and leaves it with Teamcenter itself. It also ensures that all security policies are enforced and logins are required. Some PLM software vendors even have legal protection against direct database access.

The speed of the direct database access is also much better than that of the ITK calls. In order to get a quantifiable comparison between the two a speed test was conducted that consisted of a login and the search of one row of a specific POM class. The ITK version was compiled into a standalone executable, first.exe. The SQL was executed with a script. The test was timed cumulatively as it ran 100 times, and the results greatly favored the direct access with SQL method (Figure 26). The tests were performed on a machine running both the server and client for Teamcenter Engineering v9.0.

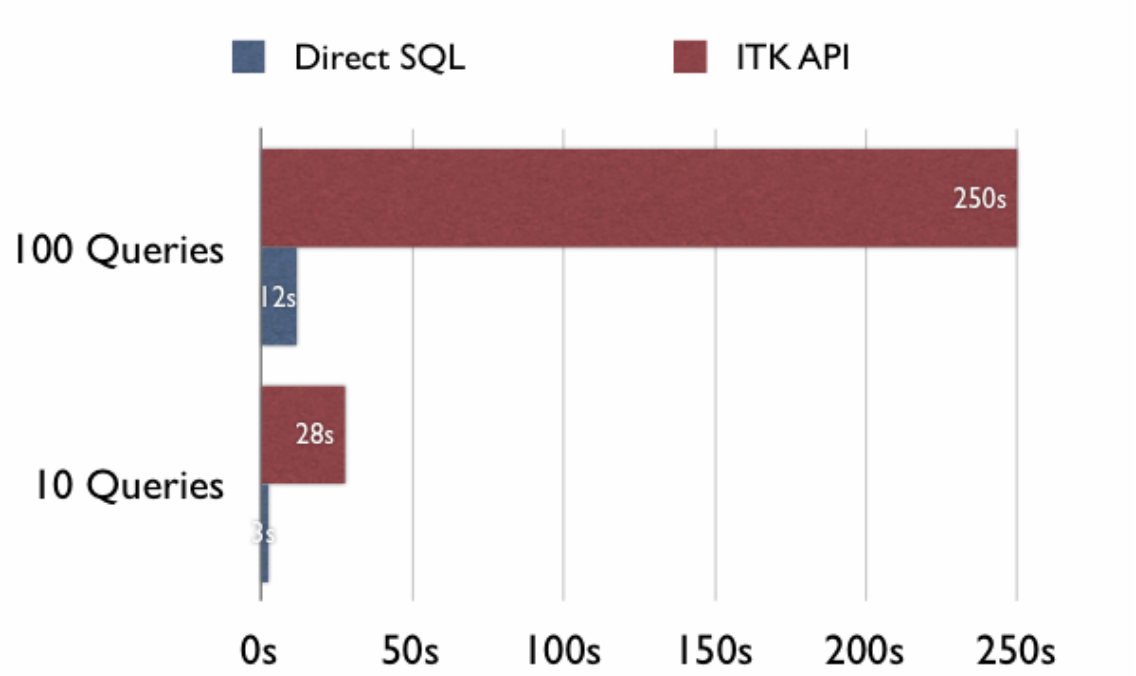


**Figure 26- Login and query speed comparison of SQL vs. ITK**

We can see that the speed of accessing Oracle from its own API is much faster that using the layers in Teamcenter Engineering. The difference between the performance of the two is likely even greater pronounced when the server is on a separate machine from the client. From a performance perspective, it is unfortunate that the security and control is weighted more heavily. However, higher performance would be useless if the integrity of the data could not be ensured.

## 6.5 Other Tests

Other tests were performed to benchmark the efficiency of saving simply metadata versus entire files. In all cases, the Teamcenter server was not located on the same machine to act similar to production PLM systems. The bandwidth requirements,

parametric efficiency, and data summarization capabilities were measured with basic tests.

### 6.5.1 Bandwidth Requirements

The first test compared the network bandwidth requirements my measuring transaction time. The time was dependent upon the number of bytes transferred, so three datasets and a simple metadata object were used. During the test, these four objects were repeatedly downloaded from the Teamcenter server and the time was averaged for each. The results are found in Table 3.

**Table 3- Benchmark of bytes vs. time**

| Setup | *Simple Form* | *Text File* | *Spreadsheet* | *CAD Model* |
|---|---|---|---|---|
| Bytes | 24 | 71 | 540000 | 3268000 |
| Time (s) | 0.0032 | 0.02 | 0.26 | 0.94 |

A simple linear regression on this data shows that for any given number of bytes needed to be transferred, *s*, the duration of the transaction time as a function of bytes transmitted can be expected to be:

$$T(s) = 3.0e^{7}s + .404 \quad \textbf{(3)}$$

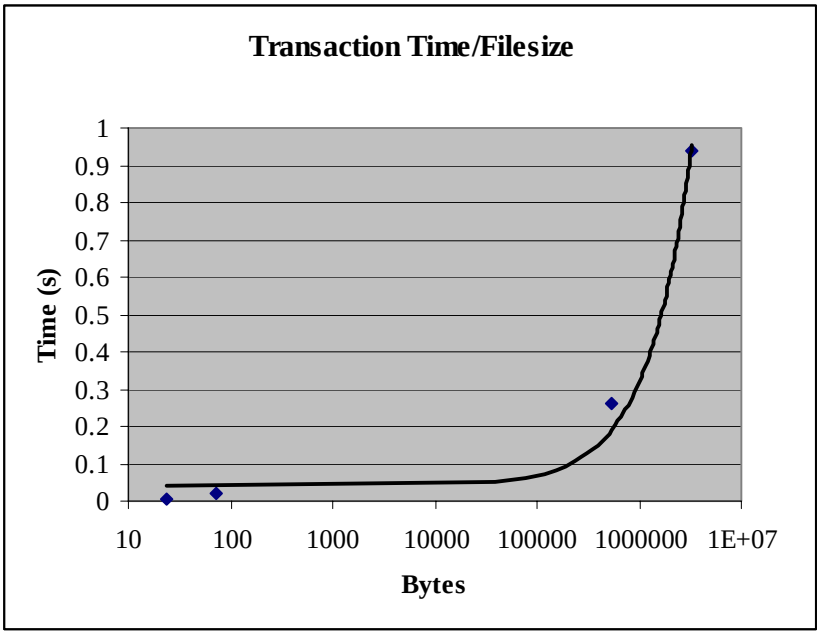


**Figure 27- Transaction time versus file size**

It can be seen that the smallest object is even faster than what would be expected of the similar file size. To test this, a text file of the exact same size (24 bytes) was exported 1870 times in 36 seconds. This results in an average transaction time of .0193 seconds per byte. This is slightly less efficient, and the regression is adjusted to be:

$$T(s) = 3.0e^{7}s + .464 \quad \textbf{(4)}$$

These results show that storing data as RDBMS metadata is only slightly less efficient when strictly looking at bandwidth requirements.

### 6.5.2 Parametric Efficiency

The second test tested the efficiency as the number of downloads increases. Because a master model must be downloaded at least once, there is a fixed cost of one download the size of the entire master model. Subsequent transactions realize the

significant benefits of the method. The test was performed with one CAD model that is 3.2 Megabytes and has 75 associated parameters. Figure 28 shows the drastic benefits of parametric usage, with a logarithmic scale.

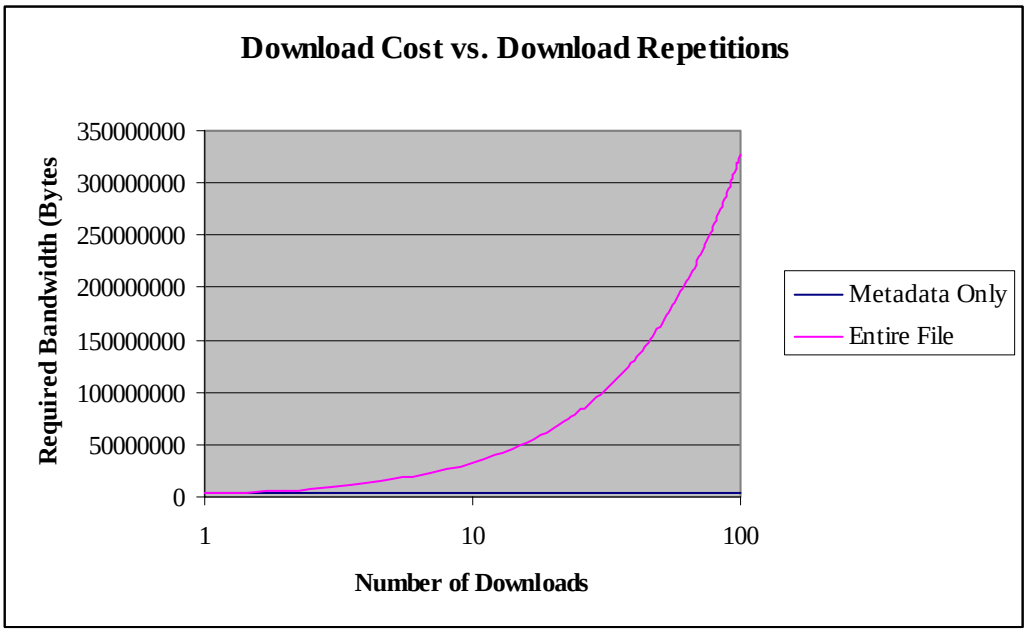


**Figure 28- Download cost vs. download repetitions**

### 6.5.3 Data Summarization

Finally, the ability to summarize the data was tested. This type of task is especially useful in optimization or setting company standards. It can also be extremely effective for design space characterization and other statistical analyses. In the case of the metadata storage, the summarization was performed using the query in Teamcenter Engineering's portal (Figure 29).

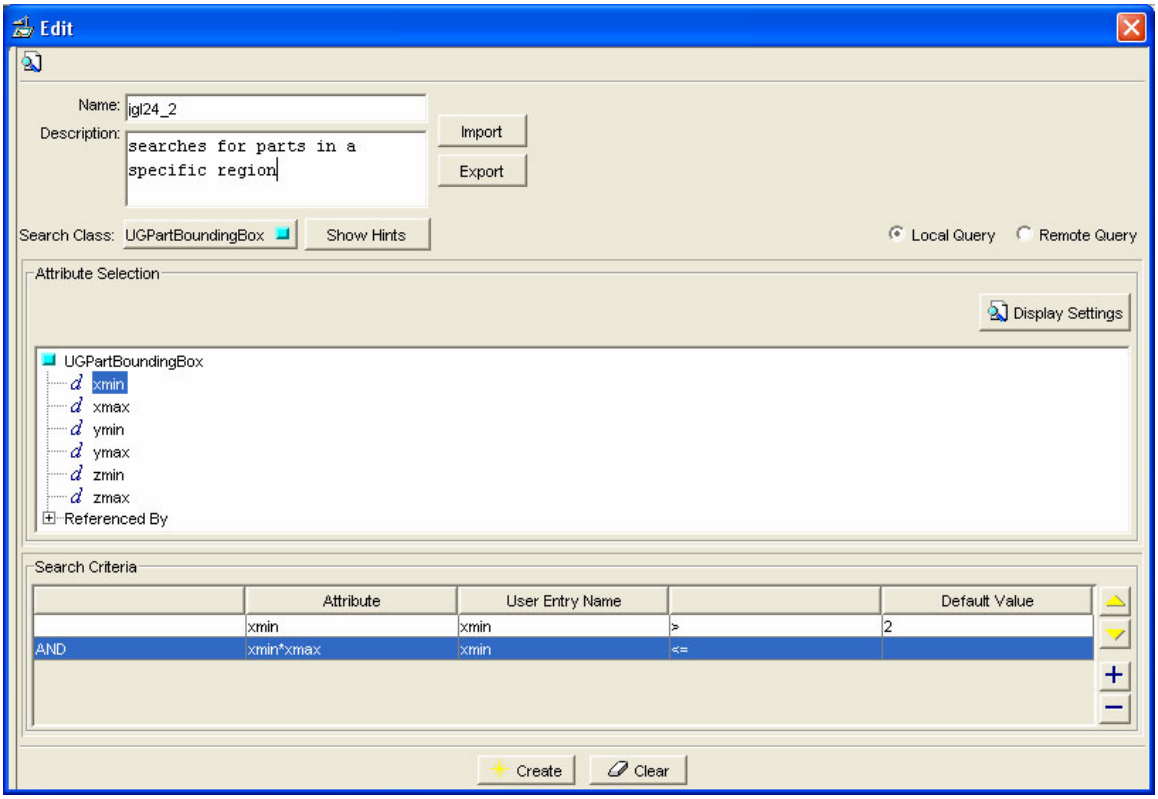


**Figure 29- Custom search from Teamcenter Portal**

This search took approximately 0.5 seconds. The same summarization in a program required that all the instances be loaded in their entirety and inspected individually. Not only was this much more tedious to program, but execution was extremely slower.

Although these results favor the storage of metadata parametrics, there is still one argument that must be satisfied. This is that using the database extensively can decrease the performance. The previous tests were performed with a fairly empty database. Therefore, in order to test how the speed might be affected by storing many more, the same query was performed after 10,000 forms of this type were added. No human-detectable differences in performance were noted.

# CHAPTER 7: CONCLUSION

The objective of this thesis was to present a design paradigm for storing model variations as basic PLM metadata. design optimization within a PLM system. This thesis answered the following related questions:

## 7.1 Can variant data be efficiently stored in PLM systems?

It was demonstrated that data can be efficiently stored in the PLM system by a successful implementation of a prototype. The prototype proved that the storage of the parameters in the PLM system could be much more efficient than that of files in terms of both speed and size. This improved efficiency can be significant, especially in interactive applications or when working with complicated models.

## 7.2 Can PLM systems be used as a basis for automation?

The prototype demonstrated data access in ways typical to automation. Data was accessed from multiple external programs and inter-process communication was enabled by the client-server architecture of the PLM system. These are the criteria for task automation and meeting them shows that this approach makes a PLM system a feasible basis for automation.

## 7.3 Can parametric data storage improve PLM system benefits?

The storage of model variations has enormous potential increases in efficiencies. This is especially true for situations of multidisciplinary design optimization (MDO) where the advantages of the methodology are leveraged.

## 7.4 Future Work

The result of the research has been an enabling technology for advancement in collaborative design. Therefore, there are many scenarios that can leverage what has been done to advance product development. With a basis for storing design parameters in neutral, distributed format, further development could enable distributed automation, CAD model direct translation, parameter-driven assembly configuration, inter-departmental and inter-site optimization, discreet optimization.

Work towards distributed PLM-based optimization leveraging parameter-based PLM has already been accomplished by Fife [25]. He used task handlers in PLM workflow to perform optimization—all the parameters were stored into the PLM's database using the methods prescribed in this research.

It is recommended that future work be aware of the methods prescribed in this work. The considerations for schema design should be especially useful.

## 7.5 Final Comments

The storage of parametrics inside a PLM system is an enabling technology that should open the door to exciting new possibilities in production PLM systems. These

systems will be able to host execution of model updates, engineering task automation, and even optimization.

Although there are many advantages that can be achieved by use of this approach, it may not be suitable for all applications. The methodology was designed for applications in which knowledge reusability needs to be PLM-based for collaborative or distributed purposes. It would not be practical to store every model created as a separate master model and associated parameter set. Instead, the parametric PLM approach should be used as a tool to assist in streamlining time-critical applications or computationally intensive tasks.

# REFERENCES


1. Xu, X. William, Liu, Tony (2003), "A Web-Enabled PDM System in a Collaborative Design Environment", Robotics and Computer-Integrated Manufacturing, vol. 19, pp. 315-328.

2. Leong, K.K., Yu, K.M., Lee, W.B. (2002), "Product Data Allocation for Distributed Product Data Management System", Computers in Industry, vol. 47, pp. 289-298.

3. Chu, Xingjun, Fan, Yuqing (1999), "Product Data Management Based on Web Technology", *Integrated Manufacturing Systems*, vol. 10, pp. 84-88.

4. EDS, "Online Help Collection", *Teamcenter Engineering*

5. Philpotts, Mike (1996), "Introduction to the concepts, benefits and terminology of product data management", *Industrial Management and Data Systems*, vol. 96, pp. 11-17.

6. Wang, Ying. Daisy, Shen, Weiming, Ghenniwa, Hamada (2003), "WebBlow: A Web/agent-based multidisciplinary design optimization", *Computers in Industry*, vol. 52, pp. 17-28.

7. Liu, Tony, Xu, X. William (2001), "A Review of Web-Based Product Data Management Systems", *Computers in Industry*, vol. 44, pp. 251-262.

8. Xu, X. William, Liu, Tony (2003), "A Web-Enabled PDM System in a Collaborative Design Environment", *Robotics and Computer-Integrated Manufacturing*, vol. 19, pp. 315-328.

9. Koonce, David, Department of Industrial Engineering, Ohio (2003), "A Hierarchical Cost Estimation Tool", *Computers in Industry*, vol. 50, pp. 293-302.

10. http://www.oculustech.com

11. *Installation and Setup Guide for FIPER*, Engineous, Inc.

12. Lund, Jonathan, Fife, Nathaniel, Jensen, C. Gregory (2004), "PLM-Based Parametrics for Design Automation and Optimization", *Computer Aided Design*, vol. 2, Nos 1-4.

13. Delap, Damon C. (2003), "CAD-Based Creation and Optimization of a Gas Turbine Flowpath Module with Multiple Parameterizations", *Brigham Young University Master's Thesis*

14. Roach, Gregory M. (2003), "The Product Design Generator--A Next Generation Approach to Detailed Design", *Brigham Young University PHD Dissertation*

15. Cochran, Jeffery K., Hong, Suck-Chul (2004), "An Automated Relational Database Structure Building System for the Life Cycle of Manufacturing Simulation Models", *International Journal of Computer Applications in Technology*, vol. 19, pp. 53-65.

16. Barbosa, C.A.M., et. al. (2002), "An Object Model for Collaborative CAD Environments", *Proceedings of the International Conference on Computer Supported Cooperative Work in Design*, vol. 7, pp. 179-184.

17. Ridjanovic, D., M. L. Brodie, (1982) "Semantic Data Model-Driven Design, Specification and Verification of Interactive Database Transactions," *Computer Corporation of America*, April.

18. Peng, Ting-Kuo, Trappey, Amy J. C. (1996), "Cad-Integrated Engineering-Data-Management System for Spring Design", *Robotics and Computer-Integrated Manufacturing*, vol. 12, pp. 271-281.

19. Baker, Tyson J. (2004), "Attribution Standardization for Integrated Concurrent Engineering", *Brigham Young University Master's Thesis*.

20. Bowland, N.W., Gao, J.X., Sharma, R. (2003), "A PDM- and CAD-integrated Assembly Modelling Environment", *Journal of Materials Processing Technology*, vol. 138, pp. 82-88.

21. Carroll, Michael P., Hawkins, Christopher M. (2002), "Web Based Analysis", *ACM Digital Library*.

22. Oh, Youchon, Han, Soon-hung, Suh, Hyowon (2001), "Mapping Product Structures Between CAD and PDM Systems Using UML", *Computer-Aided Design*, vol. 33, pp. 521-529.

23. Gao, J.X., Aziz, Hayder, et al (2003), "Application of Product Data Management Technologies for Enterprise Integration", *International Journal of Computer Integrated Manufacturing*, vol. 16, pp. 491-500.

24. Basset, Kim (1997), "Remaking Enterprise Workflow: When PDM & Document Management Really Meet", *Advanced Imaging*, vol. 12, pp. 16-18.

25. Fife, Nathaniel L. (2005), "Developing a Design Space Model Using a Multidisciplinary Design optimization Schema in a Product Lifecycle Management System to Capture Knowledge for Reuse", *Brigham Young University Master's Thesis*.

26. Lund, J., Fife, N., Jensen, C., "PLM-Based Parametrics for Design Automation and Optimization", *Computer-Aided Design and Applications*, Vol. 2, Nos. 1-4, CAD'05, 2005

27. Teare, S. "A procedure for Structuring Product Information for Reusability", *Master's Thesis BYU*, 2000.

28. Anon. "PLM: What does it mean? What do you want it to mean?" *British Plastics and Rubber*, n MAY, pages 35-36, May 2004.

29. www.ugs.com, Posted Sept. 2004.

30. Johnson, C. Gavilanes, J. "Quick and below budget", *InTech*, v 50 n 5, pages 51-54, May 2003.

31. Rahse, Wilfried, Hoffmann, Sandra "Product design - Interaction between chemistry, technology and marketing to meet customer needs" *Chemical Engineering and Technology*, v 26, n 9, pages 931-940, September, 2003.

32. Wang, L.Shen W. Xie H. Neelamkavil, P. A. "Collaborative conceptual design - State of the art and future trends", *CAD Computer Aided Design*, v 34 n13, pages 981-996, November 2002.

33. Sobieszczanski-Sobieski, J. Tulinius, J. "MDO can help resolve the designer's dilemma", *Aerospace America*, v 29 n 9, pages 32-35, 63, September 1991.

34. Stelmack, Marc A., Batill, Stephen M., Beck, Bryan C. "Design of an aircraft brake component using an interactive multidisciplinary design optimization framework", *Journal of Mechanical Design, Transactions of the ASME*, v 122, n 1, pages 70-76, March 2000.

35. Berger, "Continuous improvement and kaizen: Standardization and organizational designs" *Integrated Manufacturing Systems*, v 8, n 2, pages 110-117, 1997.

36. Kulhavy, Rudolf. "Data-centric decision support". *Proceedings of the American Control Conference*, v 4, pages 3395-3400, 2002.

37. Pine, J.B., 1993, “Mass Customizing Products and Services,” *Planning Review*, Vol. 21, No 4, pp. 6-13.

38. Roach, Gregory, “The Product Design Generator—A Next Generation Approach To Detailed Design”, Brigham Young University, Dissertation, August 2003.

39. Hogge, D, “Integrating Commercial CAx Software to Perform Multidisciplinary Design Optimization”, *Master’s Thesis BYU*, 2002.

40. Tappeta, R.V., Nagendra, S., Renaud, J.R. “Multidisciplinary design optimization approach for high temperature aircraft engine components”, *Structural Optimization*, v 18, n 2-3, pages 134-145Oct, 1999.

41. Carlson, P. "A distributed computing system used for concurrent optimization methods on a violin top", *Structural and Multidisciplinary Optimization*, v 25 n 5-6, pages 453-458, December 2003.

42. Sobieszczanski-Sobieski, J. "Multidisciplinary design optimization (MDO) methods: Their synergy with computer technology in the design process", *Aeronautical Journal*, v 103 n 1026, pages 373-382, August 1999.

43. Numata J. Lei, B. Iwashita, Y. “Information management for knowledge amplification in virtual enterprise”. *IEEE International Engineering Management Conference, Managing Virtual Enterprises: A Convergence of Communications, Computing, and Energy Technologies.* pages 281-285, 1996.

44. Huang G. Lee S. Mak K. “Web-based product and process data modeling in concurrent ‘design for X’”. *Robotics and Computer-Integrated manufacturing*; v 15 n 3, pages 53-63, June 1999.

45. Caldwell, N. Rodgers, P. “WebCADET: Facilitating distributed design support”. *London, UK: IEE Colloquium on Web-based Knowledge Servers*. n 307, pages 9/1-9/4, 1998

46. Zdrahal Z, Domingue J. “The World Wide Design Lab: an environment for distributed collaborative design”. *In Proceedings of International Conference on Engineering Design*, Tampere, Aug. 19-21, 1997.

47. Klein M. “Capturing geometry rationale for collaborative design”. *In Proceedings of the IEEE Workshops on Enabling Technologies Infrastructure for Collaborative Enterprises (WET ICE’97*). Pages 24-28, 1997.

48. Parunak, H. “What can agents do in industry, and why? AN overview of industrially-oriented R&D at CEC, Cooperative information agents II: learning, mobility and electronic commerce for information discovery on the Internet”. *In:*

*Klusch M, Weiss G, editors. Second International Workshop, CIA'98, Paris, France: Springer.* pages 1-18, 1998.

49. Cutkosky, M. Engelmore, R. Fikes, R. Genesereth, M. Bruber, T. Mark, W. Tenenbaum, J. Weber, J. "PACT: An experiment in integrating concurrent engineering systems". *IEEE Computer*; v 26(1), pages 28-37, 1993.

50. Campbell, M. Cagan, J. Kokvsky, K. "A-Design: an agent-based approach to concept design in a dynamic environment". *Research in Engineering Design* v 11, pages172-192, 1999.

51. Varma, A. Dong, A. Childambaram, B. Agogino, A. Wood, W. "Web-based tool for engineering design", *Working Paper*, 1999.

52. Toye, G. Cutkosky, M. Leifer, L. Tenenbaun, J. Glicksman, J. "SHARE: A methodology and environment for collaborative product development". *In proceeding of Second Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE Computer Society Press*. Pages 33-47, 1993.

53. http://www.engineous.com/product_FIPER.htm, Posted Feb. 2005.

54. Wang, Y. D. Shen, W. Ghenniwa, H. "WebBlow: A Web/agent-based multidisciplinary design optimization", *Computers in Industry*, v 52 n 1, pages 17-28, September 2003.

55. Klaas, O. "Embedding reliable numerical analysis capabilities into an enterprise-wide information system", *Engineering with Computers*, v 17 n 2, pages 151-161, 2001.

56. Fife, N. "Developing a Design Space Model Using a Multidisciplinary Design Optimization Schema in a Product Lifecycle Management System" *Master's Thesis BYU*, 2005.

57. Goldfarb, D. Idnani, A. "A Numerically Stable Dual Method for Solving Strictly Convex Quadratic Programs" *Math Programming*, v 27, pages 1-33, 1983.

58. Kim, Y. Kang, S. Lee, S. and Yoo, S. "A distributed, open, intelligent product data management system". *International Journal of Computer Integrated Manufacturing*, v 14 n 2, pages 224–235, March/April 2001.

59. Koonce, D. "A hierarchical cost estimation tool", *Computers in Industry*, v 50 n 3, pages 293-302, April 2003.

60. Alexandrov, N. M. Lewis, R. "Analytical and computational aspects of collaborative optimization for multidisciplinary design", *AIAA Journal*, v 40 n 2, pages 301-309, February 2002.

61. Robinson, C. "Good goals lead to better data", *Machine Design*, v 66 n 10, pages 51-60, May 1994.

62. Meade, L. Presley, A. Rodgers, K. "Tools for engineering the agile enterprise", *IEEE International Engineering Management Conference, Managing Virtual Enterprises: A Convergence of Communications, Computing*, pages 381-385, 1996.

63. Aziz, H. Gao, J. Maropoulos, P. Cheung, W. "Application of product data management technologies for enterprise integration", *International Journal of Computer Integrated Manufacturing*, v 16 n 7-8, pages 491-500, October/December 2003.

64. Rogers PA, Huxor AP, Caldwell. “Design support using distributed Web-based AI tools”. *Research in Engineering Design*, v 11, pages 31-44, 1999.

APPENDIX

# APPENDIX A: PPLM SOCKET EXAMPLE—ANSYS

```
} else if (strcmp(this->function, "ansys")==0) {
                        // ansys

printf("Running ANSYS...\n");
char cmd[256];
sprintf(cmd,"C:\\Progra~1\\AnsysI~1\\v81\\ANSYS\\bin\\intel\\ansys81.exe -b -p ansysrf -j
JobName -i %s -o ansys.out", datasetPath);

STARTUPINFO si;
PROCESS_INFORMATION pi;

ZeroMemory( &si, sizeof(si) );
si.cb = sizeof(si);
ZeroMemory( &pi, sizeof(pi) );

printf("Launching process \"%s\"", cmd);

if( !CreateProcess(                         // LAUNCH CLIENT PROCESS
                        NULL,               // No module name (use command line).
                        TEXT(cmd),          // Command line.
                        NULL,               // Process handle not inheritable.
                        NULL,               // Thread handle not inheritable.
                        FALSE,              // Set handle inheritance to FALSE.
                        0,                  // No creation flags.
                        NULL,               // Use parent's environment block.
                        NULL,               // Use parent's starting directory.
                        &si,                // Pointer to STARTUPINFO structure.
                        &pi ) ) {           // Pointer to PROCESS_INFORMATION structure.
        printf("proc_launch error <%d>", GetLastError());
        CALL (9999999);
}


//----------------------------------------
// run the socket server
//----------------------------------------


WSADATA wsaData;
SOCKET first_socket;
sockaddr_in service;
SOCKET AcceptSocket;
DWORD dwThrdParam = 1;

int iResult = WSAStartup( MAKEWORD(2,2), &wsaData );
if ( iResult != NO_ERROR ) {
        printf("DFMSRV: **ERROR: DFM data server startup failure: (WSAStartup)\n");
        CALL ( 99999 );
}


first_socket = socket( AF_INET, SOCK_STREAM, IPPROTO_TCP );
if ( first_socket == INVALID_SOCKET ){
```

```
        printf( "DFMSRV: **ERROR: DFM data server startup failure: (socket failure:
%ld)\n", WSAGetLastError() );
        WSACleanup();
        CALL(99999);
}

service.sin_family = AF_INET;
service.sin_addr.s_addr = inet_addr( "127.0.0.1" );
service.sin_port = htons( DFM_DATA_PORT );


if ( bind( first_socket, (SOCKADDR*) &service, sizeof(service) ) == SOCKET_ERROR ) {
        closesocket(first_socket);

        printf("DFMSRV: **ERROR: DFM data server startup failure: (bind failed).\n" );
        CALL ( 99999 );
}

if ( listen( first_socket, 1 ) == SOCKET_ERROR ) {
        printf( "DFMSRV: **ERROR: DFM data server startup failure: (listen failure).\n");
        CALL ( 99999 );
}

AcceptSocket = SOCKET_ERROR;
printf("DFMSRV: Waiting for a client to connect.\n");
while ( AcceptSocket == SOCKET_ERROR )
        AcceptSocket = accept( first_socket, NULL, NULL );

SOCKET m_socket = AcceptSocket;
printf("DFMSRV: Accepted socket #: %d\n", m_socket);
char sendbuf[CUSTOMBUFSIZE];
char recvbuf[CUSTOMBUFSIZE];
int bytesSent;
int bytesRecv = SOCKET_ERROR;

while (true){
        memset( recvbuf, 0, CUSTOMBUFSIZE );
        memset( sendbuf, 0, CUSTOMBUFSIZE );
        printf("DFMSRV: waiting for bytes to receive\n");
        bytesRecv = recv( m_socket, recvbuf, CUSTOMBUFSIZE-1, 0 );
                                                        // receive input
        if(bytesRecv==SOCKET_ERROR || bytesRecv == WSAECONNRESET || *recvbuf==0 ){
                break;
        }
        char* tochange = strstr(recvbuf,"\n");
        if(tochange != NULL)
                *tochange = '\0';
        tochange = strstr(recvbuf,"\r");
        if(tochange != NULL)
                *tochange = '\0';
        printf("DFMSRV: client said: \"%s\"\n", recvbuf);

        int count      = 0;

        //-----------------------------------------------------------------
        // process the command          (process_cmd(recvbuf, sendbuf))
        //-----------------------------------------------------------------

        std::vector<char*>    strings = splitString(recvbuf, ":");
        int                                   client_pid = 0;
        count          = strings.size();

        sprintf(sendbuf,"WARNING, no return value");
                        // set the default error, (it will be changed if non-error)

        if (count > 3) {
                client_pid = atoi(strings[0]);
        }

        if (count == 3) {
```

```
                if (strcmp(strings[1],"getvalue")==0) {
                        char* tmp = NULL;
                        for(int i=0;i<names->length;i++) {
                                tmp = names->get_value_at(i);
                                if(strcmp(tmp,strings[2])==0) {
                                        char* val = values->get_value_at(i);
                                        printf("Found variable %s, sending %s.\n",
strings[2], val);
                                        sprintf(sendbuf,"%s",val);
                                        SAFEDELARR ( tmp );
                                        SAFEDELARR ( val );
                                        break;
                                }
                                SAFEDELARR ( tmp );
                        }
                }

        } else if (count > 3) {
                static int numset=0;
                if(strcmp(strings[1],"setvalue")==0) {
                        onames->set_value_at(strings[2],numset);
                        ovalues->set_value_at(strings[3],numset);
                        numset++;
                        sprintf(sendbuf," ");
                }
        } else {
                printf("Unrecognized communication \"%s\"\n", recvbuf);
        }


        freeStrings(strings);

        if(count<2)
                sprintf(sendbuf, "DFMSRV: too little input received.");

        printf("DFMSRV: sending \"");printf(sendbuf);printf("\"\n");
        sprintf(sendbuf,"%s\n", sendbuf);
        bytesSent = send( m_socket, sendbuf, strlen(sendbuf), 0 );  // send result
}

printf("closing sockets..\n");
closesocket(m_socket);
closesocket(first_socket);
printf("done.\n");

WaitForSingleObject( pi.hProcess, INFINITE );

// Close process, thread, and server handles.
CloseHandle( pi.hProcess );
CloseHandle( pi.hThread );

printf("end Socket Server\n");
printf("Loading dataset...\n");
tag_t newds =  import_to_dataset ("Image", "ANSYS Report", "no description",
                        "Image", "results.jpg", 1, SS_BINARY, "IExplore");
this->form->data->getField("output_refs")->set_ref_at(newds,0);


} else if (strcmp(this->function, "csd")==0 ) {
```

# APPENDIX B: SOCKET API SYNTAX

```
  client commands:

        (all 'form' values are integers, representing the tag)


        - PID:getinfields
                input values:
                        - pid:          process id
                return values:
                        - fields:       "field1","field2","field3"

        - PID:getvalue:field
                input values:
                        - pid:          process id
                        - field:        field to get
                return values:
                        - single:       value
                        - array:        value_1:value_2:value_n
                                                        (":"-delimited)
                        - 2Darr:
        value_1a,value_1b,value_1c:value_2a,value_2b,value_3b                   (":"-
delimited, inner ","-delimited)

        - PID:setvalue:field:value
                input values:
                        - pid:          process id
                        - field:        field to set
                        - value:        value to set field
                value syntax:
                        - single:       value
                        - array:        value_1:value_2:value_n
                                                        (":"-delimited)
                        - 2Darr:
        value_1a,value_1b,value_1c:value_2a,value_2b,value_3b                   (":"-
delimited, inner ","-delimited)

        - PID:getfile:saveAs
                input values:
                        - pid:          process id
                return value:
                        - filelist:     "file1:file2:file3"

        - PID:savefile:saveAs
                input values:
```

# APPENDIX C: SOCKET API .DLL IMPLEMENTATION

```
#include "dfm_data.h"

int DFM_Data::sendsomething(char* message) {

        int bytesSent;
        int bytesRecv = SOCKET_ERROR;
        memcpy(sendbuf, message, CUSTOMBUFSIZE);
        sendbuf[CUSTOMBUFSIZE-1] = '\0';


        printf("CLIENT: Sending %s\n", sendbuf);
        bytesSent = send( m_socket, sendbuf, strlen(sendbuf), 0 );
        bytesRecv = recv( m_socket, recvbuf, CUSTOMBUFSIZE, 0 );
        recvbuf[CUSTOMBUFSIZE-1] = '\0';
        printf("CLIENT: Server said: ***%s***\n", recvbuf );

        return 0;
}

int DFM_Data::dfm_close(void) {
        closesocket(m_socket);
        WSACleanup();
        Sleep(1000);
                                                        // give the server time to close
        return 0;
}


int DFM_Data::dfm_connect(void)
{

        int iResult = WSAStartup( MAKEWORD(2,2), &wsaData );
        if ( iResult != NO_ERROR )
                printf("CLIENT: Error at WSAStartup()\n");


        m_socket = socket( AF_INET, SOCK_STREAM, IPPROTO_TCP );



        if ( m_socket == INVALID_SOCKET )
        {
                printf("CLIENT: Error at socket(): %ld\n", WSAGetLastError() );
                WSACleanup();
                return 1;
        }

        if( gethostname ( name, sizeof(name)) == 0)
        {
               if((localhostinfo = gethostbyname(name)) != NULL)
               {
                       my_ip = inet_ntoa (*(struct in_addr *)*localhostinfo->h_addr_list);
               }
```

```
        }

        service.sin_family = AF_INET;
        service.sin_addr.s_addr = inet_addr( "127.0.0.1" );//my_ip );
        service.sin_port = htons( DFM_DATA_PORT );


        if ( connect( m_socket, (SOCKADDR*) &service, sizeof(service) ) == SOCKET_ERROR)
        {
                sprintf(recvbuf,"CLIENT: Failed to connect to DFM server.\n");
                printf("%s\n", recvbuf);
                WSACleanup();
                return 1;
        }
        Sleep(1000);

        return 0;
}
```

# APPENDIX D: CLIENT USAGE OF THE C++ SOCKET API LIBRARY—UNIGRAPHICS

```
#include "NX_DFM.h"

extern int main( char argc, char *argv[] )
{
        UF_initialize();
        tag_t part;



        DFM_Data* TCdata = new DFM_Data();

        char** fields = NULL;
        int field_count = 0;
        int number_of_exps     = 0;
        int i                          = 0;
        tag_t *exps = NULL;

        field_count = TCdata->getinfields(fields);

        UF_PART_load_status_s tmp;
        UF_PART_open("c:\\demo\\demo.prt", &part, &tmp);

        for(i=0;i<field_count;i++) {
                unsigned char is_exp_in_part = 0;
                UF_MODL_is_exp_in_part (part, fields[i], &is_exp_in_part);
                if(!is_exp_in_part)
                        continue;
                char new_expr[124];
                char** field_value = NULL;
                TCdata->getvalue(fields[i],field_value);

                sprintf(new_expr, "%s=%s", fields[i], field_value[0]);
                printf("new expression: %s\n", new_expr);
                UF_MODL_edit_exp (new_expr);
                printf("Field[%d] = %s\n", i,fields[i]);
                delete [] field_value[0];
                delete [] field_value;
        }

        UF_PART_save();
        UF_PART_close(part,1,1);

        printf("NX Cleaning up...\n");
        delete TCdata;

        UF_terminate();
        printf("end NX_DFM\n");

        return 0;
}
```

# APPENDIX E: TCL SOCKET API

```
namespace eval DFM {
  variable myPID [pid]
  variable channel
  variable serverHost "127.0.0.1"
  variable serverPort "27016"
  variable arg
  variable field
  variable valueToSet
  variable luke
  variable msg

}

proc debug {arg} {
  puts $arg
}

proc DFM::getArg { arg } {
  set DFM::arg $arg
  set DFM::msg "$DFM::myPID:getarg:$DFM::arg"
  DFM::msgSend
  vwait DFM::luke
}

proc DFM::getValue { arg } {
  set DFM::arg $arg
  set DFM::msg "$DFM::myPID:getvalue:$DFM::arg"
  DFM::msgSend
  vwait DFM::luke
}

proc DFM::setValue { field valueToSet } {
  set DFM::field $field
  set DFM::valueToSet $valueToSet
  set value [ans_getvalue PARM,$valueToSet,VALUE]
  set DFM::msg "$DFM::myPID:setvalue:$DFM::field:$value"
  DFM::msgSend
}

proc DFM::msgSend { } {
  puts "TCL client sent <<<$DFM::msg>>>"
  puts $DFM::channel "$DFM::msg"; flush $DFM::channel
}

proc DFM::msgHandler { } {
  set data [gets $DFM::channel]; flush $DFM::channel
  puts stdout "TCL client recieved <<<$data>>> for message <<<$DFM::msg>>>"; flush stdout
  if { [string first "setvalue" $DFM::msg] != -1 } {
    set data [gets $DFM::channel]; flush $DFM::channel
  } elseif { [string first "getarg" $DFM::msg] != -1 } {
        ans_sendcommand *set,$DFM::arg,$data
  } elseif { [string first "getvalue" $DFM::msg] != -1 } {
        ans_sendcommand *set,$DFM::arg,$data
```

```
  }
  set DFM::luke ready
  unset DFM::luke
}

proc DFM::closeConnetion { } {
        puts "Closing TCL client"
   catch {close $DFM::channel}
}

#open get socket
if {[catch {socket $DFM::serverHost $DFM::serverPort} DFM::channel]} {
  puts stdout "Failed to connect to server at $DFM::serverHost $DFM::serverPort"; flush
stdout
  exit
} else {
  fconfigure $DFM::channel -blocking 0
  fileevent $DFM::channel readable "DFM::msgHandler"
}
set data [gets $DFM::channel]; flush $DFM::channel
```