Development of Heterogeneous CAD Assembly Tools for Collaborative Design

By

Kristie Yu

Bachelor of Engineering in Mechanical Engineering
Cooper Union for the Advancement of Science and Art, New York, NY 10003

Master of Science in Mechanical Engineering
Massachusetts Institute of Technology, Cambridge, MA 02139

Submitted to the Department of Mechanical Engineering
In Partial Fulfillment of the Requirements for the Degree of

Mechanical Engineer
at the
Massachusetts Institute of Technology

May 2002

© 2002 Massachusetts Institute of Technology, All Rights Reserved

Signature of Author......................................................

Department of Mechanical Engineering
May 10, 2002

Certified by.................................

David Wallace
Esther and Harold E. Edgerton Associate Professor of Mechanical Engineering
Thesis Supervisor

Accepted by......................................................

Ami A. Sonin
Chairman, Department Committee on Graduate Studies
Development of Heterogeneous CAD Assembly Tools for Collaborative Design

By

Kristie Yu

Submitted to the Department of Mechanical Engineering on May 10, 2002 in Partial Fulfillment of the Requirements for the Degree of Mechanical Engineer in Mechanical Engineering

ABSTRACT

The next generation of CAD plug-ins for DOME\textsuperscript{3} was developed for use in the heterogeneous CAD assembly design process originated from the MIT CADlab. This process utilizes DOME in exchanging assembly level parameters to remote native CAD models and coordinating the exchange of up-to-date neutral files. In addition, a VRML and Java based graphic user interface for assisting heterogeneous CAD assembly designers in designing assemblies through DOME was also developed. Among its features are the abilities to traverse native and imported components through a Swing assembly tree and access their design parameter information, set constraints on design parameters, modify design parameter values, and arrange design configurations on a per component basis. A case study on the design of half a rear suspension system, composed of an imported STEP flange assembly, was performed utilizing the tools developed for this thesis. Results indicate that the developed DOME CAD plug-ins and DOME VRML module have great potentials for current and future applications.
Table of Contents

Acknowledgements.......................................................................................... 4
List of Figures................................................................................................... 5
List of Tables................................................................................................... 9
1 Introduction
   1.I. Background..........................................................................................10
   1.II. Literature Review and Commercial Solutions......................................15
   1.III. Thesis Objective.............................................................................23
2 New Assembly Process
   2.I Introduction..........................................................................................28
   2.II. Linking CAD Plug-Ins to DOME.....................................................34
   2.III. Basic Plug-In Functionalities and Requirements.................................34
       2.III.A. Exporting Components..........................................................37
       2.III.B. Importing Components..........................................................43
   2.IV. Unigraphics Plug-In...........................................................................45
       2.IV.A. Unigraphics Plug-In Interface Specification............................49
       2.IV.B. Import Issues Specific to Unigraphics......................................54
   2.V. I-deas Plug-In.....................................................................................57
       2.V.A. I-deas Plug-In Interface Specification......................................58
       2.V.B. Import Issues Specific to I-deas..............................................63
   2.VI. Summary and Conclusions..................................................................65
3 DOME VRML Module

3.I. Introduction.............................................................................................. 69
3.II. Overview of New Features Offered........................................................ 69
3.III. VRML Module Functionalities and Requirements............................... 70
  3.III.A. The Design Interface........................................................................ 73
  3.III.B. The Component Manager................................................................. 82
  3.III.C. The Configuration Manager............................................................. 86
3.IV. Required VRML Components................................................................. 91
3.V. Summary and Conclusions....................................................................... 97

4 Case Study

4.I. Introduction.................................................................................................. 99
4.II. Supplier Preparation Process: Setting Up the Flange Assembly.............. 101
4.III. Assembly Designer Preparation Process: Setting Up the
      Half Rear Suspension Assembly................................................................. 107
4.IV. Rear Suspension Assembly Designer: Working With the
      DOME VRML Module................................................................................ 112
4.V. Summary and Conclusions........................................................................ 138

5 Conclusions and Future Work

5.I. Conclusions.................................................................................................. 140
5.II. Future Work.............................................................................................. 142

6 References....................................................................................................... 143
Acknowledgements

I’d like to thank my family for their continued support and guidance. Without them, I wouldn’t be the person I am today. Thanks for all your wonderful words of encouragements and for always believing in me. Thanks to Prof. David Wallace for giving me the wonderful opportunity to work at the MIT CADlab and be a part of something special. Thanks to Ford and Mr. Peter Sferro for sponsoring my research and proving me with the opportunity to work on an industry problem. Thanks to Elaine for being not only a mentor but also a great friend. And a big thanks to the following members of the CADlab family for being such an awesome, supportive, fun, and smart bunch: Aubrey, Charles, Ed, Ines, Jaehyun, Keiko, Maureen, Prabhat, Qing, Sane, and Twiggy. Special thanks to Twiggy for always bringing the best snacks to the lab so I can munch on them while I work and Sane for hanging out with me at the movies. I wish you all the best of health and prosperity for years to come.
List of Figures

Figure 2.1 Designers Working In Their Respective CAD Systems..................29
Figure 2.2 Assembled Bolted Joint Assembly.............................................29
Figure 2.3 Working With DOME And Establishing Relevant Relations..............30
Figure 2.4 Increasing the Length and Thickness of the Plate..........................31
Figure 2.5 Propagating Changes in Design Parameters From the Assembly Designer to the Suppliers.................................................................32
Figure 2.6 Updated Bolted Joint Assembly..................................................33
Figure 2.7 CAD Plug-In Initialization Information Flow Diagram....................36
Figure 2.8 Sample “density.txt” file........................................................41
Figure 2.9 Sample “import.txt” file........................................................44
Figure 2.10 Unigraphics CAD System.........................................................46
Figure 2.11 Unigraphics DOME Version 1 Interface.....................................47
Figure 2.12 DOME VRML Module..........................................................48
Figure 2.13 I-deas CAD System...............................................................57
Figure 2.14 CAD Plug-In Export Interaction Information Flow Diagram.............66
Figure 2.15 CAD Plug-In Import Interaction Information Flow Diagram.............67
Figure 3.1 DOME VRML Interface..........................................................72
Figure 3.2 The VRML Module Design Interface.........................................73
Figure 3.3 Sample Part Component Data File.............................................75
Figure 3.4 Selecting Modifiable Dimension Objects through the VRML Module...76
Figure 3.5 Entering Invalid Values for the Selected Modifiable Dimension.........77
Figure 3.6 Setting Limits on a Design Parameter ........................................ 78
Figure 3.7 Entering Limit on a Design Parameter ........................................ 79
Figure 3.8 Hiding Dimension Objects ......................................................... 80
Figure 3.9 Exceeding Design Parameters .................................................... 81
Figure 3.10 The VRML Module Component Manager ................................. 83
Figure 3.11 Selecting Component Using the Pull Down Menu ....................... 84
Figure 3.12 The VRML Module Configuration Manager .............................. 87
Figure 3.13 Recording a Design Configuration ............................................ 88
Figure 3.14 Recalling a Design Configuration ............................................. 89
Figure 3.15 Deleting a Saved Configuration ............................................... 90
Figure 3.16 Ordering Saved Configurations ............................................... 91
Figure 3.17 VRML Module and CAD Plug-In Information Flow Diagram ....... 97
Figure 4.1 Flange Assembly ................................................................. 101
Figure 4.2 Seal Assembly ................................................................. 101
Figure 4.3 Outer Seal ................................................................. 102
Figure 4.4 Inner Seal ................................................................. 102
Figure 4.5 Flange Fin ................................................................. 102
Figure 4.6 Flange Assembly “density.txt” File ........................................ 103
Figure 4.7 Unigraphics CAD Plug-In “flange_assembly.in” Input File .......... 104
Figure 4.8 Flange Assembly “flange_assembly.txt” File ................................ 105
Figure 4.9 Half Rear Suspension Assembly ............................................ 106
Figure 4.10 Rear Suspension Designer’s “import.txt” File ......................... 107
Figure 4.11 Assembled Half Rear Suspension ........................................... 109
Figure 4.12 I-deas CAD Plug-In “Rear_Suspension.in” Input File..................... 110
Figure 4.13 DOME VRML Module’s “Rear_Suspension.dat” Input File............. 111
Figure 4.14 DOME VRML Interface for Half Rear Suspension Assembly......... 113
Figure 4.15 Setting Limits on Design Parameters........................................ 114
Figure 4.16 Entering Value for the Design Parameter Bounding Limit............ 114
Figure 4.17 Viewing The Bounding Limit Just Set....................................... 115
Figure 4.18 Component Manager Pull Down Menu....................................... 116
Figure 4.19 Expanded Component Manager Swing Assembly Tree.................. 116
Figure 4.20 Expanded Component Manager Swing Assembly Tree.................. 117
Figure 4.21 Carrier Fin.............................................................................. 118
Figure 4.22 Selecting a Dependent Dimension Design Parameter.................... 119
Figure 4.23 Selecting a Modifiable Dimension Design Parameter................... 119
Figure 4.24 Changing a Modifiable Dimension Design Parameter.................... 120
Figure 4.25 Updated VRML Scene and Design Interface for Carrier Fin.......... 121
Figure 4.26 Rear Auxiliary Suspension......................................................... 122
Figure 4.27 Recording a Design Configuration............................................ 122
Figure 4.28 Recalling a Design Configuration............................................. 123
Figure 4.29 Modifying a Dimension Design Parameter.................................. 124
Figure 4.30 Updated VRML Scene and Design Interface for Rear Auxiliary Suspension.......................................................... 124
Figure 4.31 Recording a Design Configuration............................................ 125
Figure 4.32 Arranging Stored Design Configurations................................... 126
Figure 4.33 Resulting Configuration Manager............................................ 126
Figure 4.34 Deleting a Saved Design Configuration.............................. 127
Figure 4.35 Remaining Configuration Saved For the Rear Auxiliary Suspension.... 128
Figure 4.36 Flange Assembly.............................................................. 129
Figure 4.37 Selecting Inner Seal within the Flange Assembly..................... 130
Figure 4.38 Inner Seal..................................................................... 130
Figure 4.39 Outer Seal...................................................................... 131
Figure 4.40 Seal Assembly................................................................. 131
Figure 4.41 Flange Fin..................................................................... 132
Figure 4.42 Modifying the Design Parameter of Imported Component Flange Fin.. 133
Figure 4.43 Updated Flange Fin.......................................................... 134
Figure 4.44 Updated Flange Assembly............................................... 134
Figure 4.45 Updated Half Rear Suspension Assembly.............................. 135
Figure 4.46 Exceeding Design Parameter Bounds for Components.............. 136
Figure 4.47 Flange Fin..................................................................... 137
List of Tables

Table 2.1 Sample DOME Unigraphics Plug-In Interface Specification Files........ 52
Table 2.2 Sample DOME I-deas Plug-In Interface Specification Files.............. 61
Chapter 1: Introduction

I. Background

In the 1970’s, the U.S. automotive industry sought to protect its market share from imports by improving its own productivity and competitiveness through outsourcing components and concurrent engineering. As a result, lead times were reduced and the productivity gap significantly narrowed between the U.S. automotive industry and its competitors [Brunnemeier and Martin 1999].

The automotive supply chain evolved from vertically integrated organization to consist of original equipment manufacturers (OEM), first tier suppliers, sub-tier suppliers, and infrastructure suppliers. However, as the nature of the relationship between the involved parties became more complex, such as when a particular company take on business with different OEMs within the supply chain, the ability to share computer aided design (CAD) data during product design and manufacturing becomes increasingly more important. Many companies use CAD systems for communicating product design intent. Unfortunately, due to a lack of common standard for CAD model formats, companies that manufacture various components of an end-product are not able to seamlessly transfer parametrically editable data up and down the organizational structure [Kelly 2001].

In a supply or design chain, such as the automotive, many different software and hardware systems are used during the design process. Because each system has its own
proprietary data representation, product data are created and stored in multiple, incompatible formats, which makes exchanging these data difficult and often results in data files that may contain error, be incomplete, or be formatted in a way that makes them unusable for downstream applications [Brunnemeier and Martin 1999]. This contributes to the problem of CAD interoperability.

Studies have shown CAD interoperability costs the automotive industry roughly 1 billion dollars annually [Brunnemeier and Martin 1999] and the discrete manufacturing industry roughly 20 billion [PTC 2001]. For the automotive industry, there are three different types of costs related to interoperability problems. They are avoidance, mitigating, and delay costs.

Avoidance costs are incurred when automakers attempt to prevent interoperability problems before they can occur. They include: the cost of purchasing, maintaining, and training for CAD/CAM systems for the purpose of native format translation; the cost of purchasing, maintaining, and training of point-to-point translation software; the cost of purchasing, maintaining, and training for neutral format translation software; outsourcing costs incurred when outside companies are hired to provide data exchange services; investments in in-house programs aimed at addressing interoperability issues, such as implementing STEP or training engineers in proper product model creation; and, the cost of participating in industry consortia activities aimed at improving interoperability through the industry [Brunnemeier and Martin 1999].

Mitigating costs are incurred when automakers attempt to address interoperability problems after they have occurred. They include: the cost of reworking scrapped models, designs, prototypes, parts, dies, etc., that were incorrect due to interoperability problems;
and, the cost of manually reentering data when other methods of data exchange are unavailable or unsatisfactory [Brunnemeier and Martin 1999].

Delay costs are incurred when automakers fail to meet production schedule due to interoperability problems. They include: profits lost due to decline in market share caused by delays; profits lost due to delay of revenues; and, losses of consumer welfare due to delay of the availability of products with greater net value [Brunnemeier and Martin 1999].

The main cause of interoperability problems stem from a lack of standardized CAD format within a given industry. The lack of a standardized CAD format can be attributed to one important necessity, the modeling kernel that is the brains of the CAD application [Kelly 2001]. Within an existing industry, CAD applications can be created from a proprietary kernel or from a commercially available kernel [Kelly 2001]. For example, Structural Dynamics Research Corporation’s (SDRC) I-deas, Parametric Technology Corporation’s (PTC) Pro/Engineer, and Dassault Systemes’ Catia all use their own proprietary kernels. While UGS’s Unigraphics uses UGS’s Parasolid kernel and SolidWorks uses Spatial Technology Corporation’s ACIS kernel. Both Parasolid and ACIS are commercially available kernels.

In an attempt to combat the interoperability problem, OEMs and suppliers have tried to standardize their CAD packages. Point-to-point and neutral format translators were also developed. Standardizing CAD systems across OEM and suppliers does not necessarily eliminate interoperability problems since they would surface when the supplier attempts to exchange CAD data with another OEM supplier operating under a different CAD system. Point-to-point translators work fine for some cases, but requires
constant updates to keep up with the latest version of CAD packages. Also, specific
translators would be required based on which CAD system is doing the sending and
reading.

Neutral format translators such as the Initial Graphics Exchange Specification
(IGES) and the Drawing eXchange Format (DXF) were limited in their success in their
early days, but have improved significantly with newer versioning standards. Recently
adopted by the International Standards Organization (ISO) and developed by more than
38 countries, the Standard for the Exchange of Product Data Model (STEP) translator
was developed to extend data exchange capabilities to all aspects of a product’s life
cycle, from material specification to after-sale maintenance [Brunnemeier and Martin
1999]. Various tests conducted by industry participants have shown that the STEP
translator outperforms both IGES and DXF translators.

It should be noted here that the IGES, DXF, and STEP neutral file standards are
used in the automotive industry. In other industries, such as the Architecture,
Engineering, and Construction (AEC) industry, different neutral format standards are
used. Cadalyst has a very interesting article on six AEC software vendors’ views on the
CAD interoperability issue within their industry [Cadalyst 2001].

The key advantage of using neutral format translators is that it protects proprietary
knowledge in the way the detailed models are built. However the key disadvantage of
using neutral, tessellated files is that the geometry is not parametrically editable. That,
coupled with versioning issues between the OEM and supplier, may adversely add to
design iteration cycle times. Hence, if the neutral format solution is to be used, there
needs to exist an approach that will compensate for its limitations by eliminating
versioning issues and speeding up design iteration cycles for a heterogeneous CAD assembly composed of native and neutral file components.

As section II will indicate, considerable research has been devoted to the area of CAD interoperability and numerous solutions proposed. It is not the intent of this thesis to propose a solution that will completely eliminate all problems associated with CAD interoperability. It is rather, the intent of this thesis to propose and present several tools developed at the MIT CADlab for use in assisting assembly engineers in the design of heterogeneous CAD assemblies that may help in reducing cost associated with CAD interoperability.

One of the major problems associated with CAD interoperability through neutral files is its impediment on the rapid design iteration cycles desired of an assembly composed of parts from several different CAD systems. There are traditionally two ways in which such an assembly is constructed: bringing a directly converted version of the model from the sender’s system to the receiver’s system, or importing a neutral file version of the model into the receiver’s system. Regardless of which form is made available to the assembly designer, if there does not exist an efficient method in which she can perform design iterations, it will be an expensive process.

Many of the solutions presented in section II deal with providing solutions targeting towards solving the CAD interoperability issue, but they are not integrated efficiently, if at all, into the assembly designer’s design iteration cycle process. Other integrated solutions, such as design workspaces, work more efficiently but at the expense of divulging all proprietary information associated with a native CAD model. Oftentimes, when working with assembly designers outside of their company, suppliers
would rather that non-sensitive, non-proprietary information embedded within their native CAD files, such as design history, are not transmitted as a result of the process.

So, several needs were identified for both the assembly designer and parts suppliers in working collaboratively on a heterogeneous CAD assembly design: minimize issues associated with CAD interoperability, quick design iteration cycle times, correct versioning models, and protection of proprietary data. A unique solution arrived at the MIT CADlab seeks to satisfy all those needs.

In minimizing issues associated with CAD interoperability, neutral files exported by CAD systems such as IGES and STEP are used as the means for transferring models from one CAD system to another. This choice also satisfies the need to protect proprietary data. To satisfy the needs of quick design iteration cycle times and correct versioning models, DOME’s inherent features will be used. DOME, which stands for distributed object based modeling environment, is a web based design application tool that allows engineers to engage in concurrent engineering [Abrahamson, et Al. 2000] [Senin, et Al. 1997]. Tools developed by this author that enable this process are the main focus of this thesis.

II. Literature Review and Commercial Solutions

Both academia and industry analysts have conducted extensive research on the problem of CAD interoperability. Sources of the problems were identified and the pros and cons associated with each available solution were investigated. From the research gathered for this thesis, it is apparent that most of the solutions proposed for solving CAD
interoperability problems came from members of the industry and commercial companies, not the academia. This is understandable since CAD interoperability problems are tied to CAD user practices and the exchange of files between different CAD systems.

The solutions would require proper training and software solutions. Neither of which are particularly suitable for academia studies nor do they generate a tremendous amount of interest in the academia world. Software solutions in the form of new standards for exchanging CAD information are beyond what the academia can dictate. Plus, the academia is not in the position to enforce industry standards. Only members of the industry, who propose and adopt the new standards, are. Of the proposed standards for exchanging three-dimensional (3D) CAD information across different systems, the neutral file standards of IGES and STEP are the most popular.

Other types of software solutions are also not ideal for academic studies in that numerous commercial companies already offer their own unique solutions. Some major CAD software companies have started offering their proprietary kernel to outside participants in an attempt to address the CAD interoperability problem by advocating the standardization of CAD kernels into their own proprietary kernel. PTC is offering its Granite One platform, the proprietary kernel of Pro/Engineer, to its customers and various software partners.

More commonplace solutions are ones that address the generation of bad models resulting from either inadequate translation services or bad modeling practices. ITI Transcendata offers various services catered towards solving bad data and interoperability problems [Transcendata 2002]. CADIQ, a software tool produced by ITI
TranscenData, analyzes native CAD models to identify topological and geometric defects before models are released, thus saving the time and expense associated with model rework [CADIQ 2002]. BadCAD is both a provider and reseller of various data quality and CAD interoperability services [BadCAD 2002]. It also offers consulting services that make use of their software suite. One of its software packages allows user to manipulate and heal bad data it discovers to improve chances of successful translations. Another provides translations services from one proprietary system to another.

There is plenty of CAD file transferring services available today as well. CADCAM-E provides various native and neutral CAD file translation services [CADCAM-E 2002]. Spatial provides several services, among which is 3D InterOp, which facilitates in the transfer of 3D CAD data between different native formats and platforms [Spatial 2002]. Spatial also provides other services in the modeling and visualization sector.

In 1999, Spatial offered 3Dmodelserver.com, a web-based software application for repairing and improving 3D CAD models, for a trial period. This service does not appear to be currently operating and is not listed on Spatial’s web site as one of its available services. So, one can assume that it may not be offered in the future even though the idea behind it was novel. 3Dmodelserver.com allowed subscribers to import models of IGES, STEP, and ACIS SAT formats onto its server. The application installed on the server will then identify areas for improvement on those models, attempt to fix them, and save the healed model in IGES, STEP, and ACIS SAT formats for incorporation into the subscribers’ native CAD packages. All required software resided
on the web, so users always had access to the latest version of software and never needed to worry about installing the required software on their system.

Cadverter.com is an Internet based CAD/CAM data translation service [Cadverter 2002]. It supports the translation of numerous products, supports conversion of CAD geometry, assembly structure, and attribute data. It levies a fixed price pay as you use fee system. Translation Technologies Inc. (TTI) also offers native CAD translation services. Its flagship product, Acc-u-Trans, requires the user to first send his native file to the TTI secure server. Then the native file will be translated and converted into the designated native file format, complete with fully modifiable feature based parameters and functionality history tree [Translation Technologies Inc. 2002]. CADKEY Corporation’s flagship product, CADKEY, allows users to import neutral and native two-dimensional (2D) CAD drawings and 3D CAD models into its proprietary system for direct manipulation of translated data [CADKEY 2002].

CoCreate’s flagship product OneSpace offers a design workspace that allows users to conduct concurrent engineering [CoCreate 2002]. OneSpace helps companies collaborate by providing an infrastructure for teams to review, comment on, and change designs online in real time, regardless of the CAD system used to create the designs [MacKrell 2001]. OneSpace allows users to view data from multiple CAD systems, allows authorized users to modify those CAD models during the collaborative session, and allows users to construct assemblies from multiple CAD systems [MacKrell 2001]. With regards to working with 3D models, OneSpace supports the import of native geometry with parametric support. Modifications made to imported models within OneSpace can propagate back towards the original native model.
When it comes to areas of concurrent engineering, collaborative design and distributed design, one can find many works and solutions proposed by members of the academia [De Martino and Giannini 1997], [Gao and Bennett 1997], [Ray, et Al. 1999]. Caldwell and associates came up with a decision-support tool that operates in design-guidance, knowledge-viewing, and knowledge-capture modes called WebCADET, which stands for web-based Computer-Aided Design Evaluation Tool [Caldwell, et Al. 2000]. LaViola and associates worked in the area of real-time distributed multi-media environment for collaborative engineering. They presented a new approach to the design of a shared virtual environment that addressed issues of scalability, networking, and synchronization, and discussed its applicability to collaborative design [LaViola, et Al. 1997].

Lombeya and Regli developed a modeling system named CUP, which stands for Conceptual Understanding and Prototyping, that enables users to create a knowledge-level description of design in a collaborative, multi-user environment without having to perform detailed CAD and solid modeling [Lombeya and Regli 1999]. Thus far, it allowed a team of design engineers, collaborating over the Internet, to develop a high-level structure-behavior function (S-B-F) description of an assembly in a VRML (Virtual Reality Modeling Language) based virtual environment [Lombeya and Regli 1999]. Ikonomov proposed a virtual manufacturing and assembly system tool for concurrent engineering that made use of STEP files and the information embedded within them to evaluate designs for manufacturability and assemblability [Ikonomov 2000].

Central to concurrent engineering is the sharing of 3D model and data. Rossignac investigated the shortcomings of the current technology regarding the utilization of 3D
data throughout all phases of a product life cycle, identified the fundamental research issues, and reviewed recent advances in 3D data compression, in the automatic generation of levels-of-detail for interactive rendering, and in the innovative exploitation of 3D input devices for an intuitive and effective navigation [Rossignac 1997].

A trend that could be seen in the concurrent engineering and product design fields is the migration of design tools towards the Internet and virtual environments. Fernando and associates worked on developing a constraint-based virtual environment for supporting assembly and maintainability tasks [Fernando, et Al. 2000]. Ranky and associated conducted research into developing a web-enabled virtual disassembly manager (webVDM) for electronic products [Ranky, et Al. 2000] [Ranky, et Al. 2001]. University of California at Berkeley has created WebCAD, a web-based, destructive solid geometry tool that allows users to create components for integration with CyberCut [WebCAD 1999]. WebCAD will automatically generate a process plan and verify whether the designed component can be manufactured on a 3-axis milling machine.

Danesi and associates researched into creating a web-based distributed CAD system where the client has access to data stored on the server [Danesi, et Al. 2001]. Different options for underlying architecture were examined and a prototype tested. Beazley and Chapman investigated how VRML is used in engineering applications, and what potential possibilities it holds for future collaborative engineering design projects [Beazley and Chapman 1996]. Wang and company used distributed computation (in the forms of Java and CORBA) and computer graphics (in the form of VRML) to implement a collaborative and interactive client-server design framework for use in the design of rooms over the Internet [Wang, et Al. 2000].
Luchi and associates utilized VRML, Java, and the Internet to create Web/Assembly [Luchi, et Al. 2001]. Web/Assembly enables virtual assembly on the web without requiring the presence of a CAD system. However, CAD systems are required for exporting components in the VRML format for incorporation into the Web/Assembly. Another feature of the Web/Assembly is the availability of a parts VRML catalog. Nousch and Jung used VRML, Java, and JavaScript to create BEAVER, a web based application that allows users to interactively design furniture on the web and generate shopping lists for components and customized assembly instructions [Nousch and Jung 1999]. Jung has also conducted research into bringing the Virtual Constructor (VC) to the Internet. VC is a knowledge-based distributed system that enables an interactive assembly of 3D visualized mechanical parts to complex aggregates [Jung 2001]. By step-keepingly matching the geometry scene against a structured model of the target aggregate, an assembly can be dynamically conceptualized [Jung 2001].

Ramos and associates developed an Internet accessible VRML and Java based airship simulator based on airship dynamic models, where the simulator is meant to be used as a tool for the development of control and navigation methods for autonomous and semi-autonomous robotic airships and as test bed for airship pilot training [Ramos, et Al. 1999]. Park and company conducted research into the development of a three-dimensional web monitoring system using a VRML browser for visualization [Park, et Al.]. Direct application areas include the monitoring of a product as it goes through an assembly line. In the area of manufacturing, a VRML web based system for accessing manufacturing data called VIM (Visual Interface to Manufacturing) was developed [VIM 2001].
Immersive Design’s flagship product, Interactive Product Animator (IPA), provides animation and web publishing solutions that allow communication of 3D product information for interactive assembly, maintenance and repair documentation, product catalogs, and product presentations [Immersive Design 2002]. IPA supports native CAD formats and outputs in several formats, some of which are proprietary to Immersive Design, others are standard, such as Windows based AVIs and VRML files. IPA also supports hierarchical assembly tree displays, basic geometry creation, insertion of assemblies and parts, and automatic update of supported CAD components.

Kiss investigated into the development of a web based VRML modeling system for creating mesh models [Kiss]. His article provides an excellent introduction on how to use Java, VRML, and EAI in creating an interactive VRML session. Lots of companies also provide VRML based services. For example, ParallelGraphics specializes in providing web based visualization services, most of which makes use of VRML and its VRML browser, Corona [ParallelGraphics 2002]. SiteSculptor is Sculptware LLC’s, a spinoff of CADKEY Inc., VRML based 3D modeling tool that constructs sold models from NURBs (Non-Uniform Rational B-Splines) [SiteSculptor 2001].

Many tools for marking up CAD geometries have also been developed. They are useful for conveying information between different viewers of a common design. Solid Concepts’ SolidView provides 2D and 3D markup capabilities to CAD drawings and models [Solid Concepts 1998]. Cimmetry Systems offers a suite of products, known as AutoVue, that allow users to view and markup native 2D CAD drawings and 3D CAD models [Cimmetry Systems 2002]. Advanced packages include the ability to measure
dimensions and add annotations. A separate product line also allows users to extend the capabilities of AutoVue over the web through a Java based version of AutoVue.

CAD Centric’s flagship product, OneView, allows users to view, measure, and markup 2D CAD drawings and 3D CAD models without requiring the user to have any CAD system or server installed on her computer [CAD Centric 2002]. Among its key features are its ability to load native geometry, view several drawings and models simultaneously, measure geometric attributes, markup drawings and models, allow the user to selectively choose a component from an assembly for analysis, and insert 3D models into documents. OneView Lite, a free toned down version of OneView, is available for download from CAD Centric’s web site. For web viewings, CAD Centric offers the OneView Xpress publisher for publishing XML formatted data of CAD drawings and geometries in its OneView Xpress viewer.

III. Thesis Objective

In order to communicate with CAD systems through DOME and allow assembly designers to achieve parametric control over a heterogeneous CAD assembly, DOME CAD plug-ins were developed. Details on the capabilities, requirements, and information flow between the plug-ins, DOME, and the CAD systems are presented in Chapter 2 of this thesis. Related, earlier work on designing heterogeneous CAD assemblies through DOME was conducted by Bill Liteplo [Liteplo 2000].

To provide the CAD assembly designer with a more intuitive interface for conducting an interactive design session through DOME, a VRML and Java based
module was developed. This module is tightly integrated with the development of the CAD plug-ins and it communicates with the CAD plug-in in order to affect the underlying CAD geometry. In addition, the DOME VRML module allows users who do not possess any CAD licenses on their computers to view and modify any DOME CAD design session as long as they are connected to DOME and are subscribed to the model publisher’s service. More details regarding the capabilities, requirements, and information flow between the DOME VRML plug-in, DOME, and the CAD systems are presented in Chapter 3 of this thesis.

A case study section that details step-by-step instructions on using the DOME VRML module and CAD plug-ins for designing a heterogeneous CAD assembly of half a rear suspension system from a Ford pick-up truck through DOME will be presented in Chapter 4. Lastly, conclusions and recommendations for future work will be presented in Chapter 5.

It should be noted here that the author is fully aware of all limitations involved with the proposed process for heterogeneous CAD assembly design. Specific problems that the author recognizes that cannot be compensated for by the proposed solution include translated model inaccuracies associated with neutral files; significant delay time between iteration cycles associated with a highly complex assembly; and VRML scenes that cannot be effectively navigated when dealing with highly complex assemblies. Even in light of all these limitations, the author believes that the proposed method of designing a heterogeneous CAD assembly and the tools developed are sufficient enough to demonstrate the potential to be had with the proposed methodology and the use to be had when the tools are properly utilized.
When working with neutral files, one should expect a certain amount of error in the translation process. And it should not come as a surprise when some models fail to import properly. Some CAD systems have been found to export better neutral files and have better rates of successfully importing neutral files than others. Sangole and company observed that most exchange errors were due to incomplete topological definition resulting from the incompatibility in the degrees of precision of commercially available CAD systems [Sangole, et Al. 2002].

Mahadevan and associates tested data exchange among three major dissimilar CAD systems using standard exchange formats, such as IGES and STEP, and presented the level of interoperability found [Mahadevan, et Al. 1997]. Their findings indicate that neutral file standards do not completely eliminate the problem of CAD interoperability, as some neutral files could not be properly reconstructed in other CAD systems. There are, of course, commercial services that may be integrated into the proposed solution presented in this thesis to account for this particular issue in the future.

When working on large assemblies, it should be expected that the length of the design iteration cycle time would be proportional to the complexity of the model. Neutral files are generally smaller in file size compared to native files, but it still takes the CAD system some time to export or import them or for the file to be transferred over from the supplier to the assembly designer.

When working with highly complex VRML scenes resulting from a highly complex assembly, it is to be expected that there will be significant time lags in navigating the assembly scene through functions of the VRML browser. While the CAD plug-ins selectively filter unnecessary objects from the exported VRML scenes, not much
more can be done by the author to address this problem. It should be noted here that there are commercial products that can dramatically reduce the size of a VRML file to make the scene easily navigateable. Perhaps consideration can be given towards integrating such a service into the VRML module in the future. Also, it should be noted here that the larger the VRML scene (in terms of file size), the longer it will take the VRML browser to load/unload that scene during a DOME VRML session as either the VRML scene is updated to reflect changes made to the underlying geometry or when the user switches component scenes in order to work with the selected component. So, even though it may appear that the VRML browser is hanging during those times, it actually is not.

Finally, throughout this thesis, constant references will be made to the use of the developed tools through DOME. That may be misleading in that none of the plug-ins and modules described in this thesis was actually integrated with the newer version of DOME, DOME³, for which the plug-ins and modules developed for this thesis are intended. None of the experimental tests conducted for this thesis were done through DOME³ either. The main reason being that the new version of DOME was not available in time for the plug-ins’ integration. However, many proofs of concept were implemented and conducted in an earlier version of DOME, DOME v0.74, so the author knows that the proposed processes and concepts are implementable and will behave as expected [Liteplo 2000].

All the plug-ins and modules developed for this thesis are ready to be interfaced with the new DOME. Yet, prior to integrating with DOME³, these plug-ins and modules are standalone and can be controlled by user-supplied inputs. Inputs that the user actually
need to supply to DOME if and when the plug-ins are actually integrated with DOME³. All information flow diagrams provided in Chapters 2 and 3 are intended to hold through the DOME integration process. The experimental section, Chapter 4, will demonstrate actual functionality and behavior of the plug-ins during a case study. It is meant to demonstrate all that would happen during an interactive session whereby the plug-ins are fully integrated with DOME³.
Chapter 2: DOME CAD Plug-Ins

I. Introduction

Several DOME CAD plug-ins were developed at the MIT CADlab. They allowed CAD designers to offer and exchange geometry model services over the web. In this thesis, the plug-ins developed for CAD packages UGS Unigraphics (Version 16) and SDRC I-deas (Version 8) will be presented.

The process behind using the DOME CAD plug-ins to enable assembly designers to parametrically design heterogeneous CAD assemblies through DOME is relatively simple. However, that is not to say that the actual implementation process that goes into developing the CAD plug-ins or DOME is trivial. A simple bolted joint example will be presented here to give the reader an overview of the proposed process as seen through DOME v0.74. It should be noted here that the plug-ins developed for this thesis are meant for DOME^3, and as such, contain more functionality.

In this simple bolted joint example, three CAD systems are used. I-deas (Version 7) is used to design the bracket, SolidWorks is used to design the bolt, and Unigraphics (Version 16) is used to design the plate and assemble the heterogeneous CAD assembly. The I-deas (Version 7) and SolidWorks plug-ins were developed by Bill Liteplo of the MIT CADlab, while the Unigraphics plug-in was developed by the author for DOME v0.74. The individual components can be seen in Figure 2.1 and the assembled system in Figure 2.2.
Figure 2.1 Designers Working In Their Respective CAD Systems

Figure 2.2 Assembled Bolted Joint Assembly

Chapter 2: DOME CAD Plug-Ins
The assembly designer needs to first obtain neutral file versions of the bracket and bolt from the suppliers prior to creating the bolted joint assembly. Once the assembly is created, the three designers can log in to their DOME servers, select their respective CAD plug-in service from the DOME services menu, provide the plug-in with required information relating to the CAD model and publishable design parameters, and wait for the DOME CAD user interface to appear. Once the CAD plug-ins have established connection with the CAD servers and the models, the assembly designer and her suppliers can establish the relevant relationships defining their CAD geometries. This can be seen in Figure 2.3.

Figure 2.3 Working With DOME And Establishing Relevant Relations
Once the DOME relations are in place, the assembly designer has total control over the design parameters made available by the bracket and bolt suppliers through DOME. For example, the assembly designer has increased both the plate length and the plate thickness. This can be seen in Figure 2.4.

Since the established DOME relations dictate that the length of the bolt and the thickness of the bracket are functions of the plate thickness, the suppliers' DOME CAD plug-ins will update the CAD geometries to reflect this change in design parameter. This can be seen in Figure 2.5.
In Figure 2.5, the CAD geometries in the suppliers' CAD systems have yet to be updated. But once those CAD geometries are updated, new neutral files of those geometries will be generated and transferred over to the assembly designer through DOME. Once the assembly designer receives these neutral files, her CAD plug-in will automatically swap out the older neutral file components, which are basically dead geometries with no editable parametric capability, for the up-to-date components. The resulting bolted joint assembly can be seen in Figure 2.6.
The process outlined for DOME v0.74 also holds for DOME$^3$, although a slightly more complex process is involved with the introduction of the new DOME VRML module. More details on the DOME VRML module is presented in the next chapter.

This chapter will first briefly explain how the new DOME CAD plug-ins communicate with CAD packages. Then, basic functionalities available to all CAD plug-ins will be presented. Individual sections on the Unigraphics and I-deas CAD plug-ins will be presented and lastly, a summary of main features of the CAD plug-ins and how information propagates between DOME and the CAD system during an active session will be presented.
II. Linking CAD Plug-Ins To DOME

DOME interacts with CAD programs through DOME plug-ins, which make use of CAD APIs (Application Programming Interface). Each CAD system has its own API, which allows users to extract information from and modify geometry models in an active session external to the CAD system. Unigraphics has a C/C++ based API while I-deas has a CORBA based API.

III. Basic Plug-In Functionalities and Requirements

In order for DOME to create a graphical user interface (GUI) representation of the design parameters the CAD designer wishes to publish through DOME, the CAD designer needs to first specify the location of a particular text file. This text file, which needs to be parsed by DOME, will contain all information required by the plug-in to establish a connection to the CAD system and model. It also lists all the design parameters the user wishes to publish through DOME. The name of this file takes the form “<component>.in”, where <component> is the name of the CAD geometry. It should be noted here that in order for the CAD plug-ins to work successfully, it is imperative that any name required in the input text file not contain any spaces. That is, if the CAD user has a carrier fin component in her system, then she should name it “carrier_fin”, not “carrier fin”.

There are three types of design parameters the designer can specify for her DOME session: modifiable dimensions, dependent dimensions, and mass property
calculations. Modifiable dimensions are independent dimensions that drive the geometry of a part component. Their values can be directly modified through the DOME interface. Dependent dimensions are dimensions that are derived from values of other dimensions, some of which may be specified as modifiable dimensions for the DOME session. These values cannot be modified through DOME. Mass property calculations are self-explanatory. They are geometry dependent, and hence, are directly influenced by any changes made to modifiable dimensions. For an assembly, all mass property values of its children instances are calculated with respect to the assembly’s world coordinate frame.

In addition to allowing designers the ability to modify geometry through DOME, designers also have control over exported and imported geometry. The designer can publish neutral files of his model as services over DOME and subscribe to neutral file services when designing a heterogeneous CAD assembly. Export and import services only handle neutral files of types IGES and STEP. However, it is not possible at this time for the designer to both publish and subscribe to neutral file services simultaneously.

That is, if an assembly designer is working with a heterogeneous CAD assembly, she cannot export a neutral file version of her assembly for incorporation into another assembly designer’s system. This was a design decision made by the author in the development of the CAD plug-ins in an attempt to prevent further degradation in the geometry fidelity of the originally imported component if it were to be exported a second time from another CAD system. Repeated import and export of the same component as it goes through several CAD systems is bound to accumulate errors. However, with some additional work, the plug-ins can be modified to support simultaneous export and import capabilities when dealing with aggregate assemblies in the future.
An additional service the CAD designer can publish and subscribe to is the VRML (Virtual Reality Modeling Language) service. It should be noted here that even though the VRML file is exported from the CAD system, it is not lumped in the case category as the exported neutral file. VRML files are only exported for visualization purposes, not for incorporation into assemblies. Detailed information regarding the VRML service is provided in the next chapter.

A schematic of the DOME CAD plug-in initialization process can be seen in Figure 2.7.

![Figure 2.7 CAD Plug-In Initialization Information Flow Diagram](image)
Basically, once the CAD plug-in locates the user specified input text file, it will parse the file to extract the model information, design parameters, and file exchange requirements. The CAD server will then be connected to and the specified model file loaded into the active CAD session. The user specified modifiable dimension objects would then be located within the model file and the appropriate link established to allow the user to directly manipulate its value through the DOME graphic user interface (GUI). Any user specified design parameters that are invalid, such as incorrectly named in the user input file, will not be represented in the DOME GUI. If a VRML file was specified for the CAD model, then it will be exported from the CAD system.

After the CAD plug-in initializes properly and the DOME GUI is successfully created, a “<component>.txt” file will be generated. This file will contain assembly hierarchy information if the component selected for a DOME design session is an assembly. If the CAD plug-in user is a supplier who supplies his part to a heterogeneous CAD assembly designer, then he also needs to transfer this file to the assembly designer so she can properly create the “<component>.in” input text file if she wishes to utilize the DOME VRML module and traverse her supplier’s assembly.

III.A. Exporting Components

A CAD designer can publish exported neutral file services over DOME. When an assembly designer subscribes to the CAD designer’s service, the two designers can then begin designing a heterogeneous CAD assembly. In preparing his component for
integration into a heterogeneous CAD assembly, the CAD designer needs to first export his geometry in a format specified by the assembly designer and generate a text file containing density information related to his geometry. The geometry export process will need to be done manually. However, the density text file can be automatically generated for the designer through a program that runs external to DOME and is supplied along with DOME if the designer decides to forgo manually creating this text file.

To run this program, the CAD designer must have an active CAD session up and running and opened to the desired model file. The generated density file will then be placed in a specific directory, which resides as a subdirectory with the name “export” under the model directory, with the name “density.txt”. The model directory is the directory under which the model file that DOME will interact with resides. If the “export” directory does not exist, it will be created by the executable application. Any and all subsequent exported files to be generated via a DOME session will be placed in this “export” directory as well.

Then, the CAD designer needs to transfer the exported neutral and density files over to the assembly designer. This is usually done outside of DOME. Once the assembly designer have received the exported files, have run a DOME supplied executable program that automatically extracts the neutral file into her CAD system and applies the appropriate density values to the solids based on data contained in the density text file, and have integrated the extracted components into her native assembly, she and the CAD designer can then start their DOME sessions and link their services to begin real-time design iterations.
During the design iteration process, it may become necessary for the CAD designer to supply the assembly designer with a modified neutral file. Rather than requiring the supplier to manually perform this operation, the CAD plug-in will recognize when this need arises and automatically compensates for it. That is, it will generate the updated neutral file when it knows it’s required.

It should be noted here that the only allowable component that can be exported by the CAD designer during a DOME session is the top most component in his model hierarchy. For a single part component session, it is simply that part component. For an assembly component session, it is the entire assembly. However, the only exception comes in the export of assembly component VRML files for use in the DOME VRML module. This is to allow the assembly designer the ability to view all his DOME components through the VRML module.

Initially, it was intended that when performing mass property calculations for a heterogeneous CAD assembly, only values calculated from original geometry would be used. That is, geometry from imported neutral files would not be used in order to avoid inaccuracies attributed to translator errors. However, due to limitations on the amount of information that is retained through neutral files, this preferred method could not be realized. In order for this method to work, several criteria need to be met.

First, mass property calculations about the center of mass must be made available through DOME to provide up to the minute values, which may change during a design iteration process. This is not a problem. Secondly, information regarding the location of the center of mass for that exported component about its local coordinate frame needs to be extracted and made available to the assembly designer. This is also not a problem.
Third, and most importantly, the local coordinate frame of the exported component, once imported into the assembly designer’s CAD system, must remain identical to its native system. This is where problems arise. While local coordinate frames are retained during an IGES export, they are not retained for a STEP export. Because the third criterion could not be met for all cases, a different method of approach need to be taken when dealing with calculating mass property values for a heterogeneous CAD assembly.

The chosen alternative method was already mentioned in the beginning of this section: make use of a density file for identifying which density value to apply to which component when it is imported into a different CAD system. The only catch to this method is that it requires a single part to be composed of a uniform material. However, with this solution, the geometry of the imported component would be utilized and that means there will be an error introduced into the mass property calculations dependent on the translation accuracy. The significance of those errors has yet to be determined in real test cases and is expected to decrease significantly with updates to the IGES and STEP standards.

Since density values also do not carry over in a neutral file, they need to be made available to the assembly designer somehow. It was decided that a text file, to be named “density.txt”, would be the best method of conveying this information. This text file will contain information such as the name of the part component and its associated density value in the units of lb-in. An example density file for a camera assembly, which can be seen later in Figure 2.10, composed of three components (body, flash, and lens) can be seen in Figure 2.8.
An API program that functions independently of DOME has been developed to automatically generate this density file. This program requires that the user have his CAD system up and running and opened to the desired model file for export. The generated density file only needs to be accessed once during the initial startup of the heterogeneous CAD session. It is not expected that the material of the part component will ever change during a DOME session. Hence, the current process does not support the changing of material density for any imported components during a DOME session.

This API program does not export the component for the CAD designer, so the CAD designer is required to manually export her component the first time prior to transferring it to the assembly designer for incorporation into a heterogeneous CAD assembly. All subsequent export operations, once the appropriate DOME models are created, will be performed automatically through the DOME CAD plug-ins.

It should be noted here that depending on the CAD system, there might be instances that when dealing with an imported assembly, the names of the extracted part components differ from those in its native system. Different CAD systems have different ways of naming base components for an imported assembly, such as adding prefix and suffix to the original name of the part. Those patterns have been recognized and are accounted for by the CAD plug-ins. So, users do not have to worry about this issue.

When dealing with multiple imported files, all information pertaining to part densities need to be accumulated into a single density file. In this case, the assembly
designer will need to perform this operation manually. This is a very simple process and it only needs to be performed once. However, if during this process the assembly designer notices that several part components share a common name, then she must take steps to ensure that no two parts share the same original name (that is, its name within its native CAD model) by working with her suppliers.

Based on various issues encountered while developing the CAD plug-ins, it is heavily recommended that when working with an assembly that no two components ever share the same name regardless of its type, location, etc. Having two components share the same name can cause various problems to occur in the assembly and may lead to unpredictable results when running DOME. However, specific exceptions exist and they are mentioned in the individual plug-in sections.

It should be noted here that if it is the intention of the CAD designer to offer his neutral file service to an assembly designer working on a heterogeneous CAD assembly design, then he must inquire into whether the assembly designer will be working with the DOME VRML module. If the assembly designer will be working with the DOME VRML module, then the CAD designer must make sure that he also publish a VRML service and make it available to the assembly designer, along with additional files that the VRML module would require. He also needs to inform the assembly designer of the names of his subassembly components, if any, that contain design parameters publishable through DOME so those components can be traversed through the assembly tree within the VRML module. These extra steps are necessary due to how the VRML module operates and details about these requirements will be presented in the next chapter.
III.B. Importing Components

For the assembly designer working with imported neutral files, an initial import preparation process is required when she brings the imported components into an assembly for the very first time. An API program that functions independently of DOME was developed to automatically perform this initial import operation and prep the extracted components. Subsequent import operations during an active DOME session will be performed automatically by the CAD plug-ins.

The import program runs external to DOME and will extract all imported files into the appropriate directory or bin depending on which CAD system the files are intended for. The exported files, that need to be extracted for incorporation into the assembly, must be placed in the “import” subdirectory under the model directory. If this directory does not already exist, the assembly designer needs to create it. Into this directory must also be placed the “density.txt” and “import.txt” files. The purpose of the “density.txt” file was explained in the previous section. The purpose of the “import.txt” file will be explained shortly.

If the assembly designer is dealing with neutral files from several suppliers, all density files must be consolidated into a single density file containing the sum of all information. If repeated component names are discovered, steps must be taken to ensure that no two components ever share the same original name.

The purpose of the “import.txt” file is to inform the import program of the names of the files residing in the import directory that needs to be extracted for incorporation into the assembly. This provides the assembly designer with the flexibility of selecting
only a few files within the import directory for incorporation into the assembly. The "import.txt" file contains lines that follow the following format:

<prefix>_<name of file to be imported, minus the directory path>

The prefix indicates what type of exporter is required to extract the given file. Allowable prefixes are "s_" for STEP and "i_" for IGES, both in lower case only. For example, "s_bolt_01.stp" would indicate a STEP file named "bolt_01.stp" while "i_bolt_02.igs" would indicate an IGES file named "bolt_02.igs". Only files that can be identified through "import.txt" will be extracted and prepared for integration into the assembly. A sample "import.txt" file can be seen in Figure 2.9.

```
i_pist.igs
s_camera.stp
i_cam.igs
```

Figure 2.9 Sample "import.txt" file

Once all the neutral files are extracted and prepped for the assembly, the assembly designer needs to integrate those components into her assembly by applying all relevant mating conditions. This process is only required the first time the assembly designer creates her assembly. Any and all subsequent changes made to her assembly which requires that newly generated neutral files replace older components within the heterogeneous CAD assemblies, are handled automatically by the CAD plug-ins. Basically, the CAD plug-ins will see to it that all previously applied mating conditions are maintained in light of newly swapped in components. This method, which will be
gone over in greater detail in the individual plug-in sections, is more robust and renders
the need for an assembling script unnecessary.

In addition to making sure that all mating conditions carry over from one design
iteration to the next, the CAD plug-in also ensures that color does as well. This means
that if the assembly designer chooses to make her imported bolt red, then all future
imported bolts will be red. The only limitation when it comes to applying colors to
imported components is that all surfaces associated with that component will share the
same color irregardless of whether that component is a part or an assembly, and that there
is no way to alter the surface color during a DOME session. If multiple colors are
applied to a component, then the first color extracted by the plug-in will be used on all its
future replacements within the assembly.

IV. Unigraphics Plug-In

The DOME Unigraphics plug-in works with Unigraphics version 16. The
Unigraphics API is C/C++ based and is compiled using Microsoft Visual C++ 6.0. The
compiled plug-in makes external function calls to Unigraphics. A screenshot of
Unigraphics can be seen in Figure 2.10.
Since an external Unigraphics API program cannot connect with an active Unigraphics session and directly manipulate the model so that the graphic display area updates to reflect changes made to the model through DOME, a VRML interface is used to provide visualization of the CAD geometry during a DOME session. To provide convenient views for navigating the VRML scene, the Unigraphics CAD plug-in records the initial bounding box coordinates associated with the component and use that to determine the location of the VRML viewpoint cameras. For the Unigraphics plug-in, it is not necessary for the user to specify that a VRML module be made available through DOME given the above reason.

Once a DOME Unigraphics session successfully connects to the Unigraphics server and model, it will establish the proper links and extract relevant values for creating
the DOME Unigraphics GUI. A sample DOME v0.74 Unigraphics GUI can be seen in Figure 2.11.

In addition to successfully creating the GUI, the Unigraphics plug-in will also generate a VRML file. This file can then be accessed through the DOME VRML module. The DOME VRML module can be seen in Figure 2.12 and more details about it are presented in the next chapter.
Depending on the user setting for the VRML module, a new VRML file may be exported and the VRML module's scene refreshed to reflect changes made to the geometry whenever any changes are made to the Unigraphics model through DOME. Depending on the complexity of the actual geometry, it will directly impact how fast a VRML file will be made available for incorporation into the VRML module or how fast a user can navigate the VRML scene through the VRML browser.
IV.A. Unigraphics Plug-In Interface Specification

In order for the Unigraphics plug-in to know which model file is to be used for a DOME session and which design parameters the CAD designer wishes to publish through DOME, a specifically formatted text file containing certain required elements will be required. The full file path to this file is the only required input to the DOME Unigraphics plug-in before an interactive Unigraphics session is launched and ready for interaction with DOME. This input text file must follow the following format and in the specified order:

<directory name>
<offset index> <VRML component index> <VRML type>

{Top Most Component}
0 <model name>
1 <mass property analysis unit> optional
2 <modifiable dimension name>
3 <output dimension name>
4 <mass property calculation name>
5 <export file type>

{Subsequent Components}
0 <model name>
1 <mass property analysis unit> optional
2 <modifiable dimension name>
3 <output dimension name>
4 <mass property name>

{Import Components}
6 <import model>
7 <import assembly’s children component actual name> <name in Unigraphics, which differs from original intended name due to nature of the importers> <component index within its native system> <parent index within its native system> <component type: 1 for part, -1 for assembly>
The first line must contain the full file path to the directory in which the Unigraphics model resides. The next line must contain the offset index, the VRML component index, and the VRML type. These values are only relevant to the DOME VRML module. The offset index value is pivotal in enabling the user to successfully traverse the assembly tree represented within the DOME VRML interface. Normally, this value would be set to 0 if the user does not export neutral file versions of his component for incorporation into an assembly designer's CAD system. However, if he is making his component available to an assembly designer and the assembly designer wishes to design a heterogeneous CAD assembly through the DOME VRML interface, then a non zero offset index value would be required to make the transition process seamless as the assembly designer traverses her imported components for design purposes.

The VRML component index merely indicates to the CAD plug-in which component of an assembly a VRML scene and data file need to be exported for. The VRML type value is merely used to communicate to the CAD plug-in whether both the visual scene and user interface need to be updated in the DOME VRML module or just the user interface. If both the visual scene and user interface need to be updated, then the CAD plug-in will export both VRML scene and data file for incorporation into the DOME VRML module. If only the user interface needs to be updated, then only the data file will be generated.

The next line must be a newline (a line that only contains the ENTER character) that denotes the start of a new component module, where within each component module, the name of the component (its actual model file name, minus the directory file path) and
its publishable DOME objects are specified. The first component to be listed must be the component that DOME will directly interact with at the top most level. For Unigraphics, the name of that model file corresponds to the name of the root assembly or part component. Only for the top component can the user specify export and import options and they are simply denoted by “step” and “iges”.

There are four selections of mass property analysis unit users can select from. The choices are: “lb in”, “lb ft”, “g cm”, and “kg m”. In the absence of this unit, which can be specified for each component the user wishes to make available for interaction through DOME, the default value of “lb in” is used. The purpose of this unit, which is case sensitive, is to serve as an indicator to the Unigraphics plug-in which unit to calculate the mass property values in.

The user can specify a variety of mass property calculations that she wishes to publish through DOME as design parameters. The choices are:

“density”
“volume”
“area”
“mass”
first moment \{“Mxc”, “Myc”, “Mzc”\}
center of gravity \{“Xcbar”, “Ycbar”, “Zcbar”\}
moments of inertia work \{“Ixxw”, “Iyyw”, “Izzw”\}
moments of inertia centroidal \{“Ix”, “Iy”, “Iz”\}
product of inertia work \{“Pyzw”, “Pxzw”, “Pxyw”\}
product of inertia centroidal \{“Py”, “Px”, “Pxy”\}
radii of gyration work \{“Rgxw”, “Rgyw”, “Rgzw”\}
radii of gyration centroidal \{“Rgx”, “Rgy”, “Rgz”\}
principal moments of inertia \{“Ixxp”, “Iyyp”, “Izzp”\}

Where only the names in quotation marks are allowable. If mass property calculations are specified for any children instances of the top assembly, then those values are
calculated with respect to the top assembly’s world coordinate frame. All letters are case sensitive. Sample input text files can be seen in Table 2.1.

Table 2.1 Sample DOME Unigraphics Plug-In Interface Specification Files

<table>
<thead>
<tr>
<th>Part</th>
<th>Assembly that performs export operations</th>
<th>Assembly that performs import operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>C:\\UgModel\\Ug_Parts 0 0 1</code></td>
<td><code>C:\\ UgModel \\Ug_Parts 3 0-1</code></td>
<td><code>C:\\ UgModel \\Ug_Parts 0 0-1</code></td>
</tr>
<tr>
<td>0 camera_lens</td>
<td>0 camera</td>
<td>0 camera</td>
</tr>
<tr>
<td>2 full_extension</td>
<td>1 lb in</td>
<td>1 lb in</td>
</tr>
<tr>
<td>2 in_case_height</td>
<td>4 mass</td>
<td>4 mass</td>
</tr>
<tr>
<td>2 cap_out_height</td>
<td>4 volume</td>
<td>4 volume</td>
</tr>
<tr>
<td>3 cap_out_radius</td>
<td>5 step</td>
<td></td>
</tr>
<tr>
<td>3 out_case_height</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 lens_total_length</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 mass</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 volume</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Xp_X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 camera_lens</td>
<td>0 camera</td>
<td>0 camera</td>
</tr>
<tr>
<td>0 camera</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 lb in</td>
<td>1 lb in</td>
<td></td>
</tr>
<tr>
<td>4 mass</td>
<td>4 mass</td>
<td></td>
</tr>
<tr>
<td>4 volume</td>
<td>4 volume</td>
<td></td>
</tr>
<tr>
<td>4 Xp_X</td>
<td>4 Xp_X</td>
<td></td>
</tr>
<tr>
<td>0 camera_lens</td>
<td>0 camera</td>
<td>0 camera</td>
</tr>
<tr>
<td>2 full_extension</td>
<td>1 lb in</td>
<td>1 lb in</td>
</tr>
<tr>
<td>2 in_case_height</td>
<td>4 mass</td>
<td>4 mass</td>
</tr>
<tr>
<td>2 cap_out_height</td>
<td>4 volume</td>
<td>4 volume</td>
</tr>
<tr>
<td>3 cap_out_radius</td>
<td>5 step</td>
<td></td>
</tr>
<tr>
<td>3 out_case_height</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 lens_total_length</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 mass</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 volume</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Xp_X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 camera_lens</td>
<td>0 camera</td>
<td>0 camera</td>
</tr>
<tr>
<td>2 full_extension</td>
<td>1 lb in</td>
<td>1 lb in</td>
</tr>
<tr>
<td>2 in_case_height</td>
<td>4 mass</td>
<td>4 mass</td>
</tr>
<tr>
<td>2 cap_out_height</td>
<td>4 volume</td>
<td>4 volume</td>
</tr>
<tr>
<td>3 cap_out_radius</td>
<td>5 step</td>
<td></td>
</tr>
<tr>
<td>3 out_case_height</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 lens_total_length</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 mass</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 volume</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Xp_X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 camera_lens</td>
<td>0 camera</td>
<td>0 camera</td>
</tr>
<tr>
<td>0 camera</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 lb in</td>
<td>1 lb in</td>
<td></td>
</tr>
<tr>
<td>4 mass</td>
<td>4 mass</td>
<td></td>
</tr>
<tr>
<td>4 volume</td>
<td>4 volume</td>
<td></td>
</tr>
<tr>
<td>4 Xp_X</td>
<td>4 Xp_X</td>
<td></td>
</tr>
<tr>
<td>0 camera_body</td>
<td>0 camera_body</td>
<td>0 camera_body</td>
</tr>
<tr>
<td>1 kg m</td>
<td>1 kg m</td>
<td></td>
</tr>
<tr>
<td>2 base_height</td>
<td>2 base_height</td>
<td></td>
</tr>
<tr>
<td>2 base_length</td>
<td>2 base_length</td>
<td></td>
</tr>
<tr>
<td>2 battery_width</td>
<td>2 battery_width</td>
<td></td>
</tr>
<tr>
<td>3 lens_diameter</td>
<td>3 lens_diameter</td>
<td></td>
</tr>
<tr>
<td>4 Xcbar</td>
<td>4 Xcbar</td>
<td></td>
</tr>
<tr>
<td>4 Ycbar</td>
<td>4 Ycbar</td>
<td></td>
</tr>
<tr>
<td>4 Zcbar</td>
<td>4 Zcbar</td>
<td></td>
</tr>
<tr>
<td>6 s_camera_lens</td>
<td>6 s_camera_lens</td>
<td></td>
</tr>
<tr>
<td>6 i_flash</td>
<td>6 i_flash</td>
<td></td>
</tr>
</tbody>
</table>
In Table 2.1, there are three columns. The first column corresponds to a single part component that is neither exporting nor importing neutral files. The second column corresponds to an assembly component that is expected to export neutral files. And the third column corresponds to an assembly component that is a heterogeneous CAD assembly composed of imported neutral file components.

If the Unigraphics user is a heterogeneous CAD assembly designer and some of her imported components are assemblies, then she can traverse those assemblies’ children components by specifying certain parameters in her CAD plug-in input text file. It should be noted here that only components made available to DOME, through the input text files of the CAD plug-ins, could be traversed through the DOME VRML assembly tree.

First, the assembly designer needs to specify the name of the child component in her supplier’s CAD system. This information can be obtained from her supplier. Then she needs to specify the name of that component within her CAD system. Due to the nature of the Unigraphics importers, the names of the imported components may differ from that of its original counterpart in the supplier’s CAD system. Together, those two names allow the Unigraphics CAD plug-in to correctly identify the child component of the imported assembly within the heterogeneous CAD assembly. Next, three integer values need to be provided: the component index of the child component within its native CAD system; the child component’s parent index within its native CAD system; and, the child component type. These values can be extracted from the “<component>.txt” text file, supplied by the supplier, that will be automatically generated for the DOME supplier once his DOME CAD session initializes successfully and he has specified in his CAD
plug-in input text file to that he will be exporting VRML files from his CAD system. It is the responsibility of the assembly designer to coordinate efforts with her suppliers if she wants her suppliers to provide her with VRML files for working in the DOME VRML module.

If the Unigraphics user will be supplying a heterogeneous CAD assembly designer with his geometry, then he needs to run the API program “ug_density.dll” the first time he exports his geometry for incorporation into the assembly designer’s system. In order for the API program to work correctly, the user must have Unigraphics already up and running and opened to the file containing his geometry. This program will simply extract the density value of all children components, given the geometry is an assembly, and list them in a “density.txt” file.

IV.B. Import Issues Specific to Unigraphics

When the Unigraphics plug-in recognizes that an import operation will be performed as part of a DOME session, it will create an “original” subdirectory under the model directory specified in the input text file. All neutral files from the “import” directory will be copied over to this directory. This is done to safeguard the overwriting and deletion of originally extracted neutral files as new versions are transferred over and extracted to replace older versions within the heterogeneous CAD assembly. When a new assembly is extracted from a neutral file, one file will exist for each part and assembly components that this assembly is comprised of. When a newer version of this component is made available and ready to be extracted, the previously extracted
component and all its children components will be deleted from the system for better file management during the DOME session.

It is the intention that after the assembly designer terminates her DOME Unigraphics session, all neutral files transferred to her system will be deleted and replaced with the original files stored under the “original” subdirectory. This will ensure that the next time the assembly designers opens her CAD model or links that model to a new DOME session, her assembly will appear unchanged from its previously saved state. That is, the state prior to hooking up with DOME.

Unigraphics does not allow two components with the same model name to exist within the same session regardless of different directory file paths. This presents serious problems when working with a heterogeneous CAD assembly that incorporates multiple imported files. The only way to bypass this problem is for the assembly designer to check with all her suppliers to verify that none of their components share the same name. If any of her suppliers’ components share the same name, then appropriate steps will need to be taken, if allowed.

To extract all neutral files for integration into her assembly, the assembly designer must run the UG/Open User Function “extractImport.dll”. This program is provided with DOME and runs independent from it. To run this program, the assembly designer must have her Unigraphics session up and running and opened to a file located in the model directory. Running this program will then cause all existing files listed in “import.txt”, located in the “import” directory, to be extracted and saved into the “import” directory.

During the extraction process, all surfaces of the geometry will be assigned a number to facilitate the component replacement process for the heterogeneous CAD
assembly. In order for Unigraphics to successfully replace an existing component with a different one, all the while retaining mating conditions, it is necessary that all mating objects be identifiable through their names and that those names can be mapped to corresponding elements within the new component. That is, if “face1” of “comp1” is mated to “face 2” of “comp2”, then if “comp1” is replaced by “comp1b”, then “comp1b” must also possess a “face1” in order for “comp1b” to successfully replace “comp1” in the assembly, all the while maintaining the mating conditions already established through “comp1”.

To ensure this, the Unigraphics plug-in will name all solid faces starting from the base part children level, and then work its way up the assembly tree. This method was found to produce repeatable results when compared to the alternative method of naming solid faces as they are cycled through from the root assembly.

Currently, the only supported mating objects for a Unigraphics assembly is solid faces. The assembly designer should be aware of this stipulation when creating her assembly. Other mating objects, such as centerlines and edges, will be supported in the future versions of this plug-in. But for now, they are not supported and should not be used lest Unigraphics fails to replace older imported components with newer ones.

Aside from importing neutral file components and saving them as new Unigraphics models, the “extractImport.dll” program also applies the appropriate density value to part components by referring to the “density.txt” file, which also resides in the “import” directory. If this density file does not exist, then the density of all extracted components will be set to 1.0 in the user specified analysis unit system. It should be noted here that care is be taken by the Unigraphics plug-in to ensure that the appropriate
density values are applied to all newly extracted components during subsequent import processes.

V. I-deas Plug-In

The DOME I-deas plug-in works with I-deas version 8. The I-deas API is CORBA based and is compiled using Microsoft Visual C++ 6.0. A screenshot of I-deas can be seen in Figure 2.13.

Figure 2.13 I-deas CAD System
V.A. I-deas Plug-In Interface Specification

In order for the I-deas plug-in to know which model file is to be used for a DOME session and which design parameters the CAD designer wishes to publish through DOME, a specifically formatted text file containing certain required elements will be required. The full file path to this file is the only required input to the DOME I-deas plug-in before an interactive I-deas server can be connected to and an I-deas session ready for interaction with DOME. This input text file must follow the following format and in the specified order:

<host name>
<I-deas project name>
<directory name>
$model name$
<offset index> <VRML component index> <VRML type>

{Top Most Component}
$component type$ <bin name> <component name>
2 <modifiable dimension name>
3 <output dimension name>
4 <mass property calculation name>
5 <export file type>
9 <VRML export>

{Subsequent Components}
$component type$ <bin name> <component name>
2 <modifiable dimension name>
3 <output dimension name>
4 <mass property name>

{Import Components}
$import type$ <import component name> $component type$
8 <import assembly’s children component name> <component index within its native system> <parent index within its native system> <component type: 1 for part, -1 for assembly> {must directly follow its parent before the next imported component is specified in the file}
In order for an I-deas API program to connect to an I-deas server and access a model, it requires four mandatory fields: the host name (server name on a network), the I-deas project name, the full file path to the model directory, and the I-deas model name. The next line must contain the offset index, the VRML component index, and the VRML type. These values are only relevant to the DOME VRML module. The offset index value is pivotal in enabling the user to successfully traverse the assembly tree represented within the DOME VRML interface. Normally, this value would be set to 0 if the user does not export neutral file versions of his component for incorporation into an assembly designer’s CAD system. However, if he is making his component available to an assembly designer and the assembly designer wishes to design a heterogeneous CAD assembly through the DOME VRML interface, then a non zero offset index value would be required to make the transition process seamless as the assembly designer traverses her imported components for design purposes.

The VRML component index merely indicates to the CAD plug-in which component of an assembly a VRML scene and data file need to be exported for. The VRML type value is merely used to communicate to the CAD plug-in whether both the visual scene and user interface need to be updated in the DOME VRML module or just the user interface. If both the visual scene and user interface need to be updated, then the CAD plug-in will export both VRML scene and data file for incorporation into the DOME VRML module. If only the user interface needs to be updated, then only the data file will be generated.

After the mandatory data are provided, a newline is required to denote the start of a new component module, where within each component module, the component type
(part or assembly), bin name (location of where the component resides within the model file), and component name need to be provided. The user is required to specify the component type because unlike Unigraphics, I-deas can support two components with the same name given they are of different types (or of the same type but exist in different bins or contain different part numbers).

The first component to be listed directly after the required elements must be the component that DOME will directly interact with at the top most level. Only for the top component can the user specify export and import options and they are simply denoted as “step”, “iges” and “vrml”. To denote component type, a number is required in the field: 0 for part and 1 for assembly. To denote the import type, a number is also required in the field: 6 for STEP and 7 for IGES. By specifying VRML as a published service, the DOME VRML module will be created.

The user can specify a variety of mass property calculations that he wishes to publish through DOME as design parameters. The choices are:

“volume”
“solidSurfaceArea”
“openSurfaceArea”
“mass”
center of gravity {“cg_x”, “cg_y”, “cg_z”}
principal values {“i11”, “i22”, “i33”}
principal axes {“1X”, “1Y”, “1Z”, “2X”, “2Y”, “2Z”, “3X”, “3Y”, “3Z”}

Where only the names in quotation marks are allowable. If mass property calculations are specified for any children instances of the top assembly, then those values are calculated with respect to the top assembly’s world coordinate frame. The unit used for mass property calculations is the unit associated with the model file. All letters are case sensitive. Sample input text files can be seen in Table 2.2.
<table>
<thead>
<tr>
<th>Part</th>
<th>Assembly that performs export operations</th>
<th>Assembly that performs import operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>cadlab38 Administrator C:\ Ideas assem1 0 0 1</td>
<td>cadlab38 Administrator C:\ Ideas assem2 5 0 -1</td>
<td>cadlab38 Administrator C:\ Ideas assem3 0 0 -1</td>
</tr>
<tr>
<td>0 Admin t2_piston 2 piston_diameter 2 piston_length 4 principalAxes 4 volume 4 solidSurfaceArea 4 openSurfaceArea 4 mass 4 centerOfGravity 4 momentOfInertia 4 principalValues 5 step 8 vrml</td>
<td>1 Main Product 4 centerOfGravity 5 iges 0 Admin t2_piston 2 piston_diameter 2 piston_length 4 principalAxes 4 volume 4 solidSurfaceArea 4 openSurfaceArea 4 mass 4 centerOfGravity 4 momentOfInertia 4 principalValues 0 Admin t2_block 2 block_length 3 dist_bet_holes 4 mass</td>
<td>1 Main Product 4 centerOfGravity 8 vrml 0 Admin t2_piston 2 piston_diameter 2 piston_length 4 principalAxes 4 volume 4 solidSurfaceArea 4 openSurfaceArea 4 mass 4 centerOfGravity 4 momentOfInertia 4 principalValues 0 Admin t2_block 2 block_length 3 dist_bet_holes 4 mass</td>
</tr>
<tr>
<td>1 Main triangle 4 volume</td>
<td>7 i_cam 0 7 i_pist 1</td>
<td>6 s_camera 1 8 camera_lens 1 0 1 8 camera_flash 2 0 1 8 camera_body 3 0 1</td>
</tr>
</tbody>
</table>

Chapter 2: DOME CAD Plug-Ins
Similar to Table 2.1 for the Unigraphics plug-in, Table 2.2 also have three columns. The first column corresponds to a single part component that is expected to export both a neutral file and a VRML file. The second column corresponds to an assembly component that is expected to export neutral files. And the third column corresponds to an assembly component that is a heterogeneous CAD assembly composed of imported neutral file components and expected to export a VRML file for interaction with the VRML module.

If the I-deas user is a heterogeneous CAD assembly designer and some of her imported components are assemblies, then she can traverse those assemblies' children components by specifying certain parameters in her CAD plug-in input text file. It should be noted here that only components made available to DOME, through the input text files of the CAD plug-ins, could be traversed through the DOME VRML assembly tree.

First, the assembly designer needs to specify the name of the child component in her supplier’s CAD system. This information can be obtained from her supplier. Next, three integer values need to be provided: the component index of the child component within its native CAD system; the child component’s parent index within its native CAD system; and, the child component type. These values can be extracted from the “<component>.txt” text file supplied by the supplier that will be automatically generated for the DOME supplier once his DOME CAD session initializes successfully and he has specified in his CAD plug-in input text file to that he will be exporting VRML files from his CAD system. It is the responsibility of the assembly designer to coordinate efforts.
with her suppliers if she wishes them to provide her with VRML files for working in the DOME VRML module.

If the I-deas user will be supplying a heterogeneous CAD assembly designer with his geometry, then he needs to run the API program “id_density.exe” the first time he exports his geometry for incorporation into the assembly designer’s system. In order for the API program to work correctly, the user must have I-deas already up and running and opened to the file containing his geometry. He must issue the following command at the I-deas Command Prompt: `oaxx start <full file path to id_density.exe> <host name>`. For example, “C:\IdeasFunctions\id_density.exe cadlab12”. This program will simply extract the density value of all children components, given the geometry is an assembly, and list them in a “density.txt” file.

V.B. Import Issues Specific to I-deas

Unlike the DOME Unigraphics session, the DOME I-deas session does not affect the original model file when importing newer versions of neutral files into an assembly. This renders the need for an “original” subdirectory unnecessary.

Similar to the Unigraphics import preparation process, an executable file needs to be run the first time the assembly designer prepares imported files for integration into her assembly. This I-deas API program, called “import.exe”, runs independent of DOME and is provided along with DOME. To run this program, the user should enter a command of the following format at the I-deas Command Prompt: `oaxx start <full file...`
The path to import.exe > <host name>. For example, “C:\IdeasFunctions\import.exe cadlab12”.

What this program will do is import all files listed in “import.txt”, which resides in the “import” subdirectory under the model directory, and place them in the “import” bin within the model file. Care must be taken to make sure that the model file does not already contain bins with names of “import” or “import2” prior to either running this program or the DOME I-deas plug-in. Also, it is best that no subsequent bins be created after the “import” or “import2” bins are created in the model file. Doing so would adversely affect where the imported components would be placed since I-deas will automatically place all imported components within the latest created bin, which would differ from where the I-deas plug-in will search for the newly imported components.

Aside from importing neutral file components into the I-deas model, “import.exe” will also apply the appropriate density value to part components by referring to the “density.txt” file, which also resides in the “import” directory. To apply those density values, in the units of “lb in”, the program will first switch the I-deas unit system to “lb in” before applying the density value to the component. Once that is done, the unit system will be switched back to its original setting. If “density.txt” does not exist, the density value of the component is set to 1.0 in the model’s unit system.

I-deas does not possess a similar functionality to Unigraphics when it comes to replacing instances within an assembly, all the while retaining previously applied mating conditions. A more complex method that involves the recording of the older instance’s orientation and mating conditions is required. After the orientation and mating conditions information are extracted from the older instance, the older instance can then
be removed from the assembly and the newer instance brought in. All mating conditions and orientations recorded from the older instance are then applied to the newer instance. Care is also taken to ensure that the appropriate density value is applied to the new component. Finally, the older component is permanently deleted from the I-deas bin.

The process of extracting mating conditions from an older instance and applying them to the newer instance is relatively simple when only dealing with a part instance. However, it becomes a bit more complex when dealing with an assembly instance. An assembly instance may contain subassembly and part instances, some of which may be duplicates.

In order for all mating conditions to be extracted from an assembly instance, it becomes necessary that all its children instances be cycled through and all their mating conditions recorded. It should be noted here that only mating conditions applied to part instances are necessary for extraction. Mating conditions only exist between part instances. By cycling through all part instances within an assembly instance and recording their mating conditions, the I-deas plug-in is able to successfully replace older instances with newer instances within a heterogeneous CAD assembly.

VI. Summary and Conclusions

Both the Unigraphics and I-deas CAD plug-ins allow users to specify design parameters, export or import neutral files, and export VRML files. A required input file specifying those parameters is required by the plug-ins. When dealing with a model that publishes neutral file services, any changes made to the modifiable dimensions would
cause the CAD system to update the CAD geometry if the CAD plug-in is currently set to react to changes made to the design parameters. If the model has updated successfully, the DOME interface will be updated with new values. If the CAD plug-in is currently set to export files, then new neutral files will also be exported. If this model also publishes a VRML service, then if the CAD plug-in is currently set to export VRML files, a new VRML file will be generated. A schematic of this process can be seen in Figure 2.14.

Figure 2.14 CAD Plug-In Export Interaction Information Flow Diagram
When dealing with a model that subscribes to neutral file services, any changes made to the modifiable dimensions would cause the CAD system to update the CAD geometry if the CAD plug-in is currently set to react to changes made to the design parameters. If the CAD plug-in is currently set to import files, then new neutral files will also be extracted and incorporated into the heterogeneous CAD assembly once older instances are swapped out of the assembly. If the model has updated successfully, the DOME interface will be updated with new values. If this model also publishes a VRML service, then if the CAD plug-in is currently set to export VRML files, a new VRML file will be generated. A schematic of this process can be seen in Figure 2.15.

Figure 2.15 CAD Plug-In Import Interaction Information Flow Diagram
The proposed process for designing heterogeneous CAD assemblies through DOME was implemented and tested for DOME v0.74 and was found to work successfully. It is expected that once DOME^3 is completed and the new CAD plug-ins integrated into it, the process will still work successfully along with its additional functionalities. This thesis did not emphasize the difference between the CAD plug-ins developed for DOME v0.74 versus those developed for DOME^3. This thesis only emphasized the features integrated into the CAD plug-ins for DOME^3. To understand the features developed for some of the CAD plug-ins for DOME v0.74, please refer to the work of Bill Liteplo [Liteplo 2000].
Chapter 3: DOME VRML Module

I. Introduction

The DOME VRML module was specifically designed to provide the heterogeneous CAD assembly designer with the best possible visual environment to control all aspects of her design by giving her the ability to manipulate all components within her assembly as if they were native geometry. That means, through this environment, the assembly designer can directly modify the dimensions of feature parameters of her suppliers' CAD geometries even though the only files she have received from her suppliers are neutral files.

This chapter will first present an overview of key features offered by this module. It will be followed by detailed descriptions of those featured and definition of required elements of the VRML module. And lastly, a summary of main features of the VRML module and how information propagates between the VRML module and the CAD plug-in during an active DOME session will be presented.

II. Overview of New Features Offered

The VRML module not only allows designers to modify feature parameters of a component and monitor their progress, set limits on any design parameters, and warn users when those limits are violated, but it also allows the assembly designer to traverse
her assembly and her supplier’s assemblies to make modifications to design parameters. In addition, for every part component the designer traverses, she can save its design configurations, recall those configurations, and arrange their stored order.

Through the VRML module, the user can view the actual dimension objects through the VRML browser and select them for instant modification. In this thesis, a dimension object refers to the set of line segments and text that denote the location and name of a feature parameter. It is believed that by providing the designer with a visual aid for identifying feature parameters accessible during a DOME session, she will need less time trying to recall exactly which feature parameter she can modify based on its name alone as per the DOME interface.

The VRML module also allows anyone who may not currently have a suitable CAD system installed on his computer to fully control all design parameters associated with a DOME CAD model. In addition, when working with a heterogeneous CAD assembly, this user can even control the design parameters of components from all her suppliers. The only catch is that all parties involved in this particular DOME session must be currently connected to DOME and one another so that any modifications made by this user may propagate to the appropriate system to enable necessary updates.

III. VRML Module Functionalities and Requirements

The VRML module requires several inputs, in the form of text files, before it can start properly. These files are automatically generated by the CAD plug-ins, so the user does not have to worry about how they are generated or when to generate them. The
required files are: a "<component>.html" file, a "<component>.dat" file, and a "<component>#.wrl" file. Where <component> refers to the name of the CAD geometry.

The "<component>.html" file is simply an html file that contains an applet that invokes the VRML module and unique values the VRML module requires. The name of this html document corresponds to the name of the CAD component the user will interact with through DOME. In order for the component name to be valid, it must not contain any non-alphanumeric values. The VRML module is run within an Internet browser and to activate the module within the browser, the browser must be set to display the "<component>.html" file. DOME will automate this process.

The "<component>.dat" file is simply a data file. Details regarding the type of information contained within this file is presented in the Design Interface section. This file is extremely important, as it contains information regarding the design parameters and assembly tree data, among other things. Without this file, the VRML module will be unable to provide the means for the user to traverse the assembly or modify and view the design parameters.

The "<component>#.wrl" file is simply a VRML file. Without this file, it will be impossible for the user to visualize the component through the VRML browser. The "#" is merely used in facilitating the switching of VRML scenes for a component from one version to the next without overwriting the currently loaded VRML scene. This method has several advantages: it avoids the case that the VRML module would crash when objects it points to in one existing scene no longer exists as that scene is overwritten; and, all information stored during the active VRML session would not be lost, as would be the case if the Internet browser were to refresh to update the VRML scene was used instead.
Details regarding the type of structures contained within this file that enable the effective communication between the VRML scene and the VRML module is presented in the Required VRLM Components section.

The VRML module is composed of four main components: the viewing area (which is merely an area for viewing a VRML scene through a VRML browser), the Design Interface, the Component Manager, and the Configuration Manager. They can be seen in Figure 3.1. In order for the VRML module to function properly, the user must have installed on his system a VRML 2.0 compatible browser, such as the one provided by ParallelGraphics [ParallelGraphics 2002] as seen in Figure 3.1.

Figure 3.1 DOME VRML Interface
The VRML module is created using Java, Swing, and EAI (External Authoring Interface) and compiled as a Java applet. Interaction between the VRML scene and the Java GUI was enabled through the EAI, which is basically an API to VRML browsers. The Design Interface, Component Manager, and Configuration Manager modules can be accessed from the Java GUI by selecting the desired tabbed pane.

III.A. The Design Interface

The Design Interface, which can be seen in Figure 3.2, is the main module that assists designers in modifying feature design parameters and setting constraints on them.

![Figure 3.2 The VRML Module Design Interface](image-url)
This module will display mass property calculations when either an assembly component or part component is loaded into the VRML scene. When a component’s VRML scene is loaded into the VRML browser, it signals to DOME and the CAD plug-ins that any and all changes made through the VRML module will be propagated to this component. At any one time, only a single component can be selected for interaction through the VRML module. The dimension feature parameters, however, will only show up in the Design Interface if the designer is currently working on a part component.

The "<component>.dat" file mentioned in the previous section is used to load information into the Design Interface and construct the Swing assembly tree in the Component Manager. This file is generated automatically by the CAD plug-ins when the user specifies that a VRML service needs to be published through DOME. It follows the following format:

0 <component name> <GUI component index>
1 <visible modifiable dimension name> <modifiable dimension value> <modifiable dimension unit> <modifiable dimension index>
2 <invisible modifiable dimension name> <modifiable dimension value> <modifiable dimension unit> <modifiable dimension index>
3 <visible dependent dimension name> <dependent dimension value> <dependent dimension unit> <dependent dimension index>
4 <invisible dependent dimension name> <dependent dimension value> <dependent dimension unit> <dependent dimension index>
5 <mass property calculation name> <mass property calculation value> <mass property calculation unit> <mass property calculation index>
-1 <assembly component name> <assembly component index> <GUI parent index>
-2 <part component name> <part component index> <GUI parent index>
-3 <imported assembly component name> <imported assembly component index> <GUI parent index>
-4 <imported part component name> <imported part component index> <GUI parent index>
-5 <imported child assembly component name> <imported child assembly component index> <GUI parent index>
-6 <imported child part component name> <imported child part component index> <GUI parent index>
The data file used to construct the VRML module interface of Figure 3.2 can be seen in Figure 3.3.

```plaintext
0 "t2_piston" 3
 1 "piston_diameter" 30.0000 "millimeter" 0
 1 "piston_length" 60.0000 "millimeter" 1
 5 "1X" 0.0000 "unitless"
 5 "volume" 42411.4978 "cubic millimeter"
 5 "solidSurfaceArea" 7068.5829 "square millimeter"
 5 "openSurfaceArea" 0.0000 "square millimeter"
 5 "mass" 53014.3723 "tonne"
 5 "cg_z" 45.0000 "millimeter"
 5 "Ixy" 0.0000 "tonne square millimeter"
 5 "I33" 18886787.5580 "tonne square millimeter"
-1 "Product" 0 -1
-1 "Piston_assembly" 1 0
-1 "triangle" 2 0
-2 "t2_piston" 0 1
-2 "t2_block" 1 0
```

**Figure 3.3 Sample Part Component Data File**

All visible dimension objects will have a graphical representation in the VRML browser. The user has two ways of selecting a visible dimension object: through the Design Interface GUI or the VRML browser. When the user selects the dimension object from the Design Interface GUI, the GUI element will highlight in blue if its corresponding element in the VRML scene exists or orange if it does not. If this element exists in the VRML scene, it will highlight in red if it corresponds to a dependent dimension or yellow if it corresponds to a modifiable dimension. If the user selects the dimension object through the VRML browser, then the same process would occur with the dimension object in the VRML scene highlighting in the appropriate color and the
Design Interface GUI element highlighting in the appropriate color as well. Figure 3.4 demonstrates what happens when one of the modifiable dimension objects is selected.

Figure 3.4 Selecting Modifiable Dimension Objects through the VRML Module

If the selected dimension object is a modifiable dimension, then its full name, value, and unit will be displayed in the text area under the “Selected Mod. Dimension” label and its corresponding value next to the “Value” label. To modify this dimension’s value, the user should edit the value displayed next to the “Value” label and hit the “Set” button. If the value entered is invalid, in that it does not contain any numeric component that can be extracted, then a warning dialog window will appear. This can be seen in Figure 3.5.
Oftentimes in design, constraints are set to certain design parameters to guarantee that the final product meets all design criteria. To meet this particular need, the Design Interface allows users to set constraints on the design parameters in the forms of minimum and maximum allowable values limits. The user can either set the maximum limit bound, the minimum limit bound, or both. She can also modify and delete them if she chooses to later on during her design session.

Each limit bound can only be set one at a time. That is, to set both the maximum and minimum limit bounds, the user needs to perform two sets of processes. To set one of the limit bounds associated with a particular design parameter, simply right mouse
click to bring up a submenu. If a bound has already been set, its value can be seen at the bottom of this submenu. This can be seen in Figure 3.6.

Figure 3.6 Setting Limits on a Design Parameter

To actually set a limit constraint, the designer needs to select one of the options: “set min” or “set max”. This will bring up a pop-up dialog box, within which the designer can enter her value (only numerical values are allowed) and then hitting the ENTER key or pressing the “Set Range” button to implement it, or delete a pre-existing constraint by pressing the “Remove Range” button. This can be seen in Figure 3.7.
If any of the constraint bounds has been entered for a design parameter, then every time the design parameter is updated during or after design iterations (in conjunction with the CAD system), a check will be performed to determine whether any of the bounds were exceeded. If the designer entered a value for the modifiable dimension that exceeded any of its constraints, then the value will be set to the appropriate limit. If any of the dependent dimension or mass property calculation exceed their limits, the designer will be presented with a visual warning indicating which component(s) failed to satisfy its design criteria and which criteria were violated.
To view a part component free of dimension objects, the designer simply needs to press the “Hide Dimension” button located at the bottom of the Design Interface. Once pressed, the “Hide Dimension” button will change to the “Show All Dimensions” button, which when pressed, will make all dimension objects visible in the VRML browser once more. This can be seen in Figure 3.8.

Figure 3.8 Hiding Dimension Objects

The Design Interface is meant to be the main interface utilized by the designer when working on a part component through the DOME VRML module. When working on an assembly component, the Design Interface merely allows the designer to view mass property design parameter values and set their limits.
If, at any time following the update of the DOME VRML module, any of the design parameter constraints for any of the components modifiable through DOME are exceeded, a warning dialog box titled “Exceeded Design Parameters!” will pop up signaling the names of those components. This can be seen in Figure 3.9.

![Figure 3.9 Exceeding Design Parameters](image)

In addition, a red warning icon will appear next to the category label under which the design parameter constraint was exceeded. There are only two places this icon can appear: next to the Mass Property Calculation label or the Dependent Dimensions label. The warning icon will not appear next to the Modifiable Dimensions label because the
DOME VRML module accounts for all user inputted values and as such, automatically limits the range of the allowable values before passing them on to the DOME CAD plugins for implementation in the CAD system. The mass property calculations and dependent dimensions, on the other hand, cannot be automatically compensated for by the DOME VRML module. Since it would be beneficial for the user to understand specifically which design parameter had exceeded its bounding limit(s), the word "EXCEEDED" will appear next to that object. This can also be seen in Figure 3.9.

To revert the system to its previous state so that all design limiting bounds can be satisfied, all the user has to do is go to the Component Manager and press the "Revert to Previous State" button. This will revert the entire CAD system to its previous successfully updated state. More information on the properties of the Component Manager can be found in the next section.

III.B. The Component Manager

The Component Manager is the main interface utilized by the designer when working on an assembly component through the DOME VRML module. It can be seen in Figure 3.10.
The purpose of the Component Manager is two-fold: to facilitate in the navigation of assembly components during an assembly design session, and to control when the Design Interface and VRML scenes are updated. The currently displayed VRML scene corresponds to the active component and its name is listed under the “Component” label. Directly below it is a pull down menu that the designer can use to select a new component to switch to. It can be seen in Figure 3.11.
Another method of traversing the root assembly is to simply use the Swing tree displayed in the Component Manager. As previously stated, the Swing assembly tree is constructed based on information provided by component data file. To review details on what the component data file provides, please refer back to the Design Interface section.

To better assist the designer in distinguishing between part and assembly components, different graphical icons were used to denote a part component (an icon with a solid color) and an assembly component (an icon with multi-colors). For an assembly component, its branches can be collapsed or expanded.
When selecting a component either through the pull down menu or the Swing tree, the bounding box associated with that component will become visible in the VRML browser. It should be noted here that selecting the root assembly (top assembly node) would not cause its bounding box to appear in the VRML browser. But it will clear the VRML scene of any visible bounding box. For the Component Manager, only one component at any time can be selected. Multiple selections are not supported.

To switch to a different component, the designer needs to press the “Switch Component” button after she either selects a component from the Swing tree or the pull down menu. This will cause the CAD plug-in to update the system with all changes made prior to the switch, generate a new VRML scene and data files, and transfer them over to the VRML module so it can update its browser with the new scene, update the Design Interface with the new values, and recall configurations associated with this component in the Configuration Manager, if they exist. This process will occur regardless of the values set to “Update Values” and “Update Visuals” on the Configuration Manager. It should be noted here that all configurations associated with a component are stored whenever components are switched. So, the designer never has to worry about loss of data. More information about the Configuration Manager is presented following this section.

Settings that control how often the VRML updates its browser and Design Interface are found on the lower portion of the Component Manager. These are the “Update Values” and “Update Visuals” settings. The “Update Values” setting controls when the Design Interface needs to be refreshed based on changes made to the modifiable dimension design parameters of a part component. The “Update Visuals” setting controls
whether or not the VRML browser updates with a new scene when changes are about to be propagated to the CAD system. By default, these settings are set to “No” since it may be inefficient for the CAD plug-in to update the CAD geometry every time a single change is made to the design parameter of a component. The designer should exercise appropriate judgment when deciding when to set either of the update settings to “Yes”.

If, by chance, updating the CAD geometry caused any of the design parameters to exceed their constraints, then the designer can undo all changes made to her geometry by pressing the “Revert To Previous State” button found at the bottom of the Component Manager. There is a limitation on how far back the user can undo changes made to her CAD geometry. The user can only go back one level of undo. But she can do this for only part components. It makes no sense to undo any changes made to an assembly component since it contains no modifiable design parameters that can be modified to affect the underlying geometry.

The final major component to the VRML module not yet discussed in detail is the Configuration Manager. It too is meant to be a tool best utilized when the designer is working on a part component.

III.C. The Configuration Manager

When evaluating a design, the designer often needs to keep track of changes she made to a design and record their impact on other design parameters. The Configuration Manager provides her with those necessary tools. This can be seen in Figure 3.12.
The Configuration Manager allows the designer to record configurations and arrange them in the order she prefers. In this thesis, a configuration is referred to as a set of design parameters associated with a single state of a design. Visually, the Configuration Manager resembles that of the Design Interface. The actual tools themselves reside on the lower panel of the Configuration Manager.

To record the current configuration, simply press the “Record” button. This will cause the “Store Configuration” pop-up dialog box to appear prompting the user to enter a name for the configuration to be saved. It can be seen in Figure 3.13.
Simply enter the name of the configuration in the dialog box to store it for future purposes and hit “Save” or ENTER. The name entered does not necessarily have to be unique, but it would help the designer if it were.

Once a configuration has been saved, the designer can recall it by selecting its name from the pull down menu and pressing the “Show” button. For example, recalling the configuration named “two” will cause the Configuration Manager to load the stored design parameter values into the Configuration Manager. The recalled design configuration name will also be displayed under the “Design Configuration” label signaling the name of the currently recalled configuration, as seen in Figure 3.14.
Figure 3.14 Recalling a Design Configuration

If the designer decides to remove a configuration, she can do so by pressing the "Delete" button, which will bring up a warning dialog prompting the user for verification. This can be seen in Figure 3.15.
Once the designer has several configurations saved, she can then start ordering the configurations based on her design preferences. To do so, simply press the “Order Configuration” button to bring up an interactive window. This can be seen in Figure 3.16.
Within the "Arrange Configuration" window is a list of names of all saved configurations. To move configurations around the list, simply select the desired configuration and press the appropriate up or down action buttons to move it along the list. The pull down menu will automatically update to reflect all changes made.

IV. Required VRML Components

The VRML file is composed of defined geometries that when parsed by the VRML browser, will display said geometries. This means that if dimension objects or
bounding boxes exist for a VRML scene, then they need to be defined within this VRML file. Unfortunately, neither the Unigraphics API nor the I-deas API support the selective export of dimension objects nor the automatic export of component bounding boxes. It then becomes the task of the CAD plug-ins to fulfill this need.

For the Unigraphics system, the VRML exporter does not support the export of any other object aside from solids. This means that the exported VRML file would need to be edited by the Unigraphics plug-in after the plug-in extracts all relevant information regarding those dimension objects and add them to the VRML scene. The I-deas system does support the export of dimension objects along with the VRML file. But they could not be selectively exported. This means that the exported VRML file would also need to be edited by the I-deas plug-in to remove unnecessary objects from the VRML scene. A typical VRML dimension object will have the following form in the VRML file once it is added by the plug-in:

```
Group { children [
  Transform { children [
    DEF Tin1 TouchSensor {} 
    Shape {
      appearance Appearance { material DEF Min1a Material {
        diffuseColor 1 1 1 emissiveColor 1 1 1 transparency 0.0 } }
      geometry IndexedLineSet {
        coord Coordinate {
          point[0.030017 0.034213 0.045, 0.030017 -0.025787 0.045, ...] }
        coordIndex[0,1,-1,2,3,-1,4,5,-1,0,6, 7,0,-1,1,8,9,1] }
    Transform {
      translation 0.03001678 0.004213208 0.045 
      children [ Billboard { axisOfRotation 0 0 0 
        children [ Shape {
          appearance Appearance { material DEF Min1b Material {
            diffuseColor 1 1 1 emissiveColor 1 1 1 transparency 0.0 } }
          geometry DEF TxtIn1 Text { string [ "piston_length=60.0000" ]
            fontStyle FontStyle { size 0.0077 } }
```

Chapter 3: DOME VRML Module
A “Group” structure is used to group all required elements of a dimension object, which includes the line segment geometry and dimension name text, together. A “TouchSensor” is required to detect when this object is touched. If this object was activated by the designer’s touch, then the “TouchSensor” will generate a signal that can be used to route other signals, whose eventual goal is to switch on the selected dimension object to the appropriate highlight color. In the provided sample, “Min1a” is the defined material name for the line segments that this dimension object is composed of and “Min1b” corresponds to that of the dimension text. By default, all unselected dimension objects will appear white if they correspond to modifiable dimensions and blue if they correspond to dependent dimensions. To properly route the signals, the following routing structure needs to be defined for each dimension object:

ROUTE Tin1.isActive TO SIn1.highlight
ROUTE MatSet.shininess TO SIn1.listSelect
ROUTE SIn1.index TO RecordIn.record
ROUTE RecordIn.store TO SIn1.turnOff
ROUTE SIn1.colorTo TO Min1a.set_diffuseColor
ROUTE SIn1.colorTo TO Min1a.set_emissiveColor
ROUTE SIn1.colorTo TO Min1b.set_diffuseColor
ROUTE SIn1.colorTo TO Min1b.set_emissiveColor
ROUTE MatSet.transparency TO Min1a.set_transparency
ROUTE MatSet.transparency TO Min1b.set_transparency

Where, as you can see, a signal from the “TouchSensor” is used to trigger other values. For every dimension object, a script structure needs to be defined and used to control when the dimension object should set its color to highlight or not. Most importantly, this script structure will communicate not only to the VRML scene but also through EAI calls to the VRML module, which dimension object was just selected by the user in the VRML browser. The required script takes on the following structure:
DEF SIn1 Script {
  eventIn SFBool highlight eventIn SFInt32 turnOff eventIn SFFloat listSelect
  eventOut SFColor colorTo eventOut SFInt32 index eventOut SFBool wasTurnedOn
  field SFColor colorHighlight 1 1 0 field SFColor colorOriginal 1 1 1

  url "javascript:
  function initialize() { index = -1; colorTo = colorOriginal; wasTurnedOn = false; }
  function highlight(active) {
    if (active)
      { colorTo = colorHighlight; index = 1; wasTurnedOn = true; }
  }
  function turnOff(value) {
    if (value != 1 && wasTurnedOn)
      { colorTo = colorOriginal; wasTurnedOn = false; }
  }
  function listSelect(value) {
    if ((value < 1.1) && (value > 0.9) && !wasTurnedOn)
      { colorTo = colorHighlight; index = 1; wasTurnedOn = true; }
    else
      { if ((value < -0.9) && (value > -1.1))
          index = -1;
      }
  }
"}

The script needs to be defined with a unique name and it needs to have associated with it various functions. The “index” value is used by the VRML module to identify the selected dimension and its type, “colorTo” sets the color of the dimension object, and “wasTurnedOn” records whether the object was just selected by the designer or not. To communicate the value of “index” to the VRML module, the following script needs to be defined before any geometry is defined in the VRML scene. This module only needs to be defined once and it takes the following form:

DEF RecordIn Script {
  eventIn SFInt32 record eventOut SFInt32 store
  url "javascript:
  function initialize() { store = -1; }
  function record(value) {
    store = value; print(store); }
"}
To control the highlight of dimension objects from the VRML module’s Design Interface, a dummy material “MatSet” is used. Through EAI, the VRML module is able to set values defined within “MatSet” and route them to the appropriate functions to trigger the highlight of the selected dimension. Basically, when a dimension element is selected within the Design Interface, it sends a signal to “MatSet” that corresponds to the “index” value set in a script when the “TouchSensor” is activated. The required “MatSet” structure, has the following form:

```
Shape { appearance Appearance {
  material DEF MatSet Material { transparency 0.0 shininess -1.0 ambientIntensity -1.0
  } } }
```

From the “MatSet” structure, “transparency” is used to trigger when all dimension objects should be visible or hidden from the scene. The value of “transparency” is set by the “Hide All Dimensions” (“Show All Dimensions”) button of the Design Interface. The other values, “shininess” and “ambientIntensity” are used to trigger the dimension object and bounding box highlights, respectively.

The bounding box structure resembles the dimension object’s structure in form and function. All values required to construct the eight corners of this box are extracted by the CAD plug-ins. A typical bounding box object will have the following form:

```
Group { children [ Transform { children [ Shape { geometry IndexedLineSet {
  coord Coordinate { point [ 0.7500 -0.4000 0.3750, 0.7500 -0.4000 2.6250, 0.7500 -3.9625 0.3750, 0.7500 -3.9625 2.6250, -1.5000 -0.4000 0.3750, -1.5000 -0.4000 2.6250, -1.5000 -3.9625 0.3750, -1.5000 -3.9625 2.6250 ] } coordIndex [ 0 1 1 1 3 -1 3 2 -1 2 0 -1 0 4 -1 4 5 -1 5 1 -1 4 6 -1 6 2 -1 6 7 -1 7 3 -1 5 7 ] color Color { color [ 1 1 1, 1 1 1, 1 1 1, 1 1 1, 1 1 1, 1 1 1, 1 1 1, 1 1 1, 1 1 1, 1 1 1, 1 1 1, 1 1 1, 1 1 1, 1 1 1, 1 1 1, 1 1 1 ] } appearance Appearance { material DEF Mcomp1 Material { diffuseColor 1 1 1 emissiveColor 1 1 1 transparency 0.0 } }
} ] ] ]
```
And its matching script resembles the following:

```javascript
DEF Comp1 Script {
  eventIn SFFloat turnOff eventIn SFFloat highlight
  eventOut SFBool wasTurnedOn eventOut SFFloat transparency

  url "javascript:
  function initialize() { wasTurnedOn = false; transparency = 1.0; }
  function turnOff(value) {
    if (((value > 1.1) || (value < 0.9)) && wasTurnedOn)
      { wasTurnedOn = false; transparency = 1.0; }
  }
  function highlight(value) {
    if ((value < 1.1) && (value > 0.9) && !wasTurnedOn)
      { wasTurnedOn = true; transparency = 0.0; }
  }
}"}
```

The above script behaves similarly to the ones for the dimension objects. But the number of required routing statements is fewer:

ROUTE MatSet.ambientIntensity TO Comp1.highlight
ROUTE MatSet.ambientIntensity TO Comp1.turnOff
ROUTE Comp1.transparency TO Mcomp1.set_transparency

V. Summary and Conclusions

The VRML module provides the designer with a new GUI from which to conduct her DOME CAD design session. It is capable of handling heterogeneous CAD assemblies and allows the assembly designer to work with imported components as if they are native components. Greater detail on how this is so is provided in the next chapter, which details a case study setup and results. The key features of the VRML module are its ability to modify design parameters, set design limits, traverse assembly components seamlessly, and manage design configurations. An information flow block("
diagram depicting the flow of data between the VRML module and the CAD plug-in can be seen in Figure 3.17.
Several signals need to be communicated to the CAD plug-in from the VRML module. These signals will indicate to the CAD plug-in which component changes need to be made to. A file containing the values of the modifiable dimensions associated with this component will also be transferred over for scanning purposes. Once the CAD plug-in finishes scanning that file and extracting new values to be applied to the design feature parameters, it will apply changes to the CAD geometry. Once the CAD system is done updating the geometry, it will export the appropriate number of files to the VRML module. Once these files are received, the VRML module will update its GUI and load a new scene into its browser, if need be.

In experimental trials, as presented through pictures incorporated into this chapter, the tools developed for the DOME VRML module have performed consistently and correctly. It is expect that once incorporated into DOME\textsuperscript{3}, this tool will continue to perform just as well, all the while making itself an indispensable tool to the heterogeneous CAD assembly designer.
Chapter 4: Case Study

I. Introduction

The tools developed for this thesis are complex in implementation, but relatively easy to use. In this chapter, a case study on building a heterogeneous CAD assembly through the DOME VRML interface is presented. Two assembly designers are involved in this case study: the rear suspension designer who works in I-deas, and the flange assembly designer/supplier who works in Unigraphics. The assembly process takes place in the rear suspension designer’s CAD environment, while the flange assembly designer supplies his design to the rear suspension designer in the form of a STEP neutral file. For simplicity, the rear suspension was reduced to half a suspension for this case study.

To properly prepare their models for an interactive, collaborative design session through DOME, the assembly designer and her supplier(s) must perform a series of tasks in a specific order. This chapter is structured in such a manner as to follow those tasks in their intended order and they are:

1. Supplier (prior to launching DOME): Export his component(s) for the assembly designer; run the appropriate density extraction program to generate the required “density.txt” file; and, transfer those files to the assembly designer.

2. Supplier (prior to and after launching DOME): Create the required “<component>.in” input text file to interface with the DOME CAD plug-in and set the offset index value to 0; run the DOME CAD plug-in, and transfer the
generated "<component>.txt" file to the assembly designer. The "<component>" refers to the name of the component the supplier wishes to export.

3. Assembly designer (prior to launching DOME): Store all components to be imported into her assembly in the "import" directory; consolidate all received "density.txt" files into a single "density.txt" file; run the appropriate import prepping program that extracts all imported files into her assembly; and, apply relevant mating conditions to properly position the imported components within her assembly.

4. Assembly designer (prior to and after launching DOME): Create the required input text file to interface with the DOME CAD plug-in (making use of the "<component>.txt" files received for each neutral file component to be imported into her assembly from her suppliers); run the DOME CAD plug-in, and examine the exported "<component>.dat" file that serves as an input to the DOME VRML module to extract the correct offset index value(s) for the supplier(s).

5. Supplier (during a DOME session): Update his offset index value, through the DOME CAD GUI, to the value supplied by the assembly designer.

6. Assembly designer (during a DOME session): Establish appropriate DOME relations with the supplier(s) to ensure proper transfer of required files; activate the DOME VRML module, and start working on the design of her heterogeneous CAD assembly.
II. Supplier Preparation Process: Setting Up the Flange Assembly

The flange assembly ("flange_assembly") is created in Unigraphics and is composed of a seal assembly ("seal_assembly") and a flange fin ("flange_fin"). The seal assembly is composed of two seals: the outer seal ("seal1") and an inner seal ("seal2"). The part components are parametrically editable and contain some interpart relations. Interpart relations define the dimension of one part in terms of dimension(s) from other part(s). These components can be seen in Figures 4.1 – 4.5.

(a) Isometric View  
(b) Side View  

Figure 4.1 Flange Assembly

(a) Isometric View  
(b) Side View  

Figure 4.2 Seal Assembly
Figure 4.3 Outer Seal

(a) Interior View
(b) Exterior View

Figure 4.4 Inner Seal

(a) Interior View
(b) Exterior View

Figure 4.5 Flange Fin

(a) View 1
(b) View 2
The flange assembly designer should first export his component manually from his CAD system, and specify the name of the exported file as "s_flange_assembly". That is, simply add the appropriate prefix ("s_" for STEP and "i_" for IGES) to the name of the original native component he wishes to export, in this case, the "flange_assembly".

Next, he should run "ug_density.dll" to generate the needed "density.txt" file. The resulting file generated can be seen in Figures 4.6.

![Figure 4.6 Flange Assembly “density.txt” File](image)

| flange_fin | 0.00008826 |
| seal1      | 0.0860122  |
| seal2      | 0.0860122  |

The program "ug_densiy.dll" is supplied with DOME and is a standalone application. The generated density values are in the units of "lb in". The flange assembly supplier should then transfer his exported STEP and "density.txt" files to the rear suspension assembly designer for incorporation into her heterogeneous CAD assembly.

To make the flange assembly accessible through DOME, a required input text file needs to be created for the Unigraphics plug-in following the format outlined in Chapter 2 section IV.A. This input file can be seen in Figure 4.7, where the offset index is initially set to 0, the VRML component index is initialized to 0, and the type of VRML update option is set to -1. If need be, please refer back to the previous chapters to review the significance of those values.
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\t_UG</td>
<td>0 0 -1</td>
</tr>
<tr>
<td>flange assembly</td>
<td></td>
</tr>
<tr>
<td>mass</td>
<td>1 kg m</td>
</tr>
<tr>
<td>volume</td>
<td>4</td>
</tr>
<tr>
<td>step</td>
<td>5</td>
</tr>
<tr>
<td>flange_fin</td>
<td></td>
</tr>
<tr>
<td>mass</td>
<td>1 kg m</td>
</tr>
<tr>
<td>properties</td>
<td>2 p11 2 p26 2 D3 3 p24 4 Xcbar 4 Ycbar 4 Zcbar</td>
</tr>
<tr>
<td>seal assembly</td>
<td></td>
</tr>
<tr>
<td>density</td>
<td>1 kg m</td>
</tr>
<tr>
<td>mass</td>
<td>4</td>
</tr>
<tr>
<td>volume</td>
<td>4 lxx 4 lyy 4 lzz</td>
</tr>
<tr>
<td>seal1</td>
<td></td>
</tr>
<tr>
<td>mass</td>
<td>1 kg m</td>
</tr>
<tr>
<td>properties</td>
<td>2 p11 3 p19 4 mass 4 volume</td>
</tr>
<tr>
<td>seal2</td>
<td></td>
</tr>
<tr>
<td>mass</td>
<td>1 kg m</td>
</tr>
<tr>
<td>properties</td>
<td>3 p8 4 Mxc 4 Myc 4 Mzc</td>
</tr>
</tbody>
</table>

Figure 4.7 Unigraphics CAD Plug-In “flange_assembly.in” Input File
The offset index will need to be updated later by the flange assembly designer, through his DOME session, in order to ensure that subsequently generated VRML files from his CAD system conform to the rear suspension designer's DOME VRML session requirements. More detail regarding what those requirements are will be presented in the next section.

After the DOME CAD session initializes successfully, the CAD plug-in will generate a "flange_assembly.txt" file. This file will contain a list of all the distinct children components of the flange assembly that the supplier has made available through DOME based on his "flange_assembly.in" file. The "flange_assembly.txt" file can be seen in Figure 4.8.

```
flange_fin 1 0 1
seal_assembly 2 0 -1
seal1 3 2 1
seal2 4 2 1
```

Figure 4.8 Flange Assembly "flange_assembly.txt" File

The name of the child component is listed first, followed by three integer values. The first value, to be referred to as the component index for the rest of this section, is determined based on the order in which the user specified the child components in his "flange_assembly.in" file. If an earlier specified child component could not be found within the flange assembly, then the next child component found will take on a component index value sequential to the component index value of the previously found child component. For example, let us say that the following children components are specified in the following order in the input text file: child1, child2, child3, and child4.
However, child2 does not exist within the assembly. Then, the generated assembly text file will list child1 with a component index of 1, child3 with a component index of 2, and child4 with a component index of 3. The component index of 0 is exclusive to the top assembly node and is never assigned to any of the children components.

The second value, to be referred to as the parent index for the rest of this section, indicates the component index of the child’s immediate parent, as recognized by the DOME CAD session. That is, this parent may not be the actual immediate parent of the child component within the CAD model. However, it is the immediate parent of the child component as seen through DOME if an assembly tree were to be constructed based on the “flange_assembly.txt” file alone. For the “seal_assembly”, whose children are “seal1” and “seal2”, we see that the parent index of those seals correspond to the component index of 2 for the “seal_assembly”. If the parent index of the child component is 0, then this means that its immediate parent, as recognized by DOME, is the root assembly, which in this case is the flange assembly.

The third value, to be referred to as the component type index for the rest of this section, is merely used to indicate whether the child component is a part or an assembly within the flange assembly. It will have a value of 1 if the component is a part and -1 if the component is an assembly.

The “flange_assembly.txt” file should be transferred over to the rear suspension designer so she can properly construct her “rear_suspension.in” input text file for the DOME I-deas plug-in. All the flange assembly designer needs to do now is to wait for the rear suspension designer to connect to DOME so she can notify him of his new offset index value.
III. Assembly Designer Preparation Process: Setting Up the Half Rear Suspension Assembly

The half rear suspension assembly ("Rear_Suspension") is created in I-deas and is composed of roughly 130 part instances. Like the flange assembly of the supplier’s, this assembly also contains parts that are parametrically editable. Two native I-deas part components selected from this assembly for design through the DOME session are the carrier fin ("carrier_fin") and the rear auxiliary suspension ("SPG_ASY_AUX"). The half rear suspension assembly can be seen in Figure 4.9, where the carrier fin is in dark blue and the auxiliary suspension in magenta.

Figure 4.9 Half Rear Suspension Assembly
Due to problems encountered with some components within the rear suspension assembly during mass property analysis, the rear suspension assembly was reduced to nearly 80 part instances.

The rear suspension assembly designer should first create an "import" subdirectory under her model’s directory. That is, if her I-deas model is located in “C:\t_ID”, then there should now be a “C:\t_ID\import” directory. All neutral file components to be imported into her assembly should be placed there along with the consolidated “import.txt” and “density.txt” files. Since the rear suspension designer will only be importing a single neutral file component into her assembly, her “density.txt” file is simply the file the flange assembly designer has transferred to her and her “import.txt” file can be seen in Figure 4.10.

| s_flange_assembly.stp |

Figure 4.10 Rear Suspension Designer’s “import.txt” File

To import the neutral file component into her assembly and have the proper density value automatically assigned to the extracted part components, the assembly designer needs to invoke an I-deas API program, “import.exe”, from her I-deas Command Prompt. This requires that the assembly designer have her I-deas session currently open to the rear suspension assembly. The I-deas API program “import.exe” is supplied with DOME and is a standalone application. It can be placed anywhere on her system. If the “import.exe” file is located in the “C:\API” directory and the assembly designer’s I-deas server is located on the server12 host, then she should issue the
following command at the Command Prompt to execute "import.exe": "C:\API\import.exe server12". The imported components are placed in the "import" bin within the assembly designer’s l-deas model. Once incorporated into her assembly, the resulting rear suspension can be seen in Figure 4.11.

Figure 4.11 Assembled Half Rear Suspension
To make the rear suspension assembly accessible through DOME, a required "Rear_Suspension.in" input text file needs to be created for the I-deas plug-in following the format outlined in Chapter 2 section V.A. This input file can be seen in Figure 4.12.

```
server12
Administrator
C:\t_ID
sus8
0 0 -1

1 master_bin Rear_Suspension
4 mass
4 cg_x
4 cg_y
4 cg_z
9 Rear_Suspension

0 drive_housing carrier_fin
2 D33
2 D27
3 D32
4 volume
4 solidSurfaceArea
4 openSurfaceArea
4 mass

0 Main SPG_ASY_AUX
2 D50
3 D52
4 volume
4 solidSurfaceArea
4 openSurfaceArea
4 mass

6 s_flange_assembly 1
8 flange_fin 1 0 1
8 seal_assembly 2 0 -1
8 seal1 3 2 1
8 seal2 4 2 1
```

Figure 4.12 I-deas CAD Plug-In "Rear_Suspension.in" Input File
Where the offset index is set to 0, the VRML component index is set to 0, and the type of VRML update option is set to -1.

To assist the DOME VRML plug-in in properly constructing the assembly tree for component traversal in its Component Manager, the assembly designer simply needs to cut and paste the information contained within the “flange_assembly.txt” file into her “Rear_Suspension.in” file. However, those lines must be inserted immediately after the line containing the name of its parent assembly, which in this case is “s_flange_assembly”.

After the DOME CAD session initializes successfully, the CAD plug-in will generate a “Rear_Suspension.dat” file for input to the DOME VRML module. This file is generated only because the assembly designer specified in her “Rear_Suspension.in” file that she wish to export a VRML file for use with the DOME VRML module. This file can be seen in Figure 4.13.

```
0 "Rear_Suspension" 0
5 "mass" 74.4004 "kilogram"
5 "cg_x" 4913.9375 "millimeter"
5 "cg_y" 95.8253 "millimeter"
5 "cg_z" 1520.2203 "millimeter"
-1 "Rear_Suspension" 0 -1
-2 "carrier_fin" 0 0
-2 "SPG_ASY_AUX" 1 0
-3 "s_flange_assembly" 0 0
-6 "flange_fin" 0 3 1
-5 "seal_assembly" 0 3 2
-6 "seal1" 0 5 3
-6 "seal2" 0 5 4
```

Figure 4.13 DOME VRML Module’s “Rear_Suspension.dat” Input File
To extract the correct offset index value for the flange assembly designer, the rear suspension assembly designer needs to carefully examine her “Rear_Suspension.dat” file. Starting with the first line that begins with a negative value, we see that the line containing “s_flange_assembly” is located 3 lines down from the first line containing a negative value (“-1 "Rear_Suspension" 0 -1”). This means that the correct offset index value for the flange assembly designer is 3. After the flange assembly designer has adjusted his offset index value to 3 through DOME, the rear suspension designer can then launch her DOME VRML session and begin designing her assembly through the DOME VRML interface.

IV. Rear Suspension Assembly Designer: Working With the DOME VRML Module

Once the rear suspension assembly designer has activated her DOME VRML session, she will be presented with the interface seen in Figure 4.14. The interface is constructed based on information contained in the “Rear_Suspension.dat” file. The currently loaded VRML scene, that of the rear suspension, is generated by the DOME CAD plug-in based on the VRML component index specified in the “Rear_Suspension.in” file. Every time the user traverses the assembly tree in the Component Manager and swaps to a different component, the VRML component and type index values will change to reflect which component and its associated type has been selected so that the DOME CAD plug-in will know which component it needs to export a VRML file for. The user needs to never worry about these values since they are handled by the DOME VRML plug-in automatically during a DOME session.
To set bounds on any of the design parameters, such as the total mass of the half rear suspension assembly, the assembly designer simply needs to first highlight "mass" from the Mass Property Calculations display area, then right mouse click to bring up a sub menu that presents the user with the option to set the maximum or minimum constraint limits on this design parameter. This can be seen in Figure 4.15.

Selecting the "set max" menu option will bring up a pop up dialog box within which the user can set the maximum limit. For this case study, the maximum limit on the total mass of the half rear suspension system was set to 75 kg. This can be seen in Figure 4.16. Right clicking on "mass" again will bring up the sub menu, but this time, the sub menu will list the maximum range the user has just set to 75. This can be seen in Figure 4.17.
Figure 4.15 Setting Limits on Design Parameters

Figure 4.16 Entering Value for the Design Parameter Bounding Limit
To traverse other components within the rear suspension assembly, the assembly designer merely needs to select the Component Manager tab. Once in the Component Manager, the assembly designer has the option of selecting a component she wishes to switch to from either the pull down menu or from the Swing tree. Figure 4.18 shows the pull down menu, which lists all DOME accessible components within this assembly. Figure 4.19 shows the expanded Swing assembly tree for the half rear suspension system.
Figure 4.18 Component Manager Pull Down Menu

Figure 4.19 Expanded Component Manager Swing Assembly Tree
Once a component has been selected, its bounding box will become visible through the VRML browser. For example, Figure 4.20 shows that the carrier fin was selected.

![Figure 4.20 Expanded Component Manager Swing Assembly Tree](image)

To switch to the carrier fin from the rear suspension, the assembly designer needs to press the “Switch Component” button. Pressing this button will result in switching the VRML scene, as seen in the VRML browser, from the rear suspension to the carrier fin. The Design Interface and Configuration Manager will also refresh to display/recall information associated with the carrier fin. This can be seen in Figure 4.21.
To view the design parameters associated with the carrier fin, simply select the Design Interface tab. To select any of the design parameter dimensions, the assembly designer can either select it from the Design Interface or through the VRML browser. The selection of a dependent dimension design parameter can be seen in Figure 4.22. Selecting a dependent dimension will toggle the color of its associated VRML object from blue to red, where red indicates object selection. Whereas, the selection of a modifiable dimension design parameter will toggle the color of its associated VRML object from white to yellow, where yellow indicates object selection. This can be seen in Figure 4.23.
Figure 4.22 Selecting a Dependent Dimension Design Parameter

Figure 4.23 Selecting a Modifiable Dimension Design Parameter
In Figure 4.23, the modifiable dimension D27 is selected. Once a modifiable dimension object is selected, its name, value, and unit will be displayed under the “Selected Mod. Dimension” label and its current value displayed next to the “Value” label. Rather than change D27, the assembly designer decides to change D33 from 57.65 millimeters to 59 millimeters. This can be seen in Figure 4.24.

![Figure 4.24 Changing a Modifiable Dimension Design Parameter](image1)

The VRML scene first updates to reflect the new value set to the design parameter D33. Afterwards, through DOME coordinated communication between the VRML module and the appropriate CAD plug-in(s), the VRML scene is updated to reflect the changes made to the geometry of the carrier fin and the Design Interface is updated. Since D32 is dependent on D33, we see that its new value is 52.595 millimeters. The
mass property values are also affect by this change to the geometry. They can be seen in Figure 4.25.

![Updated VRML Scene and Design Interface for Carrier Fin]

Figure 4.25 Updated VRML Scene and Design Interface for Carrier Fin

Next, the assembly designer decides to switch to the rear auxiliary suspension, SPG_ASY_AUX. This can be seen in Figure 4.26. The assembly designer also decides she wants to save the current design configuration associated with the SPG_ASY_AUX component. To do so, she needs to switch to the Configuration Manager by selecting its tab. Once in the Configuration Manager, the assembly designer needs to press the “Record” button to bring up a pop up dialog box prompting her for the name she wishes to record the current configuration under. She chooses to name the configuration “one”. This can be seen in Figure 4.27.
Figure 4.26 Rear Auxiliary Suspension

Figure 4.27 Recording a Design Configuration
To recall the configuration "one", simply select it from the pull down menu and press the "Show" button. This results in updating the Configuration Manager’s display area with the recorded values. The updated Configuration Manager can be seen in Figure 4.28.

![Figure 4.28 Recalling a Design Configuration](image)

Switching to the Design Interface, the assembly designer decides to set the modifiable dimension \( D_{50} \) from 510 millimeter to 520 millimeter. This can be seen in Figure 4.29. Once the CAD system updates to reflect change made to the rear auxiliary suspension, one can see that the dependent dimension \( D_{52} \) also changed. The dimension \( D_{52} \) is dependent on \( D_{50} \). This can be seen in Figure 4.30.
Figure 4.29 Modifying a Dimension Design Parameter

Figure 4.30 Updated VRML Scene and Design Interface for Rear Auxiliary Suspension
Having found the current configuration to her liking, the assembly designer decides to save the current configuration under the name “two” through the Configuration Manager. This can be seen in Figure 4.31.

![Figure 4.31 Recording a Design Configuration](image)

The assembly designer finds that she likes configuration “two” more than “one”. So, she decides she wants to arrange her saved configurations associated with the rear auxiliary suspension. To do so, she needs to press the “Order Configuration” button to bring up the “Arrange Configuration” panel. This can be seen in Figure 4.32. By selecting “two” and pressing the up arrow button in the “Arrange Configuration” panel, the assembly designer is able to arrange her stored configurations. This also updates the pull down menu of the Configuration Manager, as can be seen in Figure 4.33.
Figure 4.32 Arranging Stored Design Configurations

Figure 4.33 Resulting Configuration Manager
If the assembly designer later on decides she no longer wishes to store her "one" configuration, then she can delete it from her session by first selecting "one" from the pull down menu of the Configuration Manager and then pressing the "Delete" button to delete it. This will bring up a pop up menu prompting the user to verify whether she actually wants to delete the selected configuration. It can be seen in Figure 4.34.

![Figure 4.34 Deleting a Saved Design Configuration](image)

By pressing "yes" in the "Delete Configuration?" pop up dialog box, the assembly designer will permanently delete configuration "one" from her session. Once a configuration is deleted it cannot be recalled. The remaining design configuration stored for the rear auxiliary suspension can be seen in Figure 4.35.
Thus far, the assembly designer has made changes to her native CAD components. However, now she wishes to make changes to the flange assembly designer’s design. To access the flange assembly designer’s design through DOME, the rear suspension assembly designer needs to switch to the Component Manager panel, select the “s_flange_assembly” component, and press the “Switch Component” button. The result can be seen in Figure 4.36.
The main purpose of the offset index value that the flange supplier needs to set, as previously mentioned in sections 2 and 3, is to ensure consistency in the highlighting of bounding box within the DOME VRML module as its associated component is selected from the Component Manager. As seen in Figure 4.37, the bounding box associated with the inner seal ("seal2") is properly highlighted as it is selected within the Component Manager.

Traversing the assembly tree within the Component Manager, one can access the design parameters associated with the inner seal ("seal2") as seen in Figure 4.38, the outer seal ("seal1") as seen in Figure 4.39, the seal assembly ("seal_assembly") as seen in Figure 4.40, and the flange fin ("flange_fin") as seen in Figure 4.41.
Figure 4.37 Selecting Inner Seal within the Flange Assembly

Figure 4.38 Inner Seal
It should be noted here that there is so far, no visual way of conveying to the user of the DOME VRML interface whether any modifiable dimension(s) associated with an imported component is dependent upon modifiable dimension(s) associated with any native component(s). That is, sometimes when working with imported components, the modifiable dimension associated with that component may be directly tied to the modifiable dimension of a native component through DOME relation(s). For example, if the imported component is a bolt and it goes into a hole in a native bracket, then a relation need to be established defining the diameter of the bolt to be equal to the diameter of the hole in the native bracket. But with the proper DOME relation(s) in place, the assembly designer should never have to worry about modifying dimensional design parameters of imported components through the DOME VRML interface.
It should also be noted here that changes made to the carrier fin does have an effect on the imported components flange fin, inner seal, and outer seal. The values displayed in the Design Interface attest to that. Now, let’s say the rear suspension assembly designer wishes to modify the dimension p26 of the flange fin from 12.7 millimeters to 20 millimeters. She can do so through the Design Interface, as seen in Figure 4.42.

![Figure 4.42 Modifying the Design Parameter of Imported Component Flange Fin](image)

The resulting flange fin can be seen in Figure 4.43 and its impact on the flange assembly seen in Figure 4.44. The total impact of all changes made to components within the rear suspension assembly can be seen in Figure 4.45.
We see that the resulting total mass of the half rear suspension system does not exceed the maximum constraint set at 75 kilograms. If, let's say, the rear suspension assembly designer has set a maximum limiting bound on the center of mass of the flange fin in the y direction (Ycbar) to be at 0.02 millimeters and a maximum limiting bound on the total mass of the flange assembly to be at 0.07 kilograms. If resulting changes made to the flange fin has caused those constraints to be exceeded, then a pop up dialog titled "Exceeded Design Parameters!" will appear notifying the user of all the components that failed to meet their constraints. In Figure 4.46, we see that assembly designer has just switched to the outer flange component.

Figure 4.45 Updated Half Rear Suspension Assembly
But since both the flange assembly and the flange fin have exceeded their design bounding limit(s) after a system update (any time the user switches to a different component, the entire CAD system is automatically updated and new data files generated for the DOME VRML module; this is to ensure that the user will be immediately notified of any adverse effect recent changes made to the CAD system through the DOME VRML module has on other components with regards to their design constraints), we see those components listed within the "Exceeded Design Parameters!" window. To actually see where the design bounding limit(s) is/are exceeded for the flange fin, the assembly designer simply needs to switch to the flange fin. This can be seen in Figure 4.47.
As seen in Figure 4.47, the design limit bound(s) for the dependent dimension p24 was exceeded as well as its center of mass about the y-axis (Ycbar). When any assigned bounding limit is exceeded for a dependent design parameter (dependent dimensions and mass property calculations), a red icon will appear next to the appropriate category label and the word "EXCEEDED" will precede the name of the corresponding design parameter. The warning signals will disappear once the design parameter no longer exceeds any of its constraints.

It should be noted here that warning signals would not be issued if any of the modifiable dimension constraints were exceeded. Since the DOME VRML module always checks changes made to the modifiable dimensions against existing bounds before
forwarding those values to the DOME CAD plug-ins, it will automatically adjust those values to ensure that they stay within their constraints.

It should also be noted here that all mass property calculations displayed for imported components in the Design Interface are calculated with respect to the supplier’s coordinate system. Unlike mass property calculations for native geometry, which are calculated with respect to the root assembly’s coordinate system (for this case study, that of the rear suspension’s).

Unfortunately, there is currently no support for transferring constraints from the supplier’s DOME VRML session to the assembly designer’s. This means that there exists a chance that changes the assembly designer makes to the supplier’s CAD geometry may cause the supplier’s design to fail its design criteria. If the supplier uses the DOME VRML interface for his design session, then he will be able to know whether any changes made to his design by the assembly designer has adversely affected his design. And if the answer is yes, then he needs to notify the assembly designer of it and rectify the situation.

V. Summary and Conclusions

In this chapter, step-by-step instructions were provided for both the assembly designer and her supplier on how they should work together in order to prepare themselves for a collaborative design session through DOME. Overall, the DOME VRML module and CAD plug-ins behaved consistently and in an expected manner. The tools are easy to use and quite powerful.
However, when working with the DOME VRML interface, expect there to be long delays while the VRML browser loads and unloads complex scenes. Aside from some limitations on the tools developed by the author, as mentioned in this chapter, the DOME VRML module has the potential to work very well in an actual interactive design setting as demonstrated in this case study.
Chapter 5: Conclusions and Future Work

I. Conclusions

The trend towards outsourcing components and concurrent engineering has spurned on the need in numerous industries, such as the automotive, to efficiently design heterogeneous CAD assemblies in light of CAD interoperability problems attributed to the lack of a standardized CAD system. A heterogeneous CAD assembly is defined as an assembly composed of geometries from at least two different CAD systems. When working with different suppliers, a company may often find itself in the position of having to work with CAD models designed by its suppliers in dissimilar CAD systems. In order to arrive at a simulated representation of their collaborative design, the company may need to work with heterogeneous CAD assemblies.

In this thesis, a novel method of designing heterogeneous CAD assemblies that was first presented in the Master's thesis of Bill Liteplo [Liteplo 2000] is expanded upon. This method was developed at the MIT CADlab and involves the use of DOME as an intermediary medium for exchanging assembly level parameters to remote native CAD models and coordinating the exchange of up-to-date neutral files. Through this method of approach, the heterogeneous CAD assembly actually becomes parametrically editable, all the while ensuring that no proprietary information is ever divulged during the model exchanging process.
The author has developed the next generation of CAD plug-ins for DOME³. The plug-ins developed for UGS Unigraphics (Version 16) and SDRC I-deas (Version 8) are quite powerful. They are fully capable of handling multiple imported neutral files, be they simple parts or complex assemblies, for heterogeneous CAD assembly designs. In updating the heterogeneous CAD assembly with components extracted from up-to-date neutral files, little, if any, user interaction is required.

In addition, a new visually intuitive environment that allows heterogeneous CAD assembly designers to parametrically modify native and imported components through DOME is also introduced. Assembly tree traversal for both native and imported components is supported, along with the ability to save design configurations on a per component basis. This VRML and Java based module is meant to communicate with the CAD plug-ins through DOME to ensure that all changes the designer makes through the VRML interface will propagate to the CAD system.

A case study was conducted on the design of half a rear suspension system composed of an imported flange assembly. The CAD plug-ins performed to expectation, as did the DOME VRML module. Aside from the noticeable time lag involved in the loading and unloading of the rear suspension VRML scene from the VRML browser used to display the geometry in the DOME VRML module, the overall heterogeneous CAD assembly design experience was rather enjoyable. The tools developed are powerful yet easy to use. They have the potential to revolutionize the way industry conducts collaborative CAD designs in the future.
II. Future Work

The most immediate step towards future work is the actual integration of the developed plug-ins into DOME\textsuperscript{3}. Aside from that, there are a few other features that can be integrated into the developed plug-ins. Such as: supporting simultaneous export and import capabilities when dealing with aggregate assemblies; and, supporting mass property calculations about user defined coordinate systems all the while allowing users to toggle between those coordinate systems during a DOME session.

More ambitious and difficult to implement features include: accounting for infidelity of neutral file to native CAD file in assembly interference checking; incorporating/integrating methods of neutral file healing to reduce the amount of error introduced into mass property calculations for a heterogeneous CAD assembly; incorporating/integrating methods of assembly analysis; incorporating/integrating methods of design for disassembly into the DOME VRML module; incorporating methods of VRML compression to reduce size of assembly files to workable sizes; and, automating reassembling of up-to-date neutral file components that contain variations in feature parameters into the heterogeneous CAD assembly.
Chapter 6: References


BadCAD, 2002: http://www.badcad.com/


CAD CAM-E, 2002: http://www.cadcam-e.com/

CAD Centric, 2002: http://www.cadcentric.com/main_ame.htm


CADKEY, 2002: http://www.cadkey.com/

Cadverter, 2002: http://www.cadverter.com/


Online Source: http://www.tech.purdue.edu/cg/courses/cgt226/cad_interop.pdf


SiteSculptor, 2001: http://vrml.ch/vrml97_awl/tools/sculptware/sitesculptor.htm


Spatial, 2002: http://www.spatial.com/


Transcendata, 2002: http://www.transcendata.com


VIM, 2001:


WebCAD, 1999: http://cybercut.berkeley.edu/html/design/webcad_user.htm

http://info.tuwien.ac.at/ecaade/proc/yeung1/paper/ecaade.htm