VIRTUAL ASSEMBLY MODELS IN DISTRIBUTED HETEROGENEOUS CAD ENVIRONMENTS

by

WILLIAM P. LITEPLO

B.S. Mechanical Engineering
Columbia University School of Engineering and Applied Science, New York, NY, 1998

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

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2000

Signature of Author.....
Department of Mechanical Engineering
May 5, 2000

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

Accepted by.....
Ain A. Sonin
Chairman, Department Committee on Graduate Students

VIRTUAL ASSEMBLY MODELS IN DISTRIBUTED HETEROGENEOUS CAD ENVIRONMENTS

by

WILLIAM P. LITEPLO

Submitted to the Department of Mechanical Engineering
on May 5, 2000 in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Mechanical Engineering

ABSTRACT

Enterprise-wide assembly modeling is a challenge for many engineering companies. This thesis presents a virtual assembly modeling paradigm that allows assembly of geometry defined in multiple CAD systems. In this environment, CAD modelers provide their geometric services over the web. Assembly engineers are enabled to visualize these three-dimensional services and incorporate desired objects into their CAD system without the ability to modify this geometry locally. However, through active links to remote CAD systems, they may also redefine the original geometry, and have their complete assembly recreated in their own modeling scope based on these design changes, mimicking parametric CAD models. The CAD systems used by the various participants need not be the same.

A research prototype system called DOME is used as the framework for this software-based assembly modeler. An example involving CAD modelers in different companies achieving a system-wide assembly model is discussed. The example illustrates the use of the assembly modeling system and the retention of proprietary information, such as 3D design history, for both parties.

Thesis Supervisor: David Wallace

Title: Esther and Harold Edgerton Associate Professor of Mechanical Engineering

ACKNOWLEDGEMENTS

I would like to thank many people for contributing to my research, writing, and general education: my family; Prof. David Wallace, my advisor, for his insight, guidance, good humour and icons; the rest of the Ford Five ("pdJeff" Lyons, Priscilla "Masta P" Wang, "What de" Juan Deniz and Chris "General Gao" Tan), for all their support and friendship; Nick Borland, Elaine Yang, Shaun Meredith, and Johnny Chang for DOME and administrative support; Stephen Smyth, for the foamschneider and SnagIt; all the rest of the folks in the CADlab; Peter Sferro, Bob Humphrey, Darrell Kleinke and Al Clark at Ford; Matt Wall and Ben Linder from DOME Solutions; cadlab26, for hanging in there; Quake3; CIPD; Donnie, Tommy, Francois and the Peanut Lady; Maggie and Maureen; Al Goncer at SDRC for tech support; Adora and Michelle, my wonderful dance partners; Manish, Jenny, Ronak, Rob, Soo, Tom, Jen, Sham and Kevin from Ashdown for all the Ashfun (Volpe); all the others in the BDT with whom I've competed (Nicki, Margaret, Sam, Yanfeng, Yedil, Steph, Michael, Mark, Karen, He-Rim, Jen, Katya), and not competed (Lauren, Tuan, Sofya and Boris, Tony, Kenia, Mohammed, Eric, Mark, Tilke, Dan, Sohrab, Genya and Mila and many others); Armin, Christine, Mark, Earl, Mike and Deirdre, the Nugents, and Warren and Elizabeth for coaching; Jenny, Nick, Ines and the other FreeBladers (it's freedom, baby); Plast, for everything and everyone; Yurchick, for Europe; Lisovi Chorty; Bill, Will and Tim (C&B); the Columbia Clefhangers; Randi, Andrea, Bonnie, Barbara, Ankur, Wayzen, Becca, Naomi, Amy, Rekha (Carman 7 and beyond); all the countless Ukes I am neglecting; and of course Boh.

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

Web-enabled technology has made it possible for the rapid and robust exchange of design information between the various participants in the product design cycle. Additionally, new technologies, such as DOME (Distributed Object-based Modeling Environment), for example, facilitate the integration of information between different engineering models. These technologies help to overcome many of the problems that arise from having designers located in geographically dispersed areas, and different sub-system models in a myriad of applications (Wallace, 2000). However, the issue of assembly modeling between different solid modelers, which requires tight interoperability at a data model level, is still problematic.

Most product development or engineering firms standardize on one single CAD (Computer Aided Design) package. This makes it easier to share models between different designers or departments. When collaboration is required across company lines, however, additional problems may arise. Suppliers may be using completely different modelers. Furthermore, actually transferring native CAD files can cause problems regarding proprietary information or trade secrets, often embedded in the geometry design history. Most CAD environments allow import and export of high-level neutral files, such as IGES or STEP. This overcomes the problem of divulging proprietary information, but, once imported into another CAD system, this geometry can no longer be modified parametrically. Thus, such an approach would require many time consuming iterations between organizations if a product is comprised of many subassemblies for different suppliers.

This research attempts to address the issues of CAD interoperability and retention of proprietary design history through a new concept enabled by a design service marketplace as is embodied by the DOME system. CAD designers will be able to build their parametric sub-system models as usual in the CAD system of choice. They will publish appropriate interfaces that define a set of services that let other participants

parametrically drive their CAD models over the Internet in a web browser-based DOME environment. Assembly designers will incorporate published high-level neutral file services from outside sources into their own CAD models. Then, when the assembly engineer makes a change to the published service interface of the remote subsystem, the actual subsystem CAD model will rebuild, and the assembly designer will automatically receive a new high-level description in their assembly model. Thus, the approach will make the high-level description appear as if it is editable parametrically (see Figure 1-1).

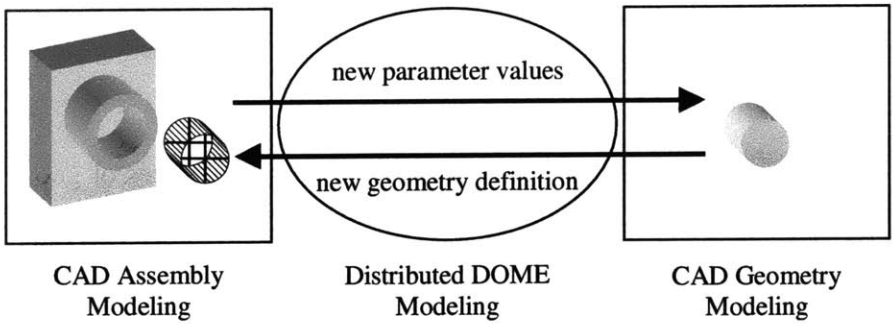

Figure 1-1 DOME assembly modeling paradigm

The assembly on the left incorporates the service of the peg (shown in wireframe) via neutral files. The geometry of the peg itself is defined and maintained in another CAD environment on the right. Through DOME, this CAD model can be parametrically driven over the web, and cause the peg to be automatically reassembled into the CAD model on the left. The result is parametric-like behavior without the transfer of proprietary design history.

This thesis first provides background information about issues in CAD interoperability and assembly modeling in Chapters 2 and 3, respectively. The DOME modeling system utilized by this research is presented in Chapter 4. Then, the interactions between DOME and CAD systems is presented in Chapter 5. This provides the framework necessary to perform virtual assembly modeling in DOME (see Chapter 6). An example of this assembly modeling system is given in Chapter 7. The outcome of this research is discussed in the final chapter.

2 BACKGROUND: CAD INTEROPERABILITY AND NEUTRAL FILES

2.1 The need for transferring geometry

There are many competing CAD/CAM/CAE (Computer Aided Design, Manufacturing, and Engineering) packages in use currently. Each tends to have different strengths and weaknesses depending on its application. CAD systems can be "quick-and-dirty" solid modelers or tie in a variety of analytical tools, such as interference checking, thermal, strength and/or other Finite Element Analyses (FEA), animations, etc. CAE tools are more commonly used for heavier computational engineering analyses of geometry, rather than for defining geometry. CAM packages are used specifically for manufacturing applications – determining tool paths, manufacturing time or cost, mold flow characteristics, and so on. As actual design of parts and systems is typically done in CAD, a need quickly arose to be able to transfer these data to CAM and CAE systems.

The existence of these varying software applications makes it difficult to collaborate within a design project when parties from different organizations are involved. One approach to address this problem is to mandate that everyone with whom one works uses the same software. This centralized data management directive must come from relatively strong upper layers of a dominant OEM. Automobile manufacturers are a good example of companies that typically use this strategy. As most companies are highly invested in their CAD/CAM software, this forced standardization is often costly, unpopular, and creates barriers to the consideration of different suppliers for sub-assemblies.

Another approach is to have direct translators from every CAD application to every other CAD application. Some packages come with such translators, but their supply is not directly in a designer's, nor even a company's, control. Furthermore, this solution scales exponentially with the number of software packages (see Figure 2-1). Finally, new

translators would have to be developed each time any CAD system released a new version, which quickly makes the problem unmanageable.

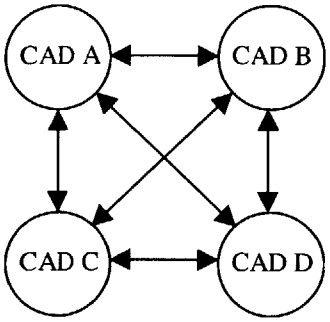

Figure 2-1 Exponential scaling of direct translators

Number required is on the order of n^2

The solution adopted by the CAD industry was to use neutral file formats. Once translators were written between every application and a neutral file type, CAD files from any supporting software package could be exported to this neutral format using a preprocessor, and imported into another system with a postprocessor (see Figure 2-2).

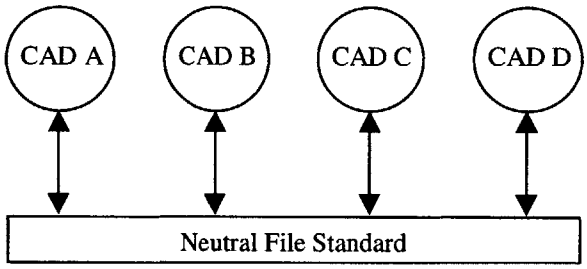

Figure 2-2 Linear scaling of translators using neutral files

Number required is n

However, these neutral file formats tend to only describe the topology of the solids and do not include the design history that allow the models to be edited parametrically. Thus, they are often referred to as dead geometry.

In certain applications, the use of neutral files is appropriate. For example, in a situation involving geometry designed in CAD transferred via neutral files to manufacturers using CAM tools, this approach suffices. Within the context of traditional assembly design, though, this method does not provide the needed modeling flexibility. The DOME assembly modeling paradigm presented in this thesis makes beneficial use of both the

ability of neutral files to capture geometry as well as their inability to transfer design history. Various neutral file formats are discussed in the remainder of this chapter.

2.2 IGES

Initial Graphics Exchange Specification (IGES) is a neutral file format originated by the National Institute of Standards and Technology (NIST) in 1979 and adopted by the American National Standards Institute (ANSI) to facilitate transfer of geometric data. The first published standard (IGES 1.0) was released in 1981 (Magoon, 1989). Improvements have been steadily throughout the years, and the most recent version is IGES 5.3, which was released in 1996. Version 6.0 is currently being developed. Iges files usually have a *.igs* or *.iges* extension. In practice, IGES has become the standard in CAD/CAM data exchange in the U.S. (Diehl, 1996).

Early versions of IGES and the CAD pre- and postprocessors that went along with them were often criticized as unreliable (Magoon, 1989). Since then, both the IGES specification itself and the translators have become more consistent and accurate. It is, however, up to each individual CAD vendor to maintain a current working version of its data processors. The use of outdated versions of the IGES standard by CAD systems may lead to versioning problems.

IGES information is based around different types of entities. The main classes of entities are geometric, annotation, and structural. Geometric entities range from points and lines to complex 3D surfaces. Other entities include dimensions, annotations, views and properties (IGES 5.x, 2000). Native CAD files do not necessarily store information as these same types of entities, and thus must map these entities to their own types. There are different levels of translation mappings. Some CAD objects might get mapped to multiple IGES entities, or vice versa, some objects may not be supported, which results in a null mapping, and others may in fact have a one-to-one correlation (Magoon, 1989).

IGES files are divided into five sections of data: Start; Global; Directory Entry; Parameter Data; and Terminate. There is also an optional Flag section to denote whether

it is in binary or compressed ASCII form. (See sample IGES file in Appendix A.1.) The Start section provides descriptive comments about the file. This information is ignored by the postprocessor and serves mainly as a human-readable description. The Global section describes the preprocessor and provides all the high-level parameters needed for the postprocessor to be able to properly deal with the file. The Directory Entry and Parameter Data sections contain all the information about the specific entities present in the file. The Terminate section is a single line at the end of the file that records how many lines long each section is.

There are both advantages and disadvantages to using IGES files. Some sources complain about their lack of robustness, but this has become much less of a problem over time. Coles, *et al.* claim that surfaces are not handled well (Coles, 1991), and yet others advocate that IGES is the most accepted format for transferring nurbs and other complex surfaces (Diehl, 1996). Thus, one can infer that the relative usefulness of IGES is not absolute. IGES versions change every few years, which makes it harder to keep the interoperability multidirectional. Furthermore, IGES supports the creation of variations and additions to IGES entities for use with a given system, known as "flavors". In a best-case scenario, processors will allow you to choose the flavor you wish to use. However, the use of flavors can often cause problems, as it may define entities not supported by the postprocessor of the importing system. For this reason, flavored IGES files are not truly neutral. Finally, as IGES is a US standard only, international businesses may find its use inappropriate.

2.3 STEP

The International Organization for Standardization (ISO) has developed its own neutral file format, standard number ISO-10303-21, more commonly known as STEP (Standard for the Exchange of Product Model Data). The standard itself is more than just a file format – it attempts to incorporate a more complete definition of the physical and functional characteristics of a product across its entire life cycle, including design, manufacture and support (UKCIC, 2000). STEP received full standard status in 1994. Since then, STEP has become used very heavily in the aerospace and defense sectors. It

has been recognized by most major CAD and CAM systems, and is inherently compatible with Product Data Management (PDM) (NRC, 1999).

STEP uses a specification language called EXPRESS to define all of its standards. It was designed to be both human readable and computer interpretable. A sample STEP file (which typically gets assigned a *.stp* or *.step* file extension) is provided in Appendix A.2. The STEP standard defines what are called "Application Protocols" (APs). Each AP determines what information a particular class of STEP-conformant tools can define. There are many different APs written for different fields of use. The most popular Application Protocol is AP 203, "configuration controlled three-dimensional designs of mechanical parts and assemblies" (NRC, 1999). This covers most of the geometric entities created in CAD. Although parametric relations are not defined within this AP, the actual data translation tends to be more robust and require less rework than IGES files. APs ensure true neutrality because a conforming STEP tool has to conform to the AP specification as well as the file format specification.

A main advantage of STEP over other formats is that it is an international standard. This has clear implications on the fast growing global economy. It also allows an older version of a compatible CAD system to open models created in later versions. This feature can be important when partnering departments or organizations use differing CAD versions.

STEP aims to have applications in multiple fields, and is capable of storing product data regarding design intent or electronics and piping, to name a few. IGES 5.3, in contrast, has added some basic electrical modeling and piping capability which has not been fully tested (NRC, 1999). STEP was made to be used as a neutral product database with long-term archiving capabilities that acts as a single storage standard for all product development data (NRC, 1999). This makes it well-suited for system integration (Duan, 1996). There are several examples of this use in industry and research.

A large sector of companies in the aerospace industry (including Lockheed Martin Tactical Aircraft Systems, Northrup Grumman Commercial Aircraft Division, and a number of suppliers) are participating in a pilot project entitled STEP Web Integrated Supplier Exchange (STEPwise) involving the sharing of STEP data over the web (NRC, 1999). The main objective of this project is to integrate PDM systems using an agreed upon standard. An encrypted ftp method is used to transfer the data that are loaded into a PDM system.

Peng, *et al.* at National Tsing Hua University in Taiwan have developed a STEP-based object-oriented database management system (Peng, 1998). The system is based on the belief that STEP accurately represents product data from many points of view, including design, manufacturing, finance, and others. It allows multiple users to collaborate on a design through web browsers. Although the data are processed by different components for different uses, they ultimately get stored in an integrated centralized database.

2.4 VRML

Virtual Reality Modeling Language (VRML) is used to publish and view 3D objects over the web. The first draft of the VRML 1.0 specification was written in late 1994 by SGI (Silicon Graphics, Inc.) and the final VRML 2.0 specification was released in 1996 (Ando, 1998). "VRML 2.0 was recognized as an international standard (ISO/IEC-14772-1:1997) by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) in December, 1997" (Crispen, 1998). VRML is a "scene description language that describes the geometry and behavior of a 3D scene or 'world'" (Crispen, 1998). Within a VRML world, one can alter one's viewpoint of the contained objects in a variety of fashions. VRML files can be viewed with a standard Internet browser using a plugin, or with standalone programs developed for this purpose. A common VRML plugin is Cosmo Player, owned by Computer Associates International, Inc. (CAI, 2000) (see Figure 2-3). VRML files have a *.wrl* extension.

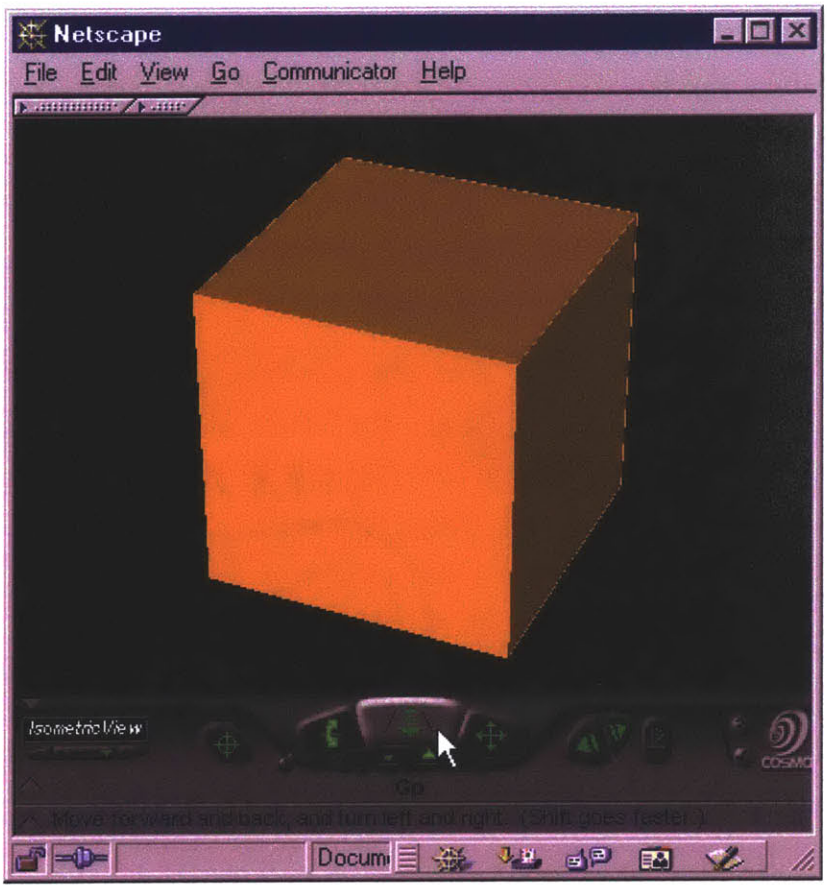

Figure 2-3 Simple VRML shown in Cosmo Player

VRML 1.0 worlds are static. They can include normal geometric objects, textures, lights and cameras, but no action. Users can manipulate the view of these worlds, but the objects themselves cannot be altered from a browser. In contrast, VRML 2.0 (also called VRML97) supports animation, switches, sensors, sounds, scripts and events (Crispen, 1998). This makes VRML worlds much more interactive. One of the points driving the design of VRML 2.0 was to make innovation and standardization co-accessible (Ando, 1998).

The VRML standard is not a major player in the CAD/CAM interoperability arena, but it has become a standard for web visualization of 3D geometry, whether originated from CAD or written directly in VRML (see sample file in Appendix A.3). The majority of CAD programs do have plugins or converters for VRML. Overwhelmingly, though, VRML models are built by people in the computer graphics industry.

2.5 Other file formats

AutoCAD's Data eXchange Format (DXF) (AutoDesk Inc.) is another common file type based on a proprietary format rather than a public one. AutoCAD has its own native file structure, but, since in 1982, its DXF format has been published for all to use. DXF files are composed of pairs of group codes and associated values. The group codes specify the type of item that follows. The associated values can be strings or numbers. Using these group code and value pairs, DXF files are organized into sections, including the HEADER, CLASSES, TABLES, BLOCKS, ENTITIES and OBJECTS sections. Each section is composed of records, which are composed of a group code and a data item. Each group code and value is on its own line in the file.

There are, however, a number of problems with the DXF format. First, it can define only geometrical entities, often favoring simplicity over richness of data (NRC, 1999). There are many problems in translation (Rudolph, 1995; Weber, 1996; Zapor, 1998). Furthermore, the format changes with each new release of AutoCAD, so staying up-to-date becomes difficult. Finally, AutoCAD itself is used much more for two-dimensional drafting and less so for 3D modeling, and the DXF format appropriately lacks robustness in its definition of three-dimensional entities.

Standard Triangulated Language (STL) files, often referred to as stereolithography files, are "the de facto industry standard in rapid prototyping" (Bohn, 1995). These files are made up of an unordered set of triangles in three-dimensional space. Each triangular facet consists of a triad of vertices and a direction. The entire set defines a surface that contains a volume, or regularized set. The STL standard is ideal for use with rapid prototyping (RP) devices because it only defines the boundary between positive and negative space, and RP is usually performed by simply adding homogeneous solid material in appropriate locations.

STL files are known to frequently fail to properly enclose a volume, especially for more complex surfaces. This problem is dependent upon the method in which the geometry

was originally built, typically in CAD. Usually this leads to tedious manual editing (Bohn, 1995). Another disadvantage with STL files is the loss of smoothness. Curved surfaces become noticeably faceted, depending on the level of triangularization. Even small prototypes still may require additional surface operations. Although most CAD packages will export STL files, many do not import them. All curvature is lost and the amount of tiny surfaces puts unnecessary strain on the processor. Curved geometry is much better represented in the CAD system's native form. This makes the interoperability quite one-sided from CAD to CAM, and even then, the use is better suited for prototyping than for production purposes.

2.6 Assessment of neutral files

For neutral files to be usable, CAD systems must conform to their standards. This can be broken down into three types of conformance. First, output files must have an acceptable structure. Second, a preprocessor must conform in that it creates conforming neutral files. Third, a postprocessor must be able to read conforming files without aborting (IGES 5.x, 2000). Rarely does a one-to-one mapping exist between native objects and their corresponding neutral entity types, so there is almost always some degradation in data when models undergo translation. However, if one accepts a particular CAD system, its inherent limitations must also be accepted. Although neutral files are not perfect, their relatively high level of functionality, ease of use and acceptance by industry make them desirable as a means of attaining CAD interoperability.

Among the aforementioned file types, STEP and IGES are the most robust and accepted for transferring geometry between CAD systems. DXF files have limited application and are not recognized as an official standard. VRML is useful for visualization, but is not commonly used for importing geometry back into CAD. Similarly, STL files are appropriate for providing data to RP devices, but do convert well going into CAD systems. Thus, use of the STEP and IGES standards has been incorporated into the assembly modeling research presented in this thesis.

3 ASSEMBLY MODELING

3.1 Virtual Assembly

Technology to bring Design and Manufacturing closer together is continually developing. These efforts stem from the desire to reduce product development cycle times. Digital, or virtual, prototypes of various types are being used in industry to reduce the need for physical testing. The field of Virtual Assembly (VA) studies the technologies that enable virtual modeling of three-dimensional geometry as related to mechanical assembly. VA exists within the scope of a larger field, Virtual Manufacturing (VM), which refers to the collection of programs that "have been developed to create an on-line environment to analyze the producibility of the product design" (Lee, 1995).

VA can be defined as "'The use of computer tools to make or assist with assembly-related engineering decisions through analysis, predictive models, visualization, and presentation of data without physical realization of the product or supporting processes'" (Connacher, 1995). There are many possible applications for VA. Some are still emerging technologies, but the concept as a whole is regarded as feasible and a logical next step. The goals of such efforts are to: "enable manufacturing engineers to evaluate, determine and select more optimal component sequencing, generate assembly/disassembly process plans, make better decisions on assembly method (i.e., automated or hand assembly), and visualize the results," (Connacher, 1995). Another goal is the ability to share and exchange information between new programs and currently used CAD/CAM/CAE applications.

As these technologies continue to grow, they will become more accepted and used in practical situations. How readily industry will embrace these applications depends on a number of criteria. First, the technology must enable engineers to assess their respective issues with clarity. Second, it must assist its users in making the needed decisions. It also must be applicable to real situations. Next, the technology must be simple to use and

easy to access. Finally, the system must be capable of interfacing with existing engineering systems (Connacher, 1995).

There are numerous roadblocks to successful VA. One of the greatest barriers to a successful assembly planning software is the fact that "assembly is dependent on a great deal of expert knowledge which has proven very difficult to formalize" (Dewar, 1997, referenced in work by Jayaram, 1999b). Another significant general problem is "generating information, both quantitative and qualitative, about the potential behavior of the design during assembly" (Lee, 1995).

This chapter explores the various industrial and academic approaches to attempting Virtual Assembly.

3.2 Assembly Modeling Research

A great deal of research on assembly modeling has been led by Dr. Daniel Whitney of MIT and The Charles Stark Draper Laboratory (Nevins, 1989, Whitney, 1996, Whitney, 1994). Whitney has developed theories and terminology regarding the lowest level details about the physical assembly and how they map to larger assembly issues. In particular, he distinguishes between *mates*, which are assembly connections that pass constraint, and *contacts*, between parts that touch but do not pass constraint. By following the links between all the mates, one can create a Datum Flow Chain, a high-level capture of knowledge depicting the causality flow of constraint. This graph enables one to trace problems that arise upon assembly back to their origin.

Limits of these theories include the assumptions that all participating bodies are rigid and incompliant, and the inability to model interactions that change over time (Lee, 1995).

A few attempts have been made to provide this functionality in software (Baldwin, 1991, Mantripragada, 1998). The outcome, however, has always been a system that takes too long to set up because geometry definition has to be manually inputted in the form of 2D drawings, rather than being tied to a CAD system.

More preferable would be for CAD systems to incorporate some of the concepts into their modeling paradigm. There are a number of reasons why this proves more difficult than one might imagine. Typical CAD packages model assemblies the same way they model parts (Whitney, 1998). To them, assemblies are little more than visualizations of multiple parts. There is little emphasis on how these assemblies actually come together on the assembly line. Thus, assembly drawings are not true assembly models, as they do not capture the information needed to assemble the parts together. Instead, they provide a visualization as to how the parts should look in relation to each other once fully assembled. Similarly, constraints, rather than being based on assembly sequence, are tied to geometric entities. Many configurations that in reality are over-constrained are analyzed by the CAD system as fully constrained (Mantripragada, 1998).

Although CAD packages are always evolving, certain paradigm changes need to be made before they are equipped to handle higher-level assembly analyses. This is one of the limitations of CAD, however their importance as a widespread, industry-accepted tool for defining geometry cannot be undermined. Much additional work has therefore been done to try to incorporate the positive functionality of CAD with additional services of other systems. This integration undoubtedly utilizes some form of standardization. Example of this work are discussed in the remaining sections of this chapter.

3.3 Types of standardization

3.3.1 Neutral files

Chapter 2 discussed the types, uses and benefits of conformation around standard neutral file formats. This method of standardization is common in all aspects of industry.

Sugimura *et al.* propose an assembly modeling system used to generate assembly sequences based on the STEP standard (Sugimura, 1996). Solid modeling is done on a Parasolid modeling kernel (by Unigraphics Solutions, also used in Unigraphics, SolidWorks, Pro/Engineer by Parametric Technology Corporation and others), in which

the user manually defines features and geometric relations. Assembly preference rules are stored in a database which allow feasible assembly sequences to be derived from the feature-based model.

3.3.2 Centralization of data

U.S. automobile manufacturers face these same problems and issues of enterprise-wide assembly mentioned in the introduction to this thesis (Wyman, 2000). Their efforts are usually large and distributed, "spread out over several continents and involving thousands of engineers and technicians" (Kaplan, 1997). Typically, they would hold group meetings in large rooms with many printed drawings and layouts spread out throughout the room. As can be imagined, this system provides only a static representation of the vehicle. Design alternatives were near impossible to picture, and there was no guarantee that all parties involved were using the most up to date information.

Within the past few years, these companies have attempted new methods to achieve fully digital, enterprise-wide, whole-vehicle assembly integration (Wyman, 2000). These methods are maintained by the CAD and Engineering divisions of their various Vehicle Programs (VPs). All participants, including suppliers, would typically have to use the same CAD and PDM systems to work toward a specific design version. They "populate" the PDM system by checking in CAD data corresponding to all different configurations, or derivatives, of the vehicle (e.g. 4x4 or 4x2, differing power trains, etc.). These data can be processed into another format by an external program for use by their assembly modeler.

Real-time visualization and analysis of whole vehicle assemblies is then possible. One can swap in and out parts, perform collision detection, and motion and clearance analysis. Especially beneficial is the ability to visualize routing problems when changes are made. With this type of setup, not only can this dynamic system be interacted with, but a lone user can do it from his or her own workstation with ease. This avoids the need for specially-built rooms set up solely for this kind of work. This system can be operated by an individual, small teams, or large groups. At critical milestones (usually 3-4 per VP),

the design is frozen, allowing version control. While the majority of this functionality is in fact available in most CAD packages, no CAD software can handle a full vehicle in one model on a typical computer.

What this type of system cannot do is dynamic manipulation of continuous variables. All the configurations of each of the components must be predefined, thus making the set of possibilities discrete. Modification of values and re-generation of geometry can take about one day to complete, making the engineers' job of keeping the PDM data up to date tougher than they would prefer. Supplier design integrity is also unclear, as suppliers must provide native CAD models to the OEM.

3.3.3 Feature definition

De Martino *et al.* describe a distributed, object-oriented, feature-based system in which both top-down and bottom-up modeling can take place (De Martino, 1998). The purpose of the research is to integrate engineering design (CAD) and application (CAE) tools by enriching the exchanged data with parametric, functional and technological information. The underlying architecture builds its object base upon CORBA (Common Object Reference Broker Architecture) to allow distributed data flow.

The system relies upon feature libraries to offer design-by-feature (a top-down process) and feature recognition (a bottom-up process). These features are linked directly to its ACIS solid modeling kernel (by Spatial Technology Inc., also used by AutoCAD, CADKEY from Baystate Technologies, Inc. and others). The features can have multiple representations depending on the eventual application.

The system architecture incorporates a design client that controls the geometry and may define features linked to that geometry. The core data flow is controlled by the "Intermediate Modeler" (IM) server. A downstream application tool client receives feature information from the IM. As design changes are made, the IM interprets their meaning in terms of feature sets and automatically provides the results to the application tool. Each client can view the feature set in a manner meaningful to their task.

The outcome is a potentially useful resource for the emerging field of feature-based modeling. However, there are numerous limitations to this system. Primarily, the feature definition libraries are not a standardized set. Although this was the goal of the IM system, the features used by this particular research work do not have any backing from industry, and industry has not defined any similar standards. Second, the feature libraries are tied to a specific solid modeling kernel. Integration with CAD/CAM/CAE systems using other kernels was not attempted. Also, the data flow is primary unidirectional, from design to application. Finally, although there may be multiple application contexts, the system can only analyze a single design at a time, so that there is little room for dynamic system growth.

3.4 VA via VR

3.4.1 Integration of VR and analysis

Typically, an assembly engineer has the job of integrating the various parts of a system and defining a viable assembly sequence. To obtain a full conceptualization of the critical assembly issues, it is usually necessary to attempt the assembly with physical prototypes. Early in the design cycle, this can be very costly from a time and money standpoint. Emerging concepts of simultaneous engineering attempt to enable this process by offering a Virtual Reality (VR) environment as a substitute for physical prototypes to reduce cycle times for design. Visualization of the model in VR enables designers to see incompleteness in the model, and thus make the needed changes before having to produce anything (Templeman, 1996).

Steffan, *et al.* describe some of the general requirements for VR-assisted assembly planning (Steffan, 1998). First, both the CAD and VR systems should have pre- and postprocessors capable of dealing with standardized file formats, such as IGES and VRML. The geometry, dimensions and manufacturing information is defined in the CAD system, while degrees of freedom of parts and tools are defined in the VR system.

Many researchers have developed VR-based assembly systems (Raghavan, 1999; Sharma, 1997; Lee, 1995). One typical example will be discussed in detail in the following section.

3.4.2 Virtual Assembly Design Environment

At Washington State University, researchers are developing the concept of "Design by Manufacture". This type of environment provides virtual access to manufacturing methods and tools and, through manipulation of those, designers can better design products up front. One such environment being developed is called VADE, Virtual Assembly Design Environment. The project began in 1995 under NIST sponsorship (Connacher, 1995). The main goal of the performed tests was to assess the accuracy of the assembly time evaluation as compared to actual assembly times (Jayaram, 1999b).

VADE is an example of a Virtual Assembly system that uses Virtual Reality. The VR environment is created by a headset and glove that the user must wear, and the software that corresponds with it. On the more conventional side, VADE interfaces with a CAD system called Pro/Engineer. The VR system takes the Pro/Engineer models and displays them as polygonal models. From within the VR environment, the user can define an assembly sequence by moving parts. From this, swept surfaces are generated and "sent back to the CAD system as a set of bi-parametric surface patches" (Connacher, 1995).

The VADE process starts with building models in CAD. The geometry, attributes and assembly information of the CAD model are extracted by the VADE preprocessor. Then VADE converts this geometry into STL files. This is then imported into the VR environment (Connacher, 1995), where parts are located where they would be in a real assembly plant. The system maintains an active link to the CAD system and uses its capabilities whenever it can (Jayaram, 1999b). Within the CAD system, users can tag certain parameters. VADE supports making design changes to these model variables from within the virtual environment. These changes are sent back to the CAD software, which rebuilds the geometry and resends it back to VADE and the VR environment. This is all done in "almost real-time". For simple parts, bi-directional translation between

VADE and CAD is essentially instantaneous; for very complex parts, a full interaction cycle takes about one minute (Jayaram, 1999a). All of the capabilities of VADE can be run synchronously or individually (Jayaram, 1999b).

One of the main advantages to this system is that assembly can be performed intuitively in the virtual environment (Jayaram, 1999a). A disadvantage is that it is not scaleable in relation to hardware capabilities, such that coarser model approximations and displays could be used on systems with less processing power. The state-of-the-art VR hardware still does not allow this process to be accessible to more than the elite echelon of users (Connacher, 1995). For reference, testing was done on an SGI Onyx with six processors! Another major limitation of VADE is that working in a VR environment has a number of human factors issues that can result in high user fatigue, so large assemblies are best split up before evaluation. Test results showed that VADE yielded the best results for complex assembly motions that are performed by humans, and poorer results for automated assemblies with simple operations (Jayaram, 1999a).

3.5 Analysis of assembly modeling techniques

The systems described in this chapter all have common goals of bringing assembly modeling into a digital, collaborative environment in order to reduce design and analysis time and produce more robust products, but each implements them differently. Everyone seems to agree that some level of standardization is required to do this well. Based on the common acceptance of neutral files by CAD companies and users, conformity over the agreement to use neutral files presents itself as the most valid method of standardization.

For this reason, the method adopted by automobile companies of forced conformance to a native CAD system is not acceptable in the framework of this research, in which users should be able to continue modeling in their preferred format, and the architecture of the assembly modeling system handles the interoperability. Furthermore, this method discourages inter-organizational collaboration, because proprietary CAD models are shared.

The other systems discussed in this chapter fail to provide a system that is applicable to real problems, easy to operate, available to many users, and interfaces with existing geometry modelers. Some of them achieve interoperability within local scopes of visualization, but do not take that any further. Although the Virtual Reality systems may become more accessible in the future, its use by the common engineer is not supported by society's current infrastructure.

The assembly modeling system presented in this thesis meets all of these requirements. Furthermore, it addresses the need of retention of proprietary designs by their owners. The architecture of the DOME system on which it runs is explained in the following chapter.

4 DOME OVERVIEW

DOME stands for Distributed Object-based Modeling Environment. The DOME concept was originally developed in the MIT CADlab (Wallace, 2000). A commercial version of the software is now being developed by Dome Solutions, Incorporated. This software is used as an infrastructure to prototype the assembly paradigm explored in this thesis. The basic concepts underlying DOME are presented in this chapter to build up a conceptual framework for this assembly modeling research.

4.1 DOME Architecture

DOME is a distributed application. The core functionality runs on DOME servers, which are much like web servers, except that they provide service-objects that can be linked to underlying application models (Wallace, 2000). These servers control all of the data flow, and interface with third-party applications using specially written plugins. Access to the servers is obtained by starting a DOME client, and logging into a server. Servers are capable of having multiple clients, each with the opportunity to view the same information simultaneously. DOME servers can additionally communicate with other DOME servers, and exchange information remotely across a network.

The kernel of the DOME server is written in C++. Between servers, CORBA is used to pass information. Various programming standards such as COM, OLE, C++ and CORBA are utilized when connecting from these servers to third-party applications. The method used depends completely on the software's API. Additionally, there is also a layer of Java on the server side, which communicates with C++ using JNI, Java Native Interface. DOME clients run a Java applet that use Remote Method Invocation (RMI) to talk with the Java server. Clients can be run from any standard web browser, such as Netscape or Internet Explorer.

4.2 "Business as Usual"

DOME is a minimally invasive tool because it allows users to continue to use the software programs with which they are familiar (Wallace, 2000). For example, a CAD designer who models in I-deas does not have to convert his files into another native format. Users can continue the work as they have in the past. The actual models or simulations used before can provide the same analytical capabilities once incorporated into a DOME model. Examples of software plugins written for DOME include Microsoft Excel, SDRC I-deas, ODBC databases, SolidWorks (SolidWorks Corporation), MSC Marc, The Math Works Matlab and Ansys (Ansys, Inc.).

4.3 Publishing Model Services

To make one's models or simulations usable by DOME, one must "publish" the services one wants to be available, or define the set of model objects desired to be accessible from DOME (Wallace, 2000). There are a group of basic service types within DOME. Included in this group are real number variables, strings, logical boolean values, matrices, containers, catalogs and relations. All of these services are DOME objects, or modules, available for use by wrapper objects to various third-party software applications.

The procedure for publishing the interface to a model depends on the third-party application. Some programs store the information needed to publish services within the simulation file itself; others store those data in an external text file. Methods of publishing for CAD models will be discussed in Chapter 5.

Once the interface for a model has been defined and published, it is very easy to connect to that model from DOME, on the web. One must first log into a DOME server from a DOME Client and open up a model (see Figure 4-1). DOME models can be built on the fly or saved in an existing configuration and reloaded at another time. The wrapper module can be added to an open model through the proper menu selection.

Figure 4-1 DOME client running in Netscape and an empty DOME model

Once this type of object has been loaded in DOME, a view of all the interface objects published earlier is presented. The various services defined in the interface are created as DOME modules. For example, a real number parameter defined in the publisher interface would appear as an Real module object in the DOME model. Then, from the web, one can change deterministic values by simply typing in numbers. This action drives the underlying simulation with the new value and can provide new simulation results as outputs from the model. Since this functionality exists on the Internet, anyone with access permissions to log into that DOME server can do so from anywhere in the world, see the same interface, and likewise modify its inputs.

4.4 Integration

Thus far, point-to-point web access to remote applications has been discussed. For DOME to truly provide modeling capabilities, one needs also to create links between different objects. This can be done using relations, which are another type of DOME object (Senin, 2000). Relations behave like links in a directed graph. They have driving and driven parameters. One can define just about any type of relationship between any of the available modules. Typically, simple equality relations are established between multiple representations of a unitary concept. For example, if more than one DOME model or part of model contain an object representing a particular length parameter, these can be equated to each other so that their values are consistent. These relations can be as simple as the aforementioned equality, or as complex as any C++ program.

In order for a service to be used in another model, one has to copy and paste an *alias*, or reference, into the scope of the new model. Essentially, that alias module serves as a representation of the original. Changing the value of the alias in reality changes the value of the original. This provides local DOME access to remote services. From this local copy, relations can be made to form consistencies between parameters across different servers. For numerical values, unit conversions are handled as long as the fundamental quanta match (e.g. length to length).

Through this process, the overall structure of the system will emerge, as users define relationships that make sense in their respective modeling scopes. There does not have to be any centralized manager sitting on top of all the models that manages relationships, so that the resulting system can be dynamic and emergent, rather than static and pre-defined.

4.5 Proprietary information

By keeping all the relevant engineering models local, in control of the model owners, the DOME architecture prevents unnecessary transfer of proprietary information (Senin, 2000). It is, in fact, these models which are often one of the core competencies of many companies. In the case of CAD models, this includes the design history of how the models were built, not just the final geometry. Sharing these models in their entirety with members outside their company would be seen as a violation of intellectual property rights. Since DOME only reveals individual parameters, and not the underlying models that control them, there is no need to be concerned about divulging one's models. This is one of the key concepts driving the assembly modeling paradigm presented in this thesis.

4.6 Service marketplace

Enabled by this structure, it is envisioned that DOME users can provide their services onto the web in a marketplace setting (Wallace, 2000). Other users can search for the services in which they are interested via an agent-based system that is being developed for DOME, or connect to known servers based on established company relationships. When someone finds a service they would like to incorporate into their model, they can

subscribe to it by copying a reference to it into their own model, as described in Section 4.4.

4.7 Addition of service types

With the evolution of the DOME architecture, a method for performing distributed assembly modeling became possible. The author's work explores the methods and rationales behind the realization of this goal. Chapter 5 describes the use of CAD tools and their geometry-based services in the context of DOME modeling. To fully utilize the services available from CAD models, however, fundamentally new DOME service types first had to be defined. Two plugins – one providing the service of a VRML file and the other a more generic neutral file – were developed to facilitate this assembly modeling research.

4.7.1 VRML plugin

VRML 2.0 files are viewable in DOME through a VRML service object. This module contains a path to the file, which can be any VRML file accessible by the server machine, or any valid web URL. Once loaded, the VRML objects are appropriately displayed. The plugin incorporates open source code from the Java 3D and VRML Working Group that enables this visualization (Sun Microsystems, 1999). As with other typical VRML worlds, the user can now rotate the object, zoom in and out, manipulate the viewpoint, change the lighting and otherwise get a rather full visualization of the geometry. At any time, the user can reset the view, reload the file, or load up a different VRML.

A DOME client running on computer other than the server, however, would not be able to load up the VRML unless the specified path was to a publicly accessible URL. For this reason, the plugin allows each client to specify a local path to which they want the VRML file to be transferred. The module regenerates the VRML file, originally on the server, on the client's local machine. This can be different for each client.

An additional parameter in the VRML module is a boolean value denoting the user's wish to keep the file up to date when changes occur to the defining geometry. The use of this variable will be discussed further in later chapters.

Every DOME module has a Graphical User Interface (GUI) that can be opened from the DOME modeling environment. In the GUI of this module, there is a text field for the path to the file on the server machine, another for the file path on the client machine, a check box for setting the *Keep Current* property, and two buttons (see Figure 4-2).

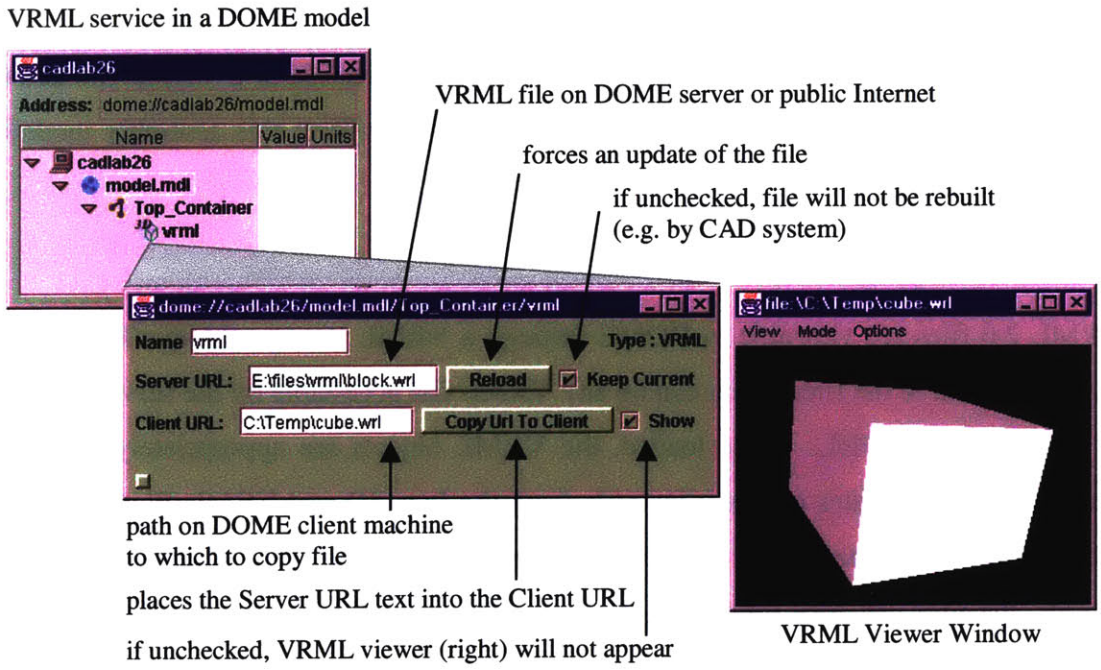

Figure 4-2 VRML module GUI

To view the file, one must specify a path on the client's machine. The *Copy Url to Client* button may be useful if the server path is an Internet address. Once this path is entered, and the referenced file is a syntactically valid VRML file, the file contents are ported to the specified client file, and the defined 3D geometry appears in a separate viewer window. A *Show* check box enables or disables the VRML from being brought to the foreground every time a change is made to the file.

4.7.2 Neutral file plugin

The Neutral File (NF) plugin provides access to neutral files. Like the VRML, the NF module stores a path to the neutral file accessible to the DOME server and an on/off

switch for keeping the file up to date, and additionally contains a file type characteristic. In particular, the NF module is compatible with all IGES versions, STEP AP203 and STEP AP214 (Core Data for Automotive Mechanical Design Processes) files, but is generic enough to handle any neutral file type (see Section 2.6). Upon specification of a valid file, the contents of the file are parsed to determine the file format and specification version, and the file type is stored accordingly (see Figure 4-3).

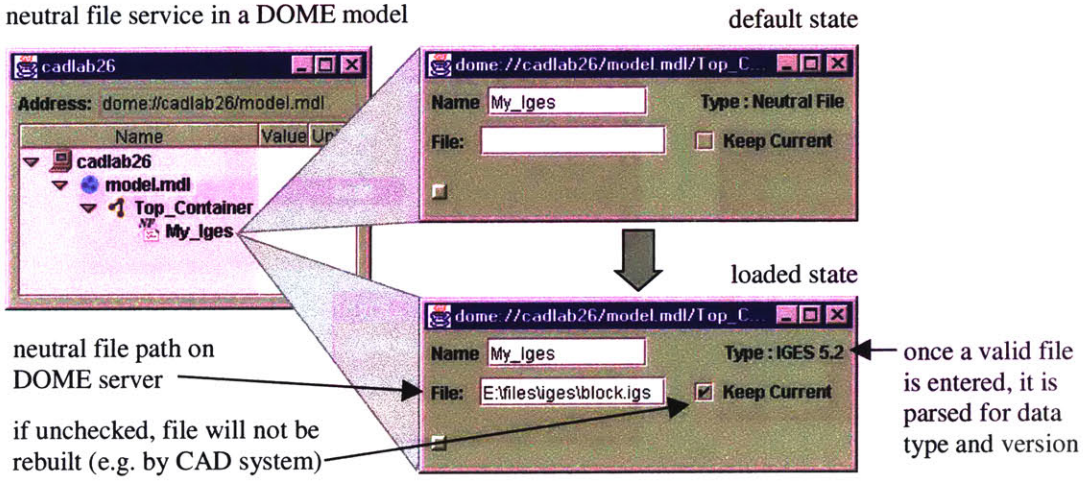

Figure 4-3 NF module GUI

Unlike the VRML module, who's primary function is visualization of remote geometry, the NF module's main functionality is providing access to that geometry. This functionality is enabled by aliasing the NF module. One can copy a NF module from a remote DOME model and paste it into one's own local scope. This module then carries with it all the contents of the neutral file, without actually displaying them. The original filename does not in fact get propagated to remote aliases, because the path would no longer be valid. What can be done is the creation of a relation equating the alias to a local NF module. Then, when the local NF module changes, or loads up a new file, the contents will automatically be transferred to the file specified by the original NF object on the remote computer (see Figure 4-4).

This ability to transfer neutral files is fundamental to the proposed DOME assembly modeling system, which will be discussed further in Chapter 6.

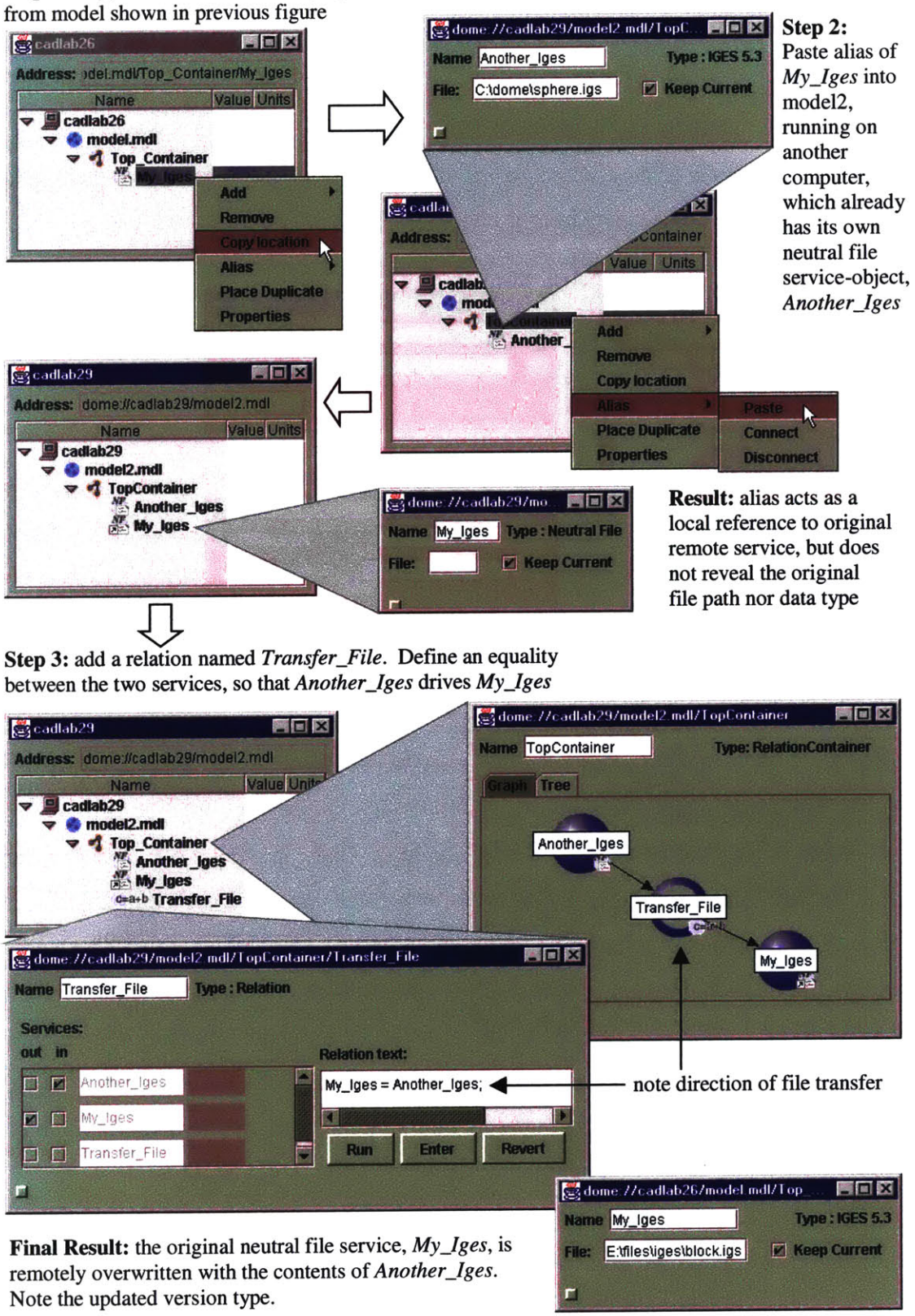

Figure 4-4 Neutral file transfer using a relation

5 CAD PLUGINS

CAD is by far the primary method of defining 3D geometry used by engineers. This makes the interface with it an extremely vital piece to include within the DOME system. The CAD plugins for DOME discussed in this chapter were developed in part for the foundation of the assembly modeling techniques explored in this thesis as well as for a pilot research application conducted with Ford Motor Company.

5.1 IDEAS

5.1.1 Software and API

I-deas is the CAD system produced by Structural Dynamics Research Corporation, SDRC. The latest version of the software used for the purposes of this research is Master Series 7, although DOME is compatible with versions 6 and 6A as well. I-deas runs both on NT and Unix workstations.

The application program interface (API) is based on Orbix, a commercially-available implementation of CORBA (SDRC, 1998). Through this interface, one can access the various objects used and created by I-deas, as well as all the methods that are supported for those objects. Thus, one can perform virtually any function that is normally done through the software's graphical interface by connecting to an I-deas server through C++. This is what the DOME I-deas plugin does. Since CORBA is a distributed object broker, one can run DOME on one computer and connect to a session of I-deas on another computer, even on a different platform.

5.1.2 Publishing dimensions

The most important numerical quantities stored in CAD models are dimensions, for they truly define the geometry. Naturally, one would want to be able to modify these dimensions remotely from DOME. To make these dimensions available, it is necessary for the interface desired to be seen from DOME to be published by the owner of the CAD model. In other words, the CAD designer must specify which dimensions are to be

accessible. To make this process easy, it can be done directly from within the I-deas environment.

Publishing I-deas dimensions is as simple as adding a specific extension to the names of the given dimensions. These could be linear, angular, radial, diametrical or percentage dimensions. If a CAD modeler wants to let DOME users change certain dimensions, he would add the suffix *_DOMEin* to their names. These would normally be the dimensions that drive the parametric model, and may or may not affect other parameters. If, in fact, there were driven dimensions in the model that were desired to be seen in DOME, one could just as simply add the suffix *_DOMEout* to those dimensions. Figure 5-1 shows dimensions in a simple I-deas model being published.

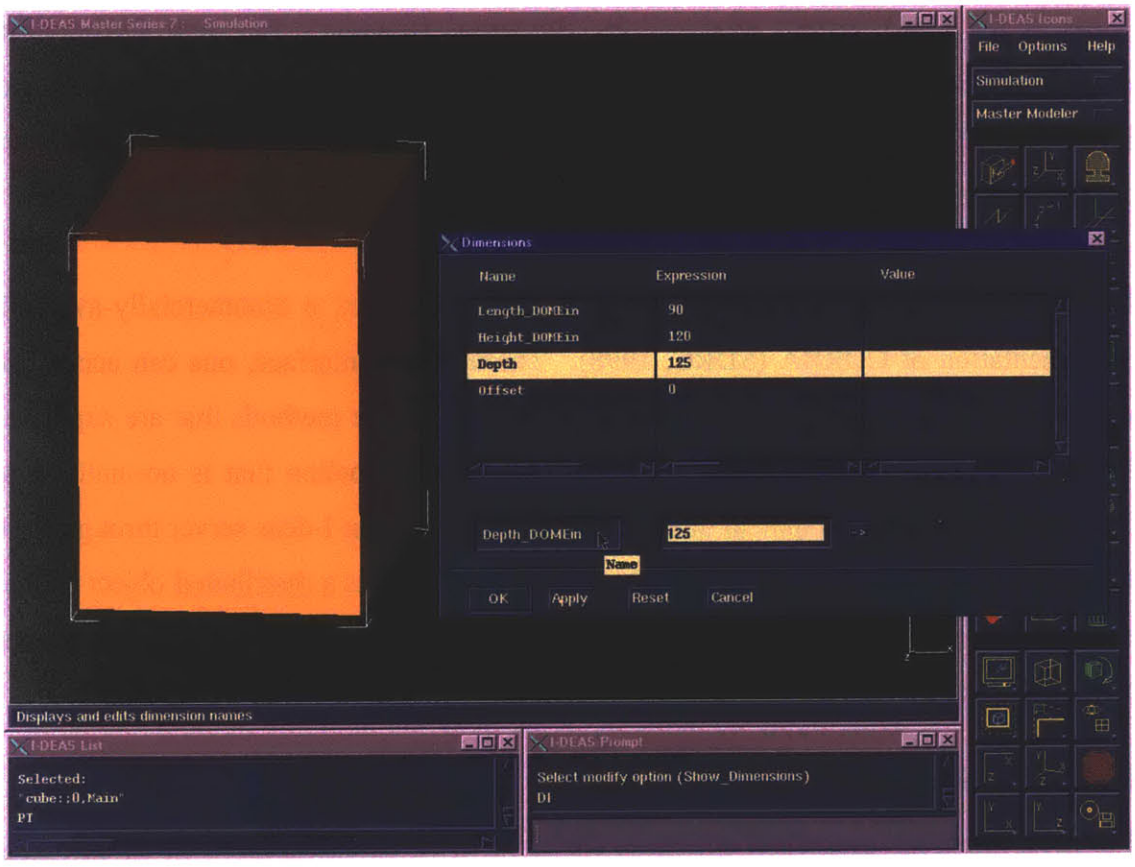

Figure 5-1 Publishing dimensions of a simple I-deas model

By specifying which model parameters are inputs to I-deas, and which ones are outputs from I-deas, the CAD designer makes the overall "black-box" architecture of the model

available, without providing access to the underlying equations and parameterizations that define the model's behavior.

5.1.3 Publishing parts and assemblies

In addition to simple dimensions, one might want to provide other information about a model that is readily available from within CAD. This can consist of various geometric and mass properties of parts, as well as of assemblies. I-deas calculates and provides the following properties: volume, surface area, mass, center of gravity, moment of inertia, principal axes and principal values. These are things that one would likely be interested in tracking upon making changes to input dimensions. Naturally, these would be considered outputs of the simulation, as changing them directly would have no geometric significance. Additionally, a user may decide to define these properties calculated about a user-defined coordinate system that may not be the default.

To make these properties available to DOME, one first publishes the owning part or assembly by adding a *_DOME* suffix to the name, similar to the method for publishing dimensions (see Figure 5-2).

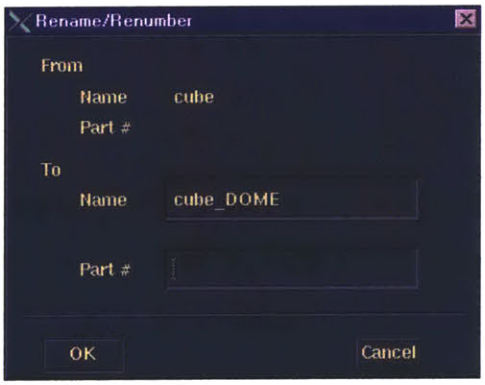

Figure 5-2 Publishing an I-deas part

In order to discriminate which properties to provide, however, there has to be an additional interface definition. I-deas does not provide any means of "tagging" these parameters as was done with dimensions. It does, however, provide a text area in which one can write a general description of the given part or assembly. It is in this text region that one publishes the various services available from a part or assembly (see Figure 5-3).

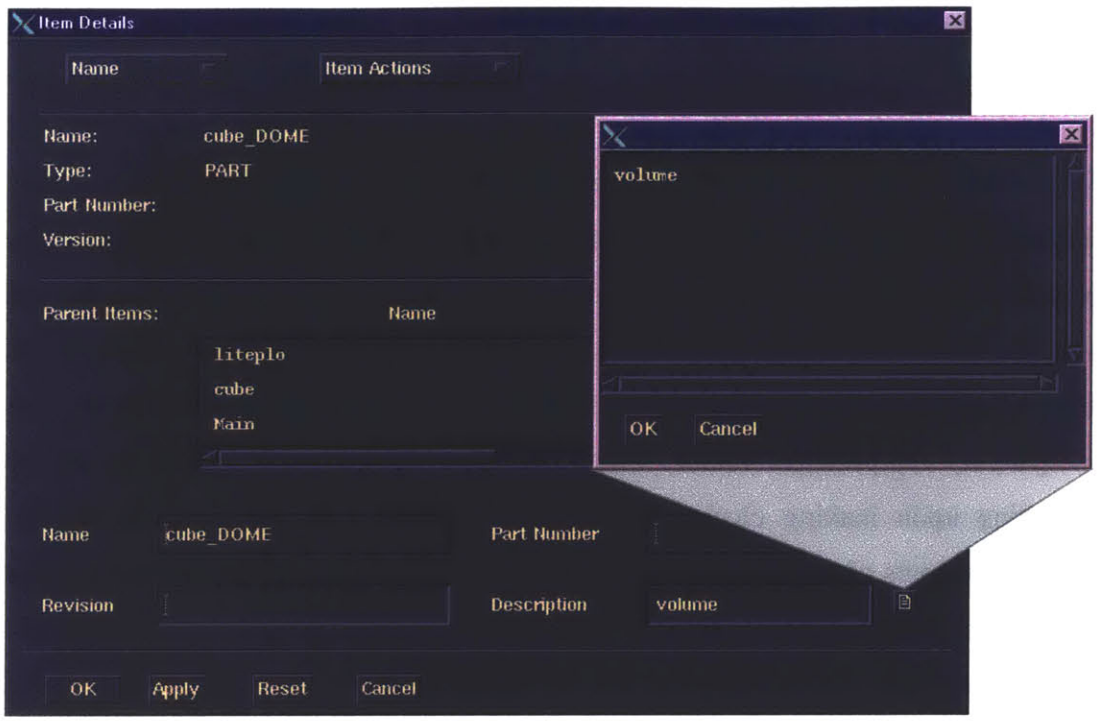

Figure 5-3 Publishing I-deas part properties

To add any of the properties mentioned above, one would simply type the name of the property, for example: *area*, *volume*, *mass*, *center of gravity*, *moment of inertia*. One could similarly declare a user-specified coordinate system about which to have I-deas calculate the mass properties.

Aside from properties, it could be very useful to provide a more complete description of the geometry of a given part or assembly. Even all the dimensions alone can not fully describe geometry for anything more complicated than primitive shapes. So goes the adage, "A picture is worth a thousand words." Well, a three-dimensional representation should be worth at least a million. Thus, another service one can publish for parts or assemblies is a VRML file. One would do this by adding *vrml* to the published interface, followed by the desired path to the VRML file. This file would be exported every time a change is made to the part or assembly.

Similarly, one can publish an IGES 5.3, STEP AP203, or STEP AP214 file of a part or assembly. The process for doing so is identical to that for publishing a VRML file,

except the keyword *iges*, *step203*, or *step214* is used. All of these neutral files are exported from I-deas, so they behave like outputs. Importing neutral files will be discussed in a later section.

5.1.4 Loading a model

Once the I-deas model is published, connecting to it from DOME requires only a few simple steps. If a DOME model is running on a server, a wrapper object to I-deas can be added to the model through a browser enabled client.

This will create an empty container that knows how to communicate with I-deas. In the GUI for this module, there are three text fields labeled *Host*, *Project* and *Model*, respectively (see Figure 5-4). In the *Host* text field goes the name or IP address of the server running I-deas. As mentioned earlier, this can be on a different machine than the one running the DOME server. I-deas stores user information in profiles it calls "projects". The project owning the desired model file should be entered in the middle text field. The last text area contains the name of the model file wished to open.

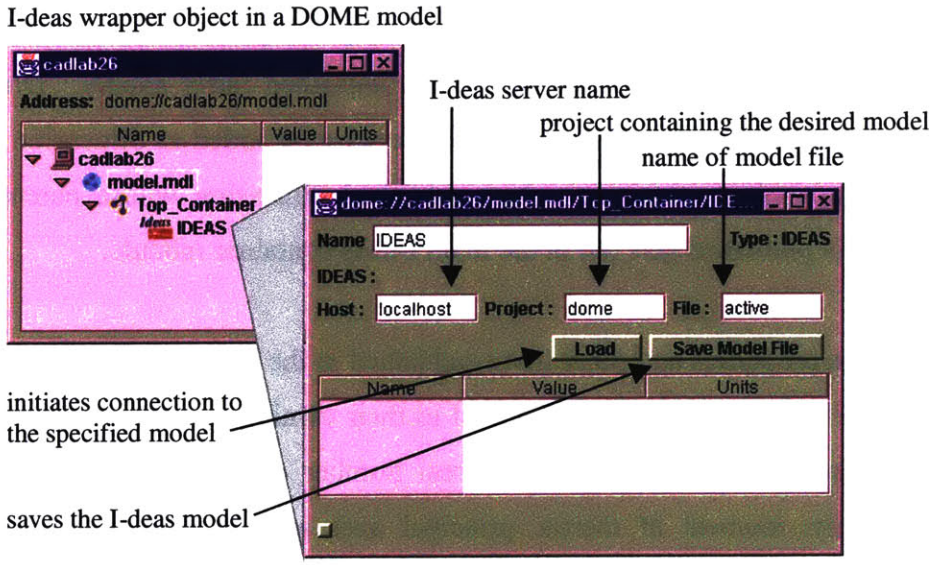

Figure 5-4 I-deas container module GUI

After these are specified, one can press the *Load* button in the GUI. This begins the process of connecting to the I-deas model and loading up the published parameters. First, DOME attempts to make contact with the Orbix daemon running on the specified host

machine. It polls the Orbix daemon for any active I-deas sessions. If an I-deas server is running, DOME connects to that session. If I-deas is not currently running on the host machine, DOME will start up a session. Next, DOME activates the specified project and model file. I-deas stores all the information relating to these items in its own pool of data, so that as long as the project exists, and it contains a model with the given name, there is no need to specify the entire path to the file.

After the correct file is loaded, DOME begins polling the file for published entities. It searches through the model for any assemblies, parts or dimensions that have the appropriate extension appended to their name. For each of these that it finds, it creates a DOME object, or module, and adds it to its object model. The top level I-deas container module acts as a folder for all the subsequent modules that will be added to it. For example, if the I-deas model's main assembly was published, an assembly module will be created and added within the main container module. Furthermore, DOME will recreate the hierarchy of objects established in the I-deas model. Thus, if a published assembly owns a part that has also been published, DOME will add the corresponding part module within the scope of the assembly module. Assembly and part modules are themselves containers for other modules, so this can be accomplished fairly simply. Dimensions belonging to a published part or assembly will likewise be added as dimension modules to the part or assembly containers. Any published dimension whose parent part or assembly was not published gets put in the main I-deas container module.

At this point, the properties and other items published in the assembly and part interfaces also get created as DOME objects and added to their owning containers. Volume, area and mass properties become deterministic real number objects, called Real modules. Center of gravity, moment of inertia, principal axes and principal values properties become vectors or tensors of the appropriate size called Matrix modules. All of the loaded dimensions and properties start out with the correct numerical value(s). In addition, the dimension modules and real modules are assigned the appropriate units. Any file types (IGES, STEP, VRML) that were published become corresponding NF or

VRML modules. Figure 5-5 shows the simple I-deas model published in Figure 5-1 through Figure 5-3 wrapped as DOME objects.

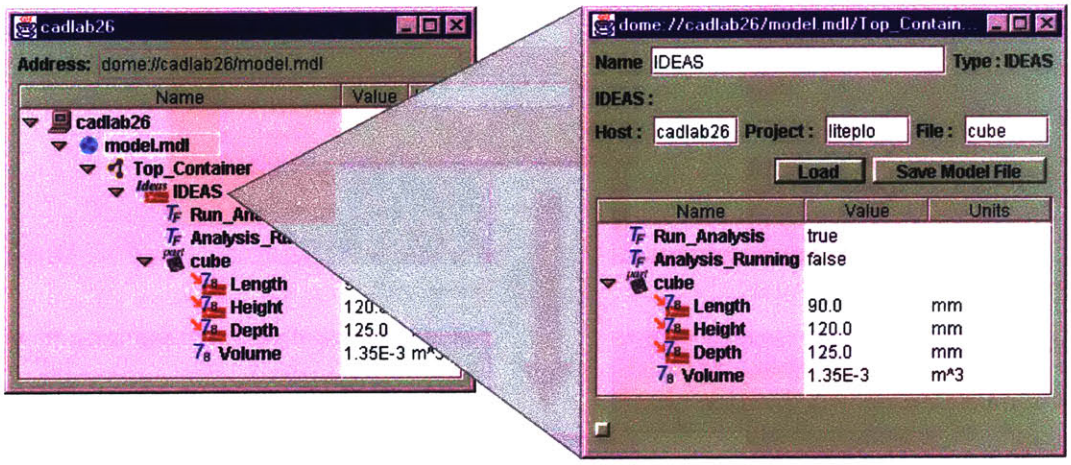

Figure 5-5 I-deas objects loaded in DOME

5.1.5 Making and saving changes

After the modules have been created and added to the DOME object model, one can interact with them. The value of any of the input dimension modules can be changed. This change will first modify the value of the corresponding dimension in I-deas. The CAD model will rebuild based on the design changes. Then, nominally, all of the outputs will update and one would be able to view these changes in DOME. The term "outputs" includes output dimensions, any properties, and neutral file exports. Thus, new values for output dimensions and properties will be propagated to DOME, and any files that need to be rebuilt are automatically re-exported to the appropriate place. This process is depicted in Figure 5-6. Note the new *Volume* value calculated by I-deas.

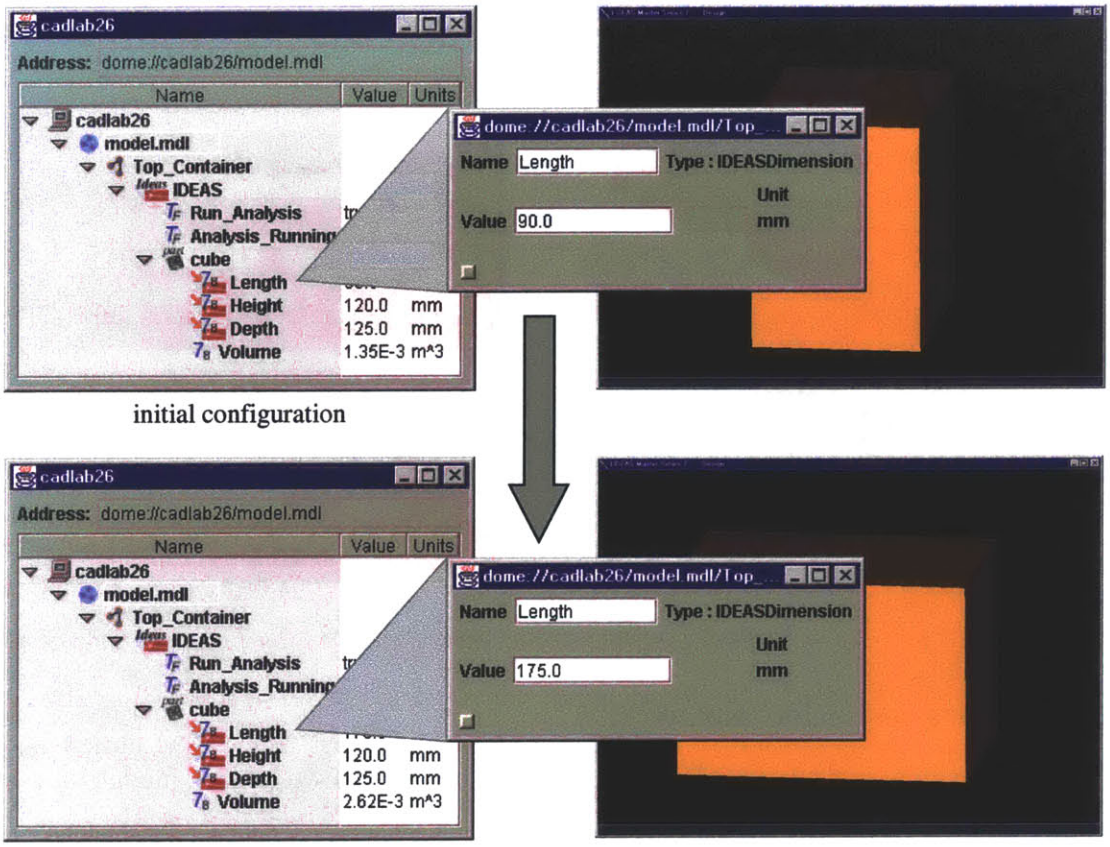

Figure 5-6 Driving an I-deas model parametrically from DOME

Naturally, a change to a part dimension will update any parent assemblies as well, and all of that assembly's outputs will be recalculated or rebuilt. Changes made directly to output dimensions or properties from the DOME interface are not recognized and the modified values are reset.

There are times when one might not need to obtain up-to-date results with every change made in DOME. This is especially relevant when rebuilding the model or exporting files can be time consuming. To facilitate control of this, two Boolean modules are added by default to every top level I-deas container module (see Figure 5-5). One is called *Run_Analysis*, and its state determines whether or not DOME will drive I-deas when a change occurs. This is a user-specified variable, and can be set to true or false at any time. The other is called *Analysis_Running*, and simply shows the status of whether or not I-deas is currently rebuilding or exporting files. This module is not to be changed by

the user, as it provides visual feedback within the DOME environment as to when I-deas is finished updating and the outputs should all be current.

These boolean objects could prove useful when, for example, changes are being made to a number of input dimensions at once. Rather than have I-deas rebuild with each change, which could be time-costly, one could first set the *Run_Analysis* variable to false, make all the desired modifications, then set the variable back to true. This will read in all the changes together and only rebuild the model once. When dealing with large and complex models, this may be a necessary tool to keep modeling on a reasonable time scale.

Additionally, the NF and VRML modules have their own single boolean value that the user can set (see Sections 4.7.1 and 4.7.2). This variable controls whether or not the files are rebuilt when the model changes. The idea is identical to the previous one – time can be saved if the neutral files are not needed at every step – only implemented on an individual module basis.

DOME has the capability of saving modules to a file and loading them at a later time. While it would be possible to save all the values for all the modules within an I-deas container, instead only the parameters needed to reconnect to the model file, i.e. the host, project and model name, are stored. This allows the definition of the DOME model structure to lie within the I-deas model. This is preferred because that is exactly the location where it was defined originally. Saving a second version of this would be redundant, and may conflict with changes made to the interface definition from I-deas.

If, at any point, a DOME user wishes to save the changes made to the I-deas model, there is a *Save Model File* button in the GUI for the main I-deas container module. Pressing this button will prompt I-deas to save the active model file (see Figure 5-4).

5.2 SolidWorks

The same concepts behind the I-deas plugin are applicable to the SolidWorks plugin. Most of the functionality is identical because of the object-based framework upon which

it was written. The implementation, however, is significantly different due to the large discrepancies in API between the two CAD packages.

5.2.1 Software and API

SolidWorks is a PC CAD software developed by SolidWorks Corporation. The version used by the plugin for DOME is SolidWorks 97 Plus.

The API is available through a few languages: Visual Basic, OLE and COM. Visual Basic is not C++ compatible, and thus was not used. OLE, or Dispatch, is based on OpenDoc, which is an interface to files themselves. COM is a more robust language also used by many Microsoft-based products that provides an interface to the application itself. For this plugin, OLE was used. Since the API for SolidWorks does not support distributed methods, SolidWorks has to be running on the same machine as the DOME server. As with any plugin to a third-party application, the API sets the limits on what functionality will be available.

5.2.2 Publishing services

Interface definition for SolidWorks is significantly different than for I-deas. In this case, publishing is done through an external text file. This method was implemented because SolidWorks does not provide a convenient place to store this kind of information within the model file and corresponding API methods to access that information.

Creating this interface text file is fairly straightforward. In an empty file, one must simply write a keyword defining the type of module desired, and then follow that with a few defining parameters. Dimensions are defined by their module name for DOME, their full name from SolidWorks, the part or assembly file in which they exist, their units and an input or output characteristic (see Figure 5-7). SolidWorks stores dimension names as belonging within a certain sketch in a certain part, and it is critical to use this full path name, which is easily accessible from SolidWorks.

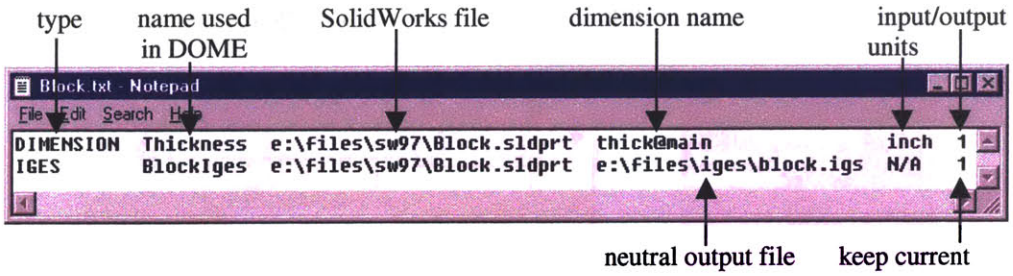

Figure 5-7 Sample SolidWorks publisher file

The output properties such as mass and volume are also available by specifying the part or assembly file for which these parameters are to be calculated. Export VRML 1.0, IGES 5.2 and STEP AP203 modules can be similarly defined, along with their *Keep Current* parameter. DOME makes use of the executable vrml1tovrml2.exe provided by Cosmo Player to convert the VRML files to a format viewable by the VRML plugin. Additionally, one can define a list of different configurations for SolidWorks parts or assemblies. This will allow the DOME user to toggle between the specified configurations of a given part or assembly model.

It is simpler to define the interface in one single publisher file for all the desired CAD models, rather than creating a separate text file for every SolidWorks file to be opened. This is different than the publishing implementation of I-deas, which contains internal text areas stored with the model, and cannot have more than one model file open at a time per session of I-deas.

5.2.3 Loading models

Adding a SolidWorks container module to a DOME model will create an empty shell for the published modules. Within the GUI for this container module, there is a text field corresponding to the publisher file previously created (see Figure 5-8).

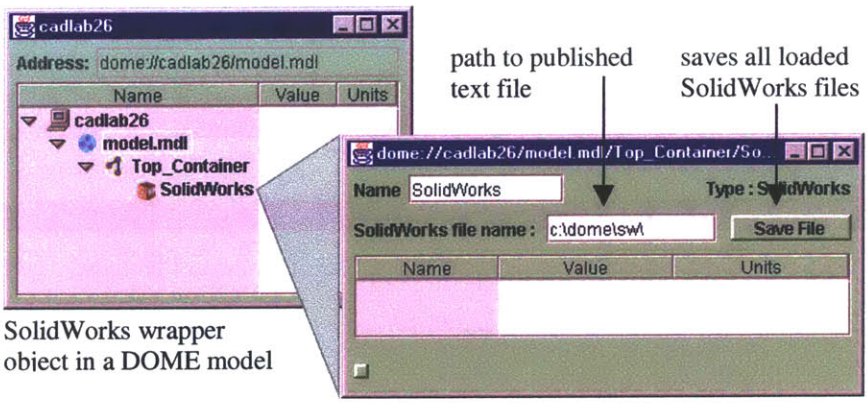

Figure 5-8 SolidWorks Container module GUI

Once an appropriate file name is entered there, DOME attempts to connect to SolidWorks. Like with I-deas, the DOME plugin will attach to an running session of the CAD program or launch a new one if none is available. At this point, the objects declared in the interface definition file are created and added to the main container. Any part or assembly files that need to be accessed are loaded up in SolidWorks' memory. All input and output values are synched with the correct values from the models. Any neutral files that need to be regenerated are exported. Figure 5-9 shows the sample model from Figure 5-7 loaded in DOME.

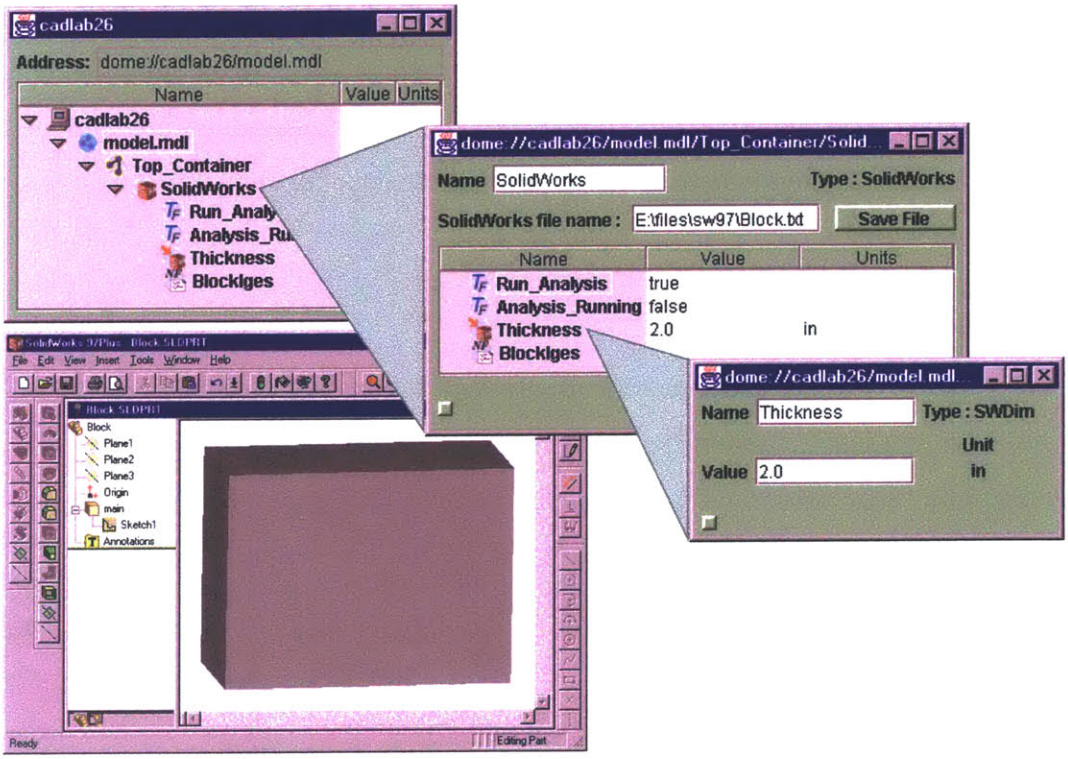

Figure 5-9 SolidWorks objects in DOME and the underlying solid model

If configuration options were defined in the publisher file, a DOME Catalog module is added to the container, as well as some additional modules that this catalog references. This catalog will contain as options the named configurations of the part or assembly. To switch configurations, one can open the GUI for the catalog object, select the desired configuration name, and hit the *Select* button. This feature allows the DOME user to not only explore the design space of continuous variables, but also make arbitrarily large model changes, swap in and out parts via suppression of undesired features. The two Boolean modules described in Section 5.1.5 for the I-deas container are also present in the SolidWorks container. Functionally, they behave identically.

Upon saving, DOME will record only the path to the publisher file, and any catalog data. When reloading from a saved DOME model, all the other internal modules are rebuilt. This, as in the case of the I-deas plugin, is to avoid redundancy of model definition. The SolidWorks container module GUI also has a *Save File* button that will save all the opened files (see Figure 5-8).

6 VIRTUAL ASSEMBLY MODELING IN DOME

6.1 Setting up assembly integration

Thus far, the ability to quickly access object models relating to 3D geometry from the web has been established. The following scenario describes the behavior of a possible system that can emerge based on relating these service-objects. The process involves five major steps, which will be illustrated as the scenario unfolds.

An engineer at a given OEM is responsible for building CAD models of an assembly system and analyzing it. He can do the majority of his work on his own, except for a certain part or subassembly that the company has decided to outsource. The engineer has also been given the task of finding the most suitable supplier to provide this part. He has built a DOME model that represents the assembly system for which he is responsible. Included in his CAD assembly interface is an NF import module to act as a placeholder for this part he needs.

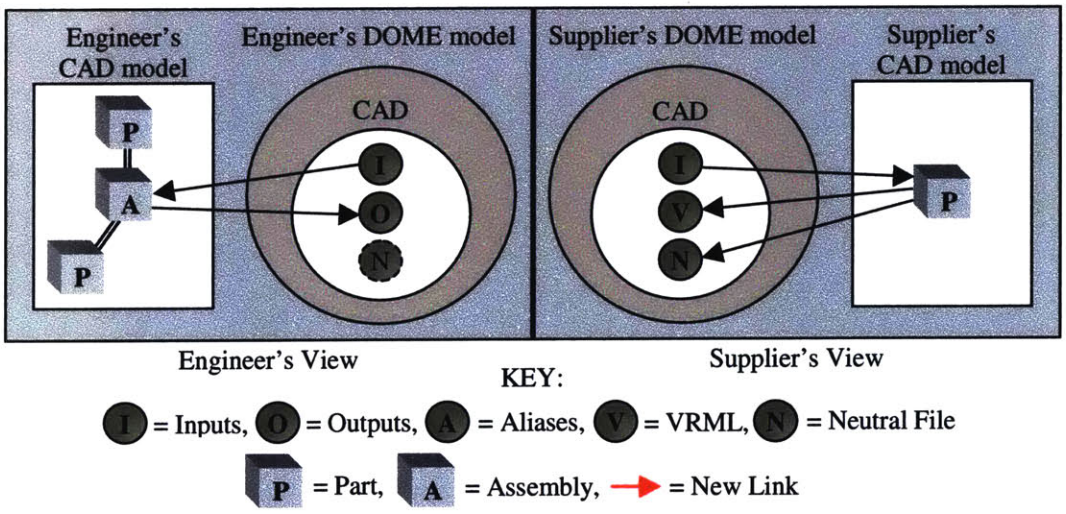

Figure 6-1 Step 1: The two participants publish their CAD models and wrap them in DOME. Note that the engineer's CAD assembly is incomplete and his neutral file object contains no data.

The engineer can now look for DOME services of various suppliers. Using the agent architecture provided in DOME, he can search for suppliers based on keywords and his own custom profile. He can connect to various services on remote servers based on the

search results. Alternatively, if he is aware of any suppliers from more traditional means or through prior relationships, he can connect to their DOME servers. Assuming privileges have been given to him or to a guest account, he can log on that supplier's DOME server.

Once inside the DOME model, he has permissions to view the CAD services that have been published. The engineer knows that this supplier makes the type of part for which he is looking, but doesn't really obtain much understanding of the geometry from just the dimensional interface. However, the supplier provides a VRML module of the part, which the engineer views to be able to conceptualize the geometry. Additionally, he can change the values of input dimensions and visualize the effects they have on the geometry, because the underlying CAD model is running and providing results to DOME. The engineer can paste aliases of the dimension modules from the supplier's model into his own model, from whence he can drive them.

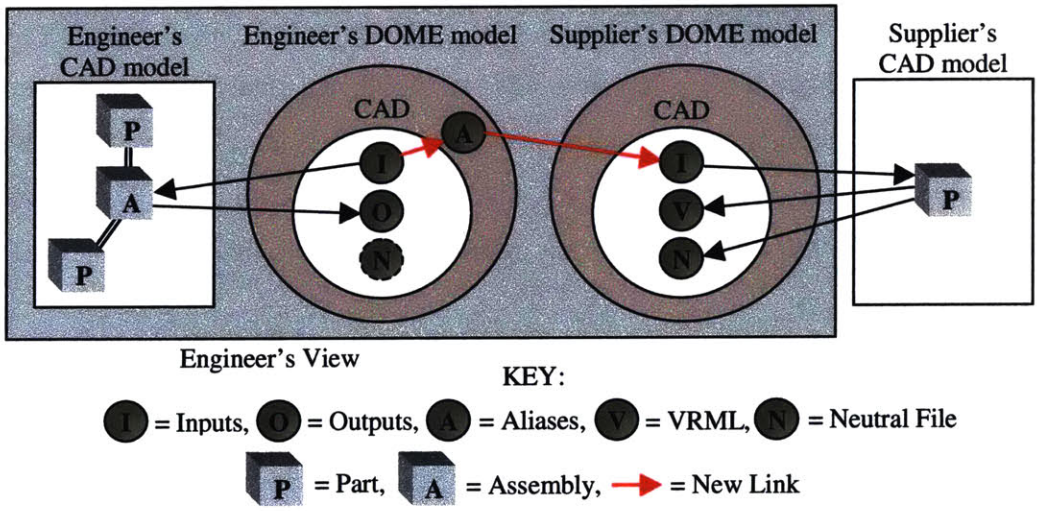

Figure 6-2 Step 2: The engineer drives the supplier's CAD model parametrically from DOME.

The engineer decides that his company may do business with this supplier. He cannot, however, fully analyze his system unless he obtains the supplier's CAD modeling services. To enable this, he asks the supplier to provide him with the service. The supplier does this by logging into the engineer's DOME server, copying the engineer's NF import module, pasting it into his own DOME model and defining a relation that sets the engineer's alias equal to the NF module from the supplier's model, as shown in

Section 4.7.2. This can be done regardless of the CAD systems used by the two participants. At this point, each party can log out of the other's DOME server.

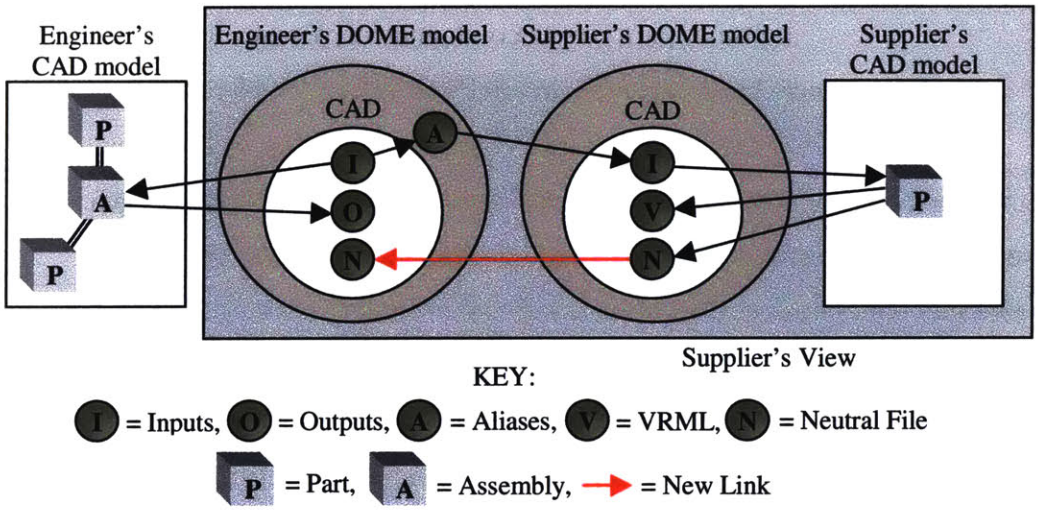

Figure 6-3 Step 3: The supplier drives the engineer's neutral file service by linking it to his own. The engineer's neutral file object now points to a valid file.

6.2 Recording assembly steps

The engineer now has all the tools he needs to build and analyze his full assembly system. The working environment shifts back to his CAD system briefly, where he first has to import the neutral file into his CAD assembly using his CAD system's postprocessor. This file exists locally due to the remote file transfer property of the NF module.

Next he has to record a sequence of events that assemble the part (or subassembly) into the desired position in the assembly. This process of recording is handled differently by differing CAD packages, but most CAD systems provide a way to record a set of tasks and run a single command that will execute the entire set. This function is often used to quickly perform repetitive tasks.

This set of commands should add the imported part to the desired assembly and locate it properly with constraints. Any subsequent assembly tasks should be recorded as well. If color changes are desired, they should be performed at this time. Typically, these files can be nested to call other procedure files.

Recording a robust procedure file is non-trivial. It should be able to correctly perform its functions given a varying input part. CAD commands driven by icons or menus are not affected by this variance, but commands driven by screen picks may not repeat the proper selection reliably. This is highly dependent upon the recording capabilities of the CAD system. Some (I-deas, for example), record view settings, so that picking features with mouse clicks may be appropriate if a consistent view is acquired. One method to bypass this problem is to refer to features by name. This method is quite robust given dimensional changes of the originating geometry, but may not be appropriate when features are added or removed from the original part. Another issue is the visibility of objects such as centerlines and origins, which, when switched off, will not be available for selection. Several iterations may be needed to record an adequate procedure file. However, the engineer is performing the tasks that are his core competency – assembling CAD models – in the environment in which he is acclimated to working.

Once this procedure file is created, it is published in a manner similar to the various interface modules, and the I-deas container module can be reloaded.

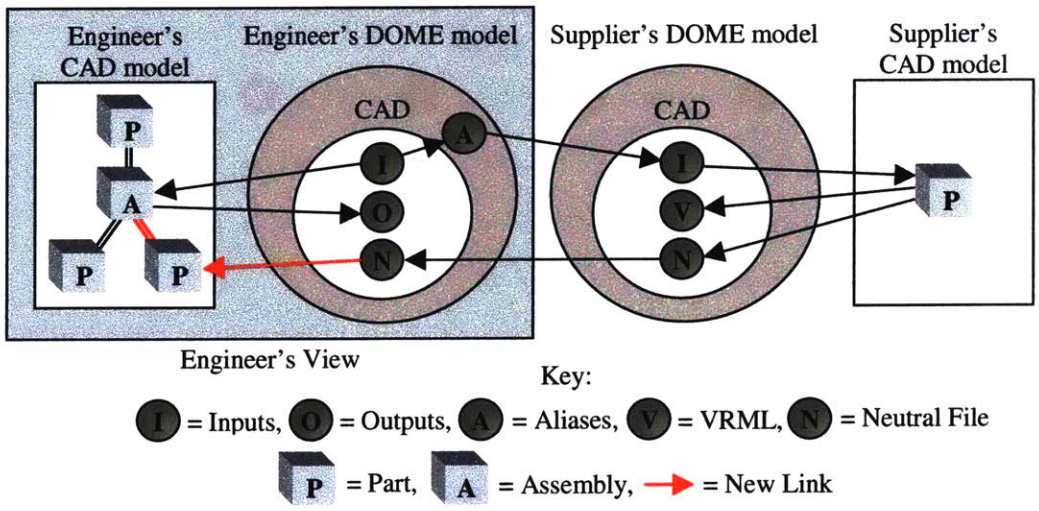

Figure 6-4 Step 4: The engineer incorporates the new part into his assembly.

Note that this part contains no design history and cannot be modified directly from his CAD system.

6.3 Assembly sequence

Now, a number of things happen when the engineer decides to change an input to the supplier's model (see the numbered arrows in Figure 6-5). First, the parameters are changed (1) and the new information is relayed through DOME to the supplier's model (2). The CAD model rebuilds (3), sends DOME new output values, and regenerates the neutral file (4). This causes the NF module on the engineer's DOME server to update (5), the part to be imported again into the CAD assembly (6), the assembly configuration to be recreated (7), and new assembly property values to be calculated (8). This is accomplished interactively over the web with only neutral files passing from the supplier to the OEM engineer. The supplier's detailed proprietary CAD models remains protected within his organization.

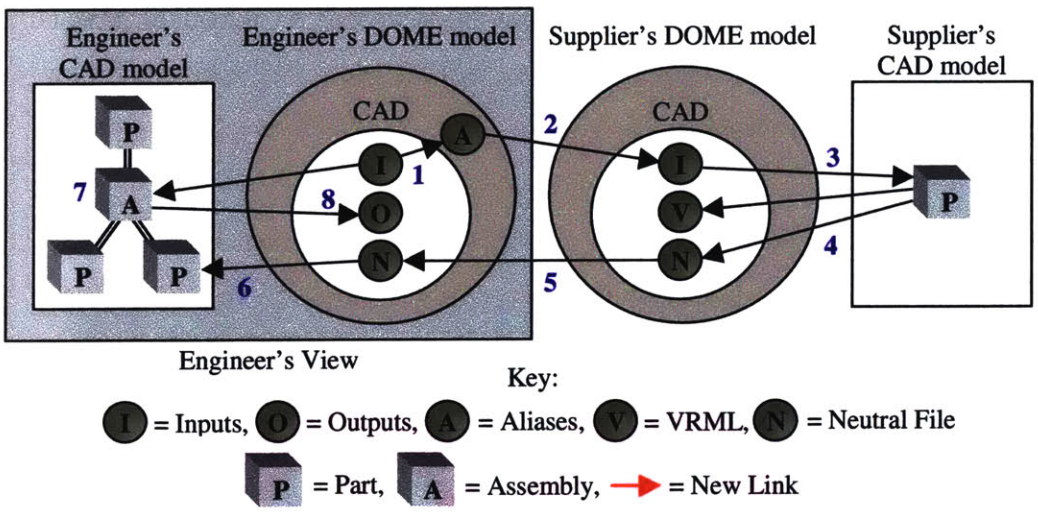

Figure 6-5 Step 5: The engineer rebuilds his assembly by driving the supplier's model.

This reactive chain of data flow links the two underlying CAD simulations together such that the supplier's part attains virtual existence within the engineer's assembly. Thus, the system behaves as if the remote part was *parametrically* editable within the engineer's model, without actual native CAD model transfer.

6.4 Advantages of the DOME assembly paradigm

Assembly modeling in DOME enables parts from different CAD systems on different machines to be assembled and respond to parametric changes automatically without revealing proprietary design history. This has not been implemented previously.

Traditional engineer-supplier relationships are much more difficult to establish and many orders of magnitudes slower to react. Without DOME, a supplier's deliverables may or may not be well defined. Often, complete CAD models are required, which may be seen as a breach into the supplier's intellectual property. In other cases, suppliers never offer any CAD models, in which case a complete computer analysis of the system can not be accomplished.

Even if a regular relationship with a supplier has been established, obtaining results from design changes can be very costly in terms of time to production. Apart from the nuisance of phone calls, emails and schedule conflicts, this sort of design iteration across company lines traditionally takes at least two weeks, and up to a month or more (Kleinke, 2000)! Performing the heretofore described assembly modeling in DOME allows one to receive near-instantaneous results and make quick "what-if" tradeoff analyses that were not previously possible on a reasonable time scale.

Another benefit to the proposed method is the ability to cross the language, time zone and unit system barriers. As long as DOME servers are running and are linked to the underlying simulations, none of those issues remain as problems.

Fundamentally, all of these advantages stem from the facts that using DOME mandates careful definition of the services that are provided or required, and that these services can be made into standardized DOME service-objects. When forced to specify what parameters are actually important to change, and which ones are inconsequential, a clearer picture of what services a supplier is supplying emerges. The ability to see those services from the web makes it easy to know what a supplier's capabilities are, and what capabilities not to expect. The ability to connect and use those services, and to swap them in and out swiftly with those of competing suppliers, makes fast tradeoff analysis a reality. Emergent from this analysis is the supplier that has proven the capability of having the most useful services. Due to global Internet enabling, this supplier need not

be co-located with the partner organization. Because of service object standardization in DOME, CAD interoperability can be realized.

7 CASE STUDY – CAMERA EXAMPLE

7.1 The Scenario

An example is presented to illustrate the concepts heretofore described in the preceding chapters. One CAD model and DOME model each were run on two computers for this simulation. I-deas ran on a 333 MHz computer with 192 M RAM, while SolidWorks ran on a 500 MHz computer with 384 M RAM, both running Windows NT.

The situation involves a CAD engineer who works for a camera manufacturer. His job is to run a CAD analysis on the camera model using I-deas to optimize certain design parameters. In particular, the center of gravity of the entire assembly is used in part to determine where to place the hole for the tripod mount. Included in his assembly are the main body of the camera, some buttons, a view window and a flash (see Figure 7-1).

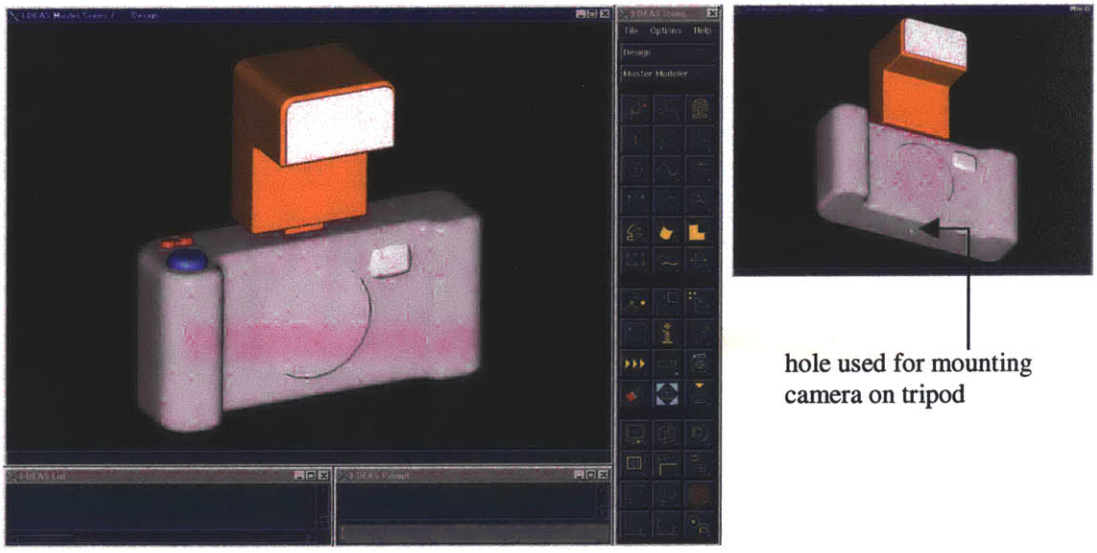

Figure 7-1 Assembler's camera model in I-deas

The company has recently decided to outsource the making of the camera lenses for its new high-end model. This makes the analysis more challenging for the engineer, as he does not have a full CAD model with which to work. Thus, he goes about finding an appropriate supplier as described in Section 6.1.

The model of the supplier to which he connects allows a number of dimensions to be changed, including the overall diameter of the lens as well as its length. In addition to its mass, the supplier provides a VRML, IGES and STEP of the part as outputs. Furthermore, there are configurations made for both with and without the lens cap. The supplier utilizes SolidWorks for CAD modeling (see Figure 7-2).

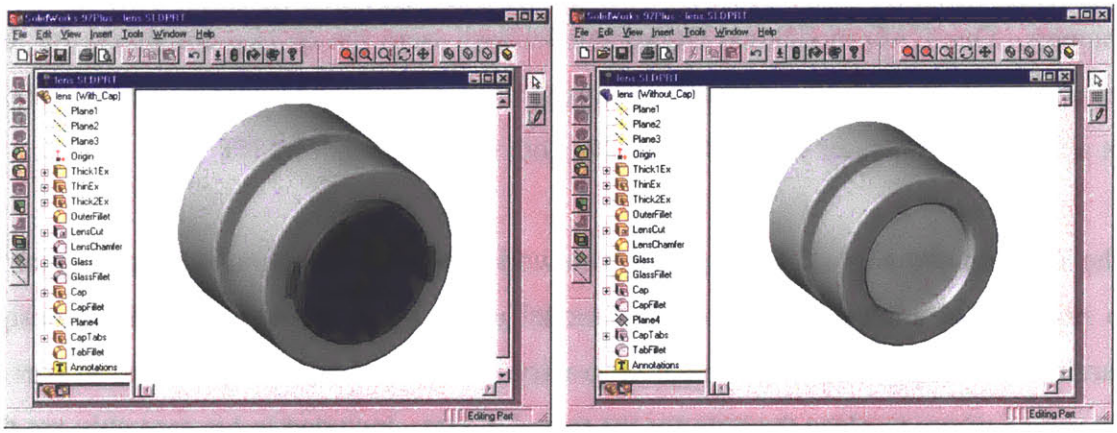

Figure 7-2 Supplier's lens model in SolidWorks, two configurations

7.2 The DOME models

The assembly modeler publishes his interface following the process described in Section 5.1. The following figures show the windows containing the relevant interface text.

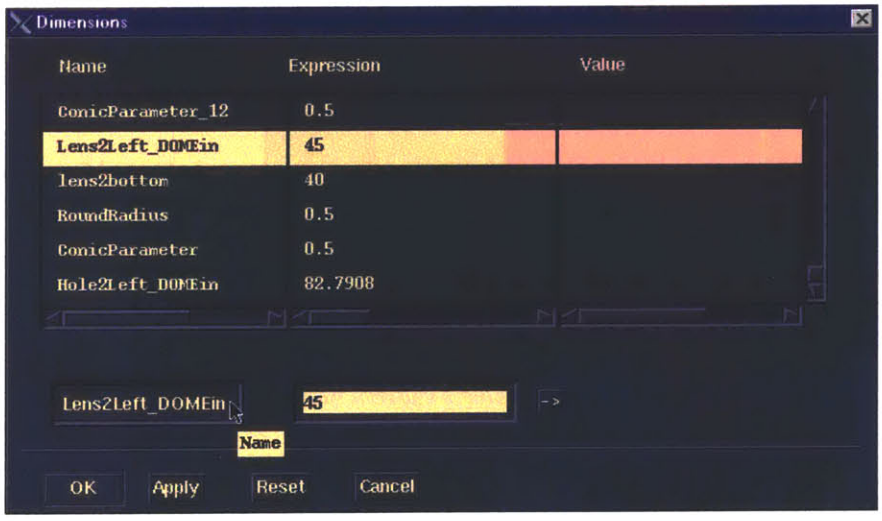

Figure 7-3 Publishing dimensions of the main body of the camera

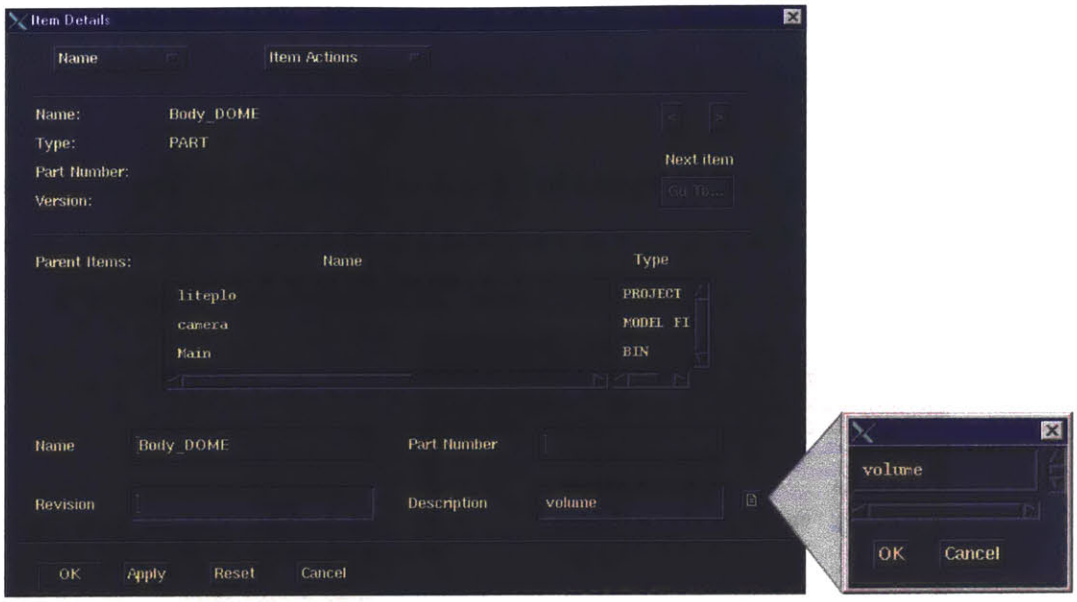

Figure 7-4 Publishing the volume of the camera body

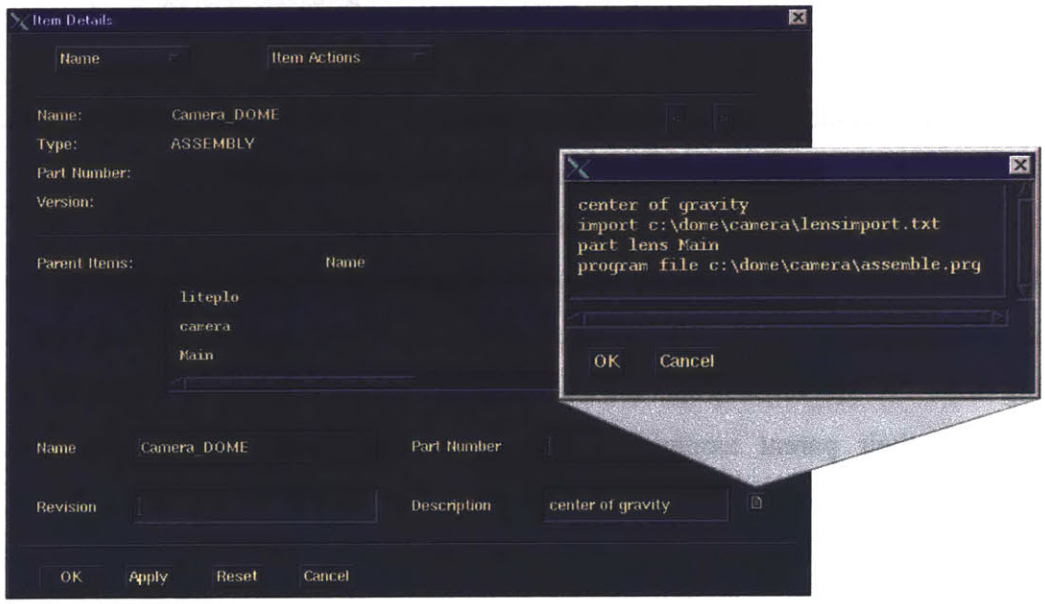

Figure 7-5 Publishing services for the camera assembly

The assembler has published a NF module to be imported into his assembly. This acts as a placeholder for the neutral file he knows he must find. At this point, it has not been determined whether the import type will be an IGES or STEP file. This is irrelevant within his DOME model because they both types are supported by the NF module. He specifies the part name he wants to give the imported object and the bin in which it should get placed. If an IGES will be used, an import flavor may be declared for the

postprocessor to use. The flavor file should be located in the correct directory of the I-deas installation. If no flavor is defined, the default import flavor will be used.

For creating a DOME model to connect to his I-deas simulation, the engineer starts a DOME server and logs into it from a client running in Netscape. An I-deas container module is to the top scope, which loads the I-deas CAD model he has already built.

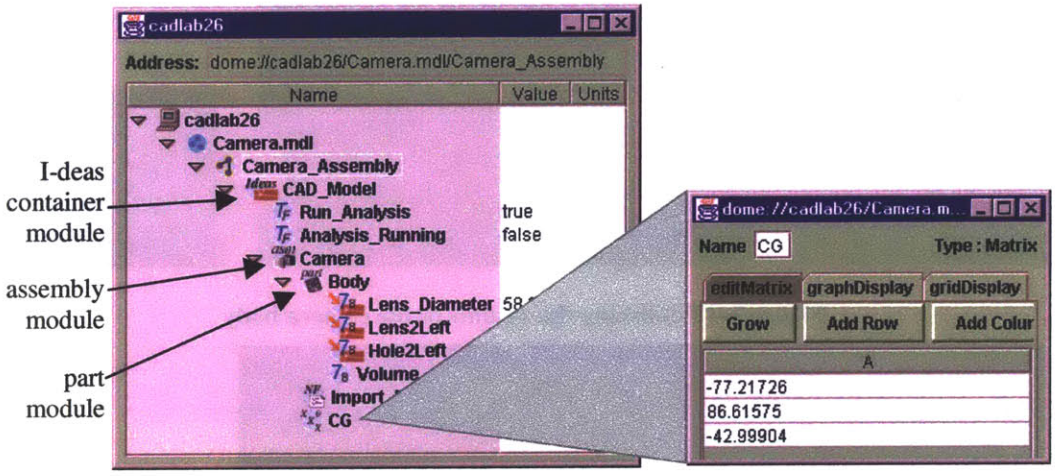

Figure 7-6 Assembly engineer's initial DOME model

***CG* is the assembly's center of gravity.**

Loading this model takes about 30 seconds. Simpler interfaces allow quicker loading times. Part properties, especially matrices, add a significant amount of overhead to the loading time. DOME reconstructs the assembly architecture defined in I-deas by placing parts within their parent assemblies, and dimensions inside their owning parts or assemblies. The various services that were published appear as objects in the DOME model. The engineer could work locally with his own model by changing any of the input dimensions. This would drive the I-deas model and provide new results for *CG* and the volume of the part *Body*. At this time, the *Keep Current* property of the NF module *Import_Part* has been turned off, as it does not reference a valid file.

The supplier creates a file containing the following text that defines the interface that will be made available to DOME (see Figure 7-7).

lens.txt - Notepad

File Edit Search Help

DIMENSION	Extension	C:\dome\lens\lens.SLDPRT	Extension@ThinEx	inch	1
DIMENSION	OuterDiameter	C:\dome\lens\lens.SLDPRT	Diameter@Thick1	inch	1
DIMENSION	InnerDiameter	C:\dome\lens\lens.SLDPRT	Diameter@Thin	inch	1
VOLUME	Volume	C:\dome\lens\lens.SLDPRT	N/A	cubic_meter	0
MASS	Mass	C:\dome\lens\lens.SLDPRT	N/A	kilogram	0
IGES	LensIges	C:\dome\lens\lens.SLDPRT	C:\dome\lens\lens.igs	N/A	1
STEP	LensStep	C:\dome\lens\lens.SLDPRT	C:\dome\lens\lens.step	N/A	1
VRML	LensVrml	C:\dome\lens\lens.SLDPRT	C:\dome\lens\lens.wrl	N/A	1
CONFIGURATION	Without_Cap	C:\dome\lens\lens.SLDPRT	N/A	N/A	1
CONFIGURATION	With_Cap	C:\dome\lens\lens.SLDPRT	N/A	N/A	1

Figure 7-7 SolidWorks publisher file used in case study

The supplier similarly starts a DOME server and creates a SolidWorks container module for his model. Upon reading in the interface definition file and launching the SolidWorks application, the following modules are created and added to the container.

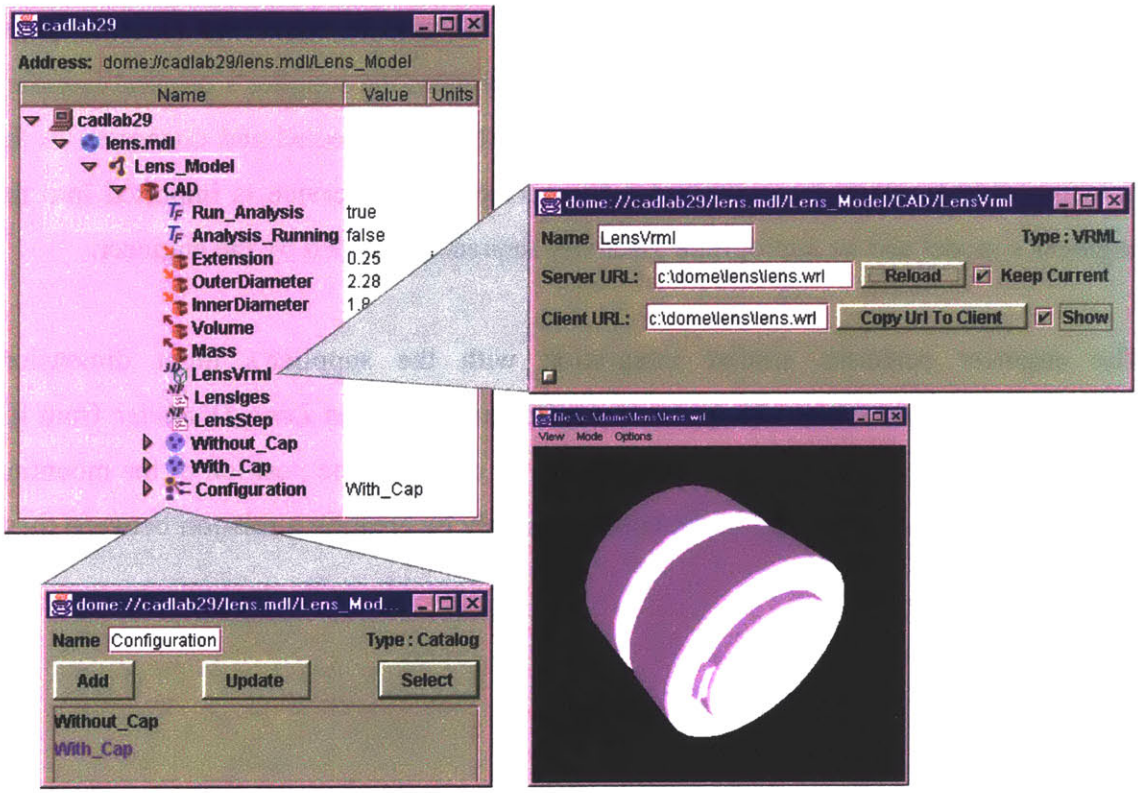

Figure 7-8 Supplier's initial DOME model

The model is in the *With_Cap* configuration. Note that the VRML file is being viewed by the supplier, as his client is on the same machine as his server.

Locally, as with the I-deas modules, the SolidWorks input dimensions could be modified to rebuild the SolidWorks model, obtain new results and export new neutral files. Additionally, one could select a part configuration from the catalog module, which causes the same process of events to occur.

7.3 Integration and Resulting System

These two DOME models can be simply integrated to form a larger system. Each participant adds a container module to the top scope of their respective model. They have chosen to name this container *Integration.* This area will be used for making relations between objects. A representation of this integration process can be seen in Figure 7-9.

The service that the engineer is most interested in using is supplier's neutral file. To make this meaningful to his own model, he asks the supplier to drive a module in his DOME model. The supplier copies *Import_Part* from the assembly model and pastes it into his own *Integration* scope, where he has already placed an alias to his neutral file service, *LensNeutralFile*, and a relation that can be used to equate the services (as in Figure 4-4). The NF module *LensNeutralFile* references the supplier's original *LensStep* service-object (STEP AP203), but it could easily be disconnected and connected to the *LensIges* object, which is an IGES 5.2 file service. This service is imported into the engineer's model and an appropriate assembly sequence recorded by the engineer.

The engineer performs similar integration with the supplier's input dimension, *OuterDiameter.* He drives this parameter with the dimension *Lens_Diameter* from his own model. Finally, as was his original desire, he can drive the position of the mounting hole with the center of gravity of the entire assembly by creating a relation between these two parameters.

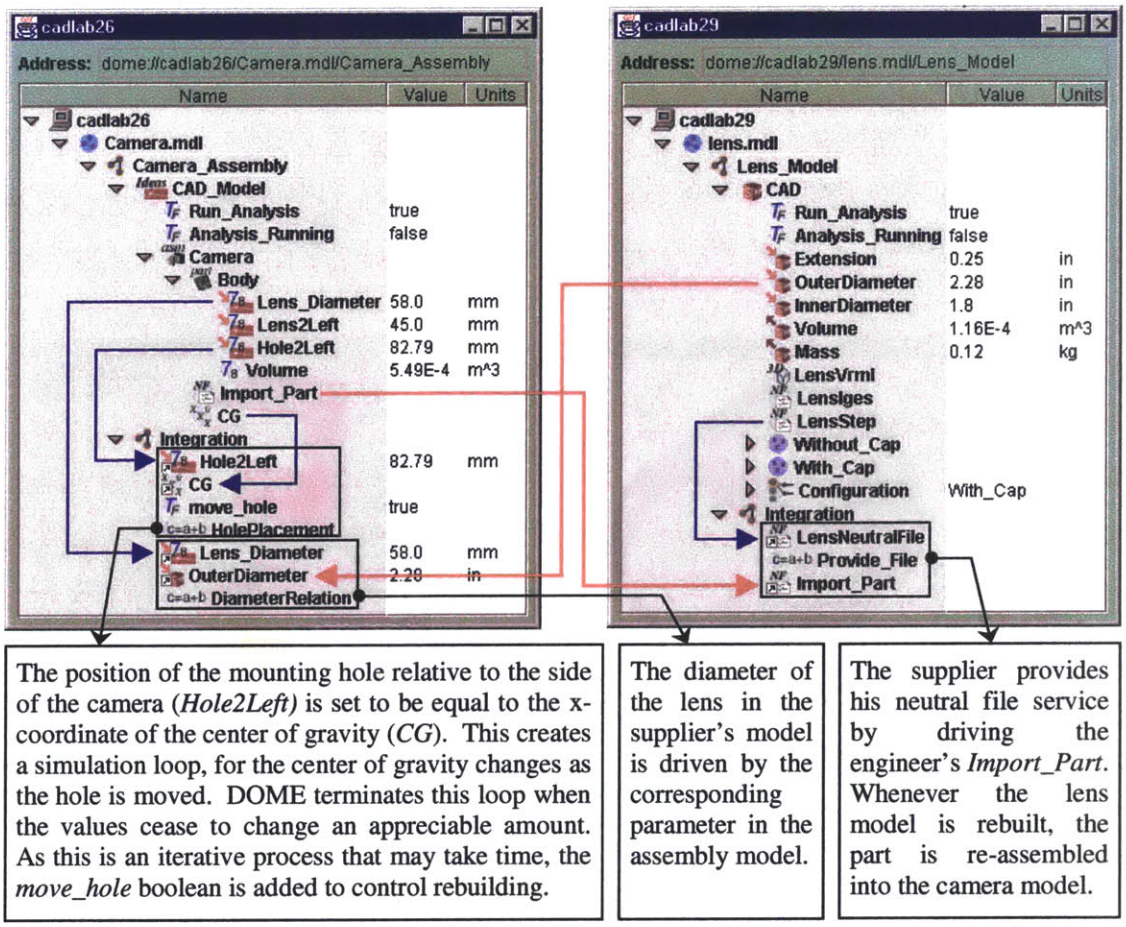

Figure 7-9 Integration of the two models

The arrows denote local and remote alias pasting.

In just a few steps, a system has emerged that is simple in its interface and definition yet complex in the underlying enabling architecture. It allows the engineer to make design changes in DOME and parametrically drive an assembly who's defining geometry exists on different computers.

The initial configuration of the lens has its cap on (see Figure 7-10). The calculations are most relevant with the cap off, so the engineer chooses the *Without_Cap* configuration with the DOME catalog object, *Configuration*. The resulting system state is shown in Figure 7-11. The engineer would also like to be able to analyze the assembly with the telephoto lens fully extended. He can do so by modifying the *Extension* parameter made available by the supplier's model. As before, both models rebuild and new outputs are

attained (see Figure 7-12). Similarly, he can change the two diameters of the lens and alter where the lens is positioned on the camera body, based on the dimensional parameters published from the part and assembly models (see Figure 7-13). Even after the supplier's part is assembled into his CAD model, these types of parametric changes would not be possible without the facilitation of simulation integration through DOME.

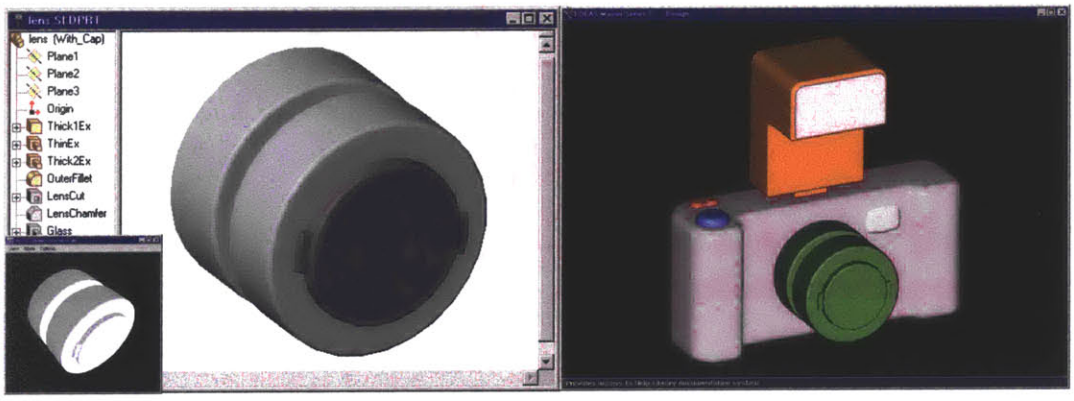

Figure 7-10 Initial state of the linked models

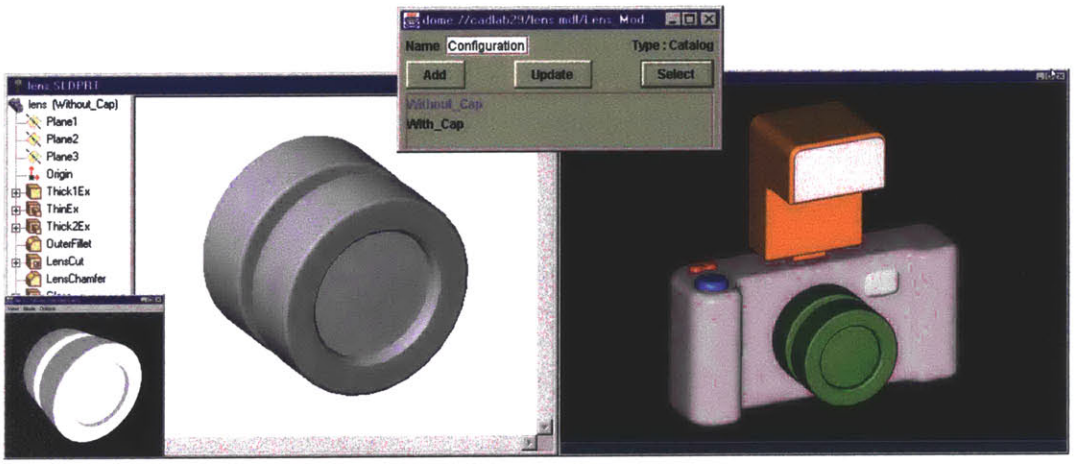

Figure 7-11 Making a catalog change

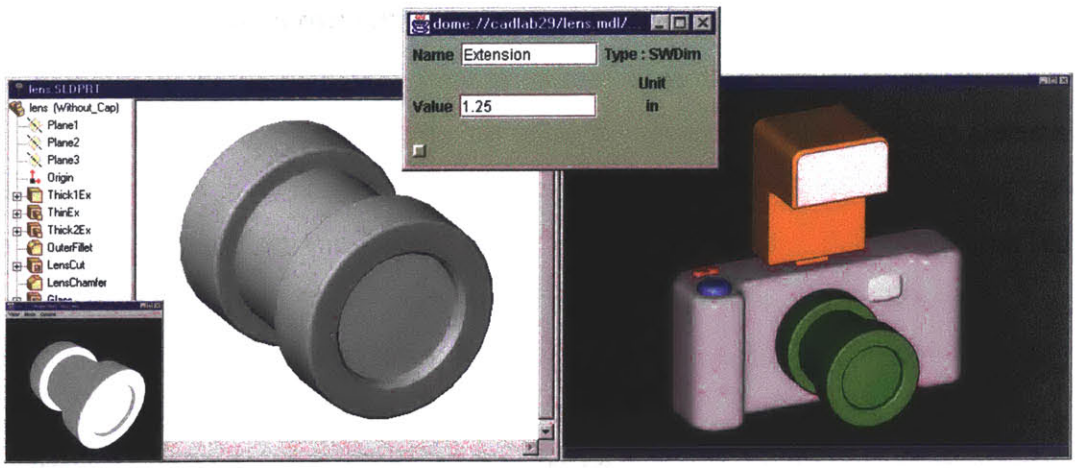

Figure 7-12 Elongating the lens

Figure 7-13 Modifying additional parameters

One iterative loop will be described in detail to clarify the underlying tasks being performed (see Figure 7-14). A change in the *Extension* dimension in DOME causes the SolidWorks model to register this new value and rebuild the lens model (1). Among other outputs, the neutral file service *LensStep* is regenerated (2). Its alias, *LensNeutralFile* (3), fires the relation *Provide_File* (4), which, through the alias (5), writes the contents of the file to *Import_Part* (6). This notifies I-deas to remove the old lens from the assembly, import the newly created file, and reassemble the camera model (7). After this operation, the assembly's center of gravity is recalculated and the module *CG* is updated (8), as is its alias (9). This change fires the relation *HolePlacement* (10),

which modifies the alias dimension *Hole2Left* (11). The original dimension (12) prompts I-deas to move this mounting hole feature accordingly (13), and again regenerate values for the camera's center of gravity (8). *CG* (9) runs *HolePlacement* another time (10). If the new value for *Hole2Left* (11) is significantly different that its previous value (determined by DOME), it will rebuild the I-deas model, and this smaller loop (steps 8-13) will be repeated. Otherwise, the simulation stops there. As mentioned in Section 6.4, this single iteration may take a number of weeks in a traditional development cycle. This example model accomplished the same task in about 60 seconds.

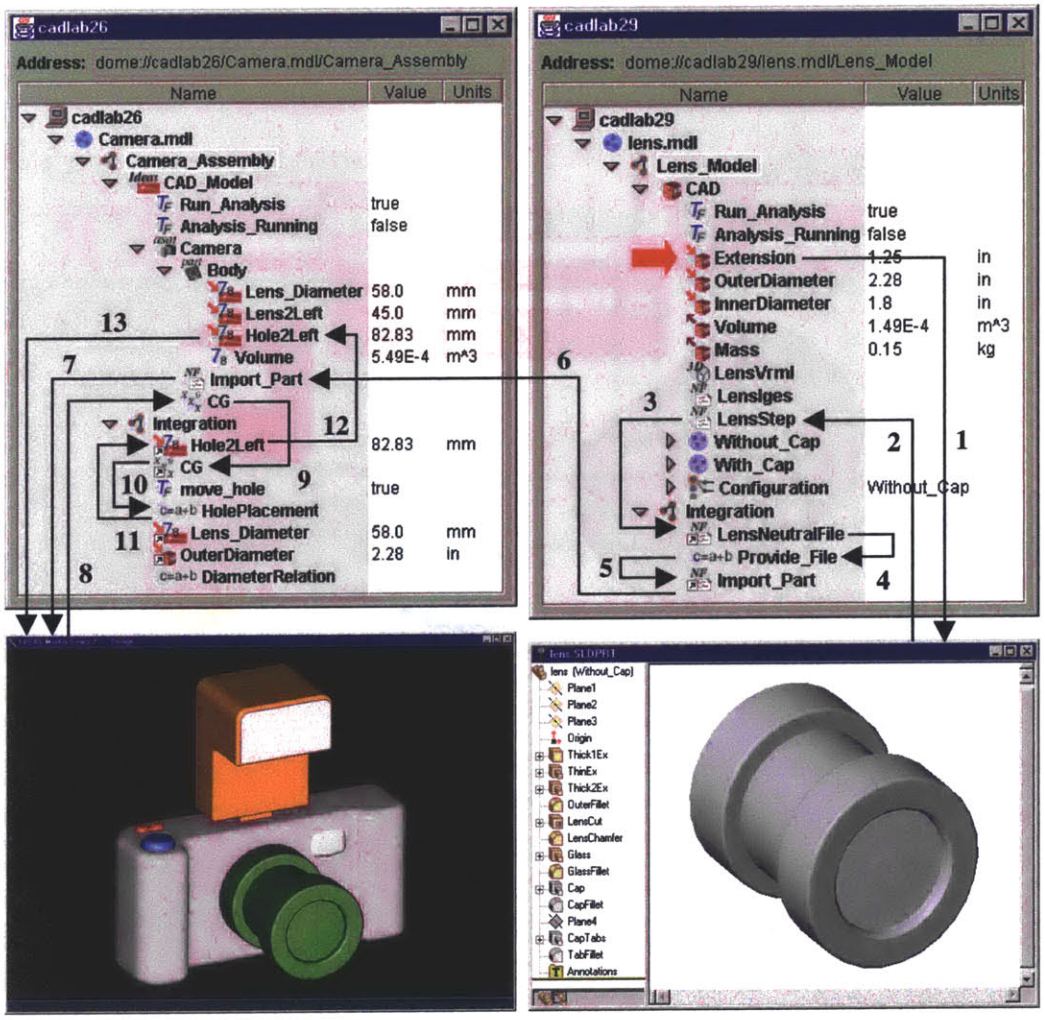

Figure 7-14 Event propagation in distributed model

8 CONCLUSIONS

8.1 Summary

This thesis presented a virtual assembly modeling system for a heterogeneous CAD-engineering environment. The need for sharing geometric data across corporation lines without sharing the actual underlying data models was addressed. Specifically, high-level neutral files such as IGES and STEP were utilized as standards of geometry services to circumvent CAD interoperability issues.

Having incorporated the use of neutral files into the assembly modeling paradigm, the problem of modification of dead geometry was then addressed. First, the underlying capabilities of the DOME system were presented. The additional ability to dynamically interface with CAD applications from DOME, over the web, was developed. This would allow CAD designers to publish their modeling services to DOME. These would be available for parametric manipulation and visualization. Included in the set of services one could provide were said neutral files.

An example involving a camera assembly was provided to illustrate the assembly modeler. By connecting to distributed CAD simulations through DOME, the two models were shown to be individually manipulable. Via integration tools inherent to the DOME structure, these models were linked together to behave like a coherent system. The assembly modeling system made use of this structure, and the existence of neutral file services, to allow incorporation of remotely defined geometry into CAD tools. The procedure for recording appropriate assembly sequences was discussed.

The resulting system was one where full assembly analysis could be performed, and changes to continuous and discrete variables would parametrically update the assembly based on active links to the remotely connected CAD model. No proprietary design history was transferred throughout the process. This assembly modeling system has the

potential to improve geometric design iterations from a traditional time scale of weeks to interactive modeling in minutes.

8.2 Limitations to the paradigm

The proposed assembly modeling paradigm inherits a number of limitations from the tools with which it interfaces. CAD API limitations can block certain functionality from being applied to DOME. Application pre- and postprocessors for neutral files also have their own sets of limitations and robustness issues. The ability of the DOME assembly modeling system to import a valid part is completely dependent upon these processors. In this context, IGES flavors come to the foreground as potential sources of conflict. However, the system does not claim to improve the quality of the individual services being provided to DOME in its service marketplace, but to improve the quality of the resulting system that can emerge.

Versioning issues can cause similar problems. CAD systems often only output one version of any given specification. Often, it the most recent, but this is not always the case. Naturally, they can import previous versions of that format, but never later versions. For example, the latest version of SolidWorks is only compatible with IGES 5.2, so it cannot import an IGES 5.3 file created by I-deas. Again, the assembly modeling system is only as usable as the tools with which it interfaces.

Finally, the task of recording a procedure file as described in Section 6.2 is a difficult one to master. This also falls into the realm of only being as beneficial as the underlying application, but this is not a core functionality of most CAD packages, and cannot be expected to be as robust as the rest of the software. There is modest overhead in learning to create good procedure files, and the greater the number of steps, the more difficult this is to do well.

8.3 Future work

8.3.1 Additional concepts to incorporate

There are a myriad of ways to improve upon the assembly system described in this thesis. Foremost is the assembly sequencing issue. Rather than force the assembler to use his or her best judgement as to how to assemble the remote part (which is often enough), a refined system would provide feature data to orient the assembler's efforts. This might occur in the form of low-level geometric entities such as surface/face names, centerlines, centerpoints, or as high-level features such as pegs, holes and shafts. In the first case, this can be very difficult to do depending on the CAD postprocessor used. Not all of them record entity names. In the second case, further difficulties arise when attempting to agree on the definition of features, as well as when attempting to map those features to entities in the neutral files. This operation is best done by an actual file translator, which is not the intended purpose of the assembly modeler. Thus, providing reasonable assembly-related data along with a neutral file service is a complex task. The highest yielding results per time may be for the service provider to give documentation on the neutral file services containing the best guess for appropriate information. If this documentation is put onto the Internet, it can be easily accessed from DOME.

A robustness feature that could be added to the assembly system is error-checking during the running of the recorded procedure file. This would account for things like selecting an item in empty space, which tends to halt operation. "Smart" program checking could, for example, pick the nearest item. I-deas program files can already check for errors, but it is up to the assembly designer to program what is to be done when problems do arise. A more robust assembly system in DOME is envisioned in which sensible and nonsensical operations can be distinguished and reasonable alternatives chosen when needed. This would drastically reduce the overhead of recording a procedure file for the assembly engineer.

Typical assembly modeling operations such as tolerance, interference and collision analyses would well fit the paradigm in this thesis. These functions are highly dependent

on the application performing them. In the same vein, additional CAD/CAM/CAE tools could be added to the set of DOME-compliant programs. This would expand the usability of DOME in industry. This level of integration has not been sought after yet as DOME-related research.

8.3.2 Existing concepts to implement

There are many additional features than can be added to the assembly modeler. Some of them involve allowing the publication of additional model parameters as existing DOME service types. For example, a catalog could be made containing the various material types known by the CAD system and being able to make tradeoffs in that design space. Other improvements could be made to the user interface, such as a listing of the available IGES flavors. Still others involve adding more automatic processes to the post-import assembly model, thereby decreasing the number of steps that need to be recorded in the procedure file. These include adding the part as an instance of the assembly and changing its color.

The import-and-assemble methodology, which was shown to work in I-deas, was not implemented in SolidWorks, due to issues reloading files that have been loaded into memory. The paradigm, however, remains identical to the one described in this thesis. Assembly procedure files have been recorded using Visual Basic macros.

Also, the method of publishing SolidWorks files is simple, but does not provide the nested object hierarchy that the I-deas plugin does. A better method may be to utilize the Design Table object in SolidWorks.

A final limitation to the current implementation of the assembly modeling system is the loss of the ability to connect to remote sessions of I-deas from DOME. As the neutral file services are propagated through DOME, a local session of I-deas was utilized to be able to import this file. More recent versions of the DOME kernel than the one used for this research allow DOME servers on all platforms, thus reducing this problem.

REFERENCES

Ando, H, Kubota, A, Kiriyama, T, (1998), Study on the collaborative design process over the Internet: a case study on VRML 2.0 specification design, *Design Studies*, vol. 19, pp. 289-307.

Baldwin, D F, Abell, T E, Lui, M M, De Fazio, T L, Whitney, D E, (1991), An Integrated Computer Aid for Generating and Evaluating Assembly Sequences for Mechanical Products, *IEEE Transactions on Robotics and Automation*, vol. 7, no. 1, pp. 78-94.

Bohn, J H, (1995), Removing Zero-Volume Parts from CAD Models for Layered Manufacturing, *IEEE Computer Graphics and Applications*, vol. 15, no. 6, pp. 27-34.

Coles, K, Hou, C A, (1991), Enhancement of IGES Preprocessor for Data Exchange, American Society of Mechanical Engineers paper, pp. 1-5.

Computer Associates International, Inc. (CAI), (2000), http://www.cai.com/cosmo.

Connacher, H I, Jayaram, S, Lyons, K, (1995), Virtual Assembly Design Environment, *Proceedings of the Computers in Engineering Conference and the Engineering Database Symposium*, pp. 875-885.

Crispen, B, comp.lang.vrml faq, (1998), http://home.hiwaay.net/~crispen/vrmlworks/faq.

De Martino, T, Falcidieno, B, Hassinger, S, (1998), Design and engineering process integration through a multiple view intermediate modeller in a distributed object-oriented system environment, *Computer-Aided Design*, vol. 30, no. 6, pp. 437-452.

Dewar, R G, Carpenter, I D, Ritchie, J M, Simmons, J E, (1997), Assembly planning in a Virtual Environment, *Proceedings of PICMET.*

Diehl, A, (1996), Transferring Files from CAD to CAM, *Computer-Aided Engineering*, vol. 15, no. 1, pp. 50, 52.

Duan, G, Wang, J, Liu, D, Lei, N, Bian, W, (1996), Research on an Object-Oriented CAD/CAPP/CAM Integrated System Based on STEP, *Proceedings of the IEEE International Conference on Industrial Technology*, pp. 29-33.

The IGES 5.x Preservation Society, (2000), http://www.iges5x.org/archives/version5x/.

Jayaram, S, Jayaram, U, Wang, Y, Tirumali, H, Lyons, K, Hart, P, (1999a), VADE: A Virtual Assembly Design Environment, *IEEE Computer Graphics & Applications*, vol. 19, no. 6, pp. 44-50.

Jayaram, S, Wang, Y, Jayaram, U, Lyons, K, Hart, P, (1999b), A Virtual Assembly Design Environment, *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pp. 172-179.

Kaplan, G, (1997), Auto manufacture digitizes in depth, *IEEE Spec*trum, vol. 34 no. 11, pp. 62-69.

Kleinke, D, (2000), personal communication.

Lee, D E, Hahn, H T, (1995), Virtual Assembly Production Analysis of Composite Aircraft Structures, *Computers in Engineering ASME Database Symposium*, pp. 867-874.

Magoon, G I, Pfrommer, C, (1989), Ironing Out IGES, *Computer-Aided Engineering*, vol. 8, no. 1, pp. 52, 54.

Mantripragada, R, (1998), Assembly Oriented Design: Concepts, Algorithms and Computational Tools, PhD Thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology.

National Research Council – IMTI, (Strategis Industry Canada Online), (1999), http://strategis.ic.gc.ca/SSG/ad03597e.html–http://strategis.ic.gc.ca/SSG/ad03604e.html.

Nevins, J L, Whitney, D E, Eds., (1989), *Concurrent Design of Products and Processes: A Strategy for the Next Generation in Manufacturing*, McGraw-Hill, New York.

Peng, T, Trappey, A J C, (1998), A step toward STEP-compatible engineering data management: the data models of product structure and engineering changes, *Robotics and Computer-Integrated Manufacturing*, vol. 14, pp. 89-109.

Raghavan, V, Molineros, J, Sharma, R, (1999), Interactive Evaluation of Assembly Sequences Using Augmented Reality, *IEEE Transactions on Robotics & Automation*, vol. 15, no. 3, pp. 435-449.

Rudolph, D, (1995) DXF: Can You Get There From Here?: New Concepts, New Entities, New Attributes, *CADENCE Magazine*, March edition, http://www.cadenceweb.com/1995/mar/r13dxf.html.

Senin, N, Wallace, D R, Borland, N, (2000), Distributed Object-Based Modeling in Design Simulation Marketplace, MIT Computer-Aided Design Laboratory, in review.

Sharma, R, Molineros, J, Raghavan, V, (1997), Interactive Evaluation of Assembly Sequences with Mixed (Real and Virtual) Prototyping, *Proceedings of the IEEE International Symposium on Assembly and Task Planning*, pp. 287-292.

Steffan, R, Schull, U, Kuhlen, T, (1998), Integration of Virtual Reality based Assembly Simulation into CAD/CAM environments, *IECON Proceedings (Industrial Electronics Conference)*, vol. 4, pp. 2535-2537.

Structural Dynamics Research Corporation (SDRC), (1998), I-DEAS Master Series 7 Open I-DEAS API Reference.

Sugimura, N, Moriwaki, T, Kakino, T, (1996), A Study on Assembly Model based on STEP and Its Application to Assembly Process Planning, *Proceedings of the Japan/USA Symposium on Flexible Automation*, vol. 2 pp. 791-794.

Sun Microsystems, Web3D Consortium: The Java 3D and VRML Working group, (1999), http://www.vrml.org/WorkingGroups/vrml-java3d.

Templeman, M, (1996), All in the mind, *Manufacturing Engineer*, vol. 75, no. 3, pp. 133-135.

UK Council for Electronic Business, (2000), http://www.ukceb.org/step/step.htm.

Wallace, D R, Abrahamson, S, Senin, N, Sferro, P, (2000), Integrated Design in a Service Marketplace, *Computer-aided Design*, vol. 32, no. 2, pp. 97-107.

Weber, D, (1996), http://www.larch.umd.edu/resources/faqs/faqdxf/dxfme.html.

Whitney, D E, (1996), The Potential for Assembly Modeling in Product Development and Manufacturing, http://web.mit.edu/ctpid/www/Whitney/papers.html.

Whitney, D E, Nevins, J L, De Fazio, T L, Gustavson, R E, (1994), Problems and Issues in Design and Manufacture of Complex Electro-mechanical Systems, C S Draper Laboratory Report R- 2577.

Wyman, M, (2000), personal communication.

Zapor, W, (1998), http://www.voicenet.com/~waltz/dxf_usrs.html.

Appendix A: SAMPLE NEUTRAL FILES

A.1 Sample IGES file

Excerpts taken from lens.igs used in camera example.

```
SolidWorks IGES FILE using analytic representation for surfaces          S      1
1H,,1H;,11Hlens.SLDPRT,21He:\Bill\iges\lens.igs,39HSolidWorks 97 by SoliG      1
dWorks Corporation,11HVersion 2.0,32,308,15,308,15,11Hlens.SLDPRT,1.,1, G      2
2HIN,50,0.125,14H1000526.043701,1E-008,500.,7Hliteplo,,10,0,;            G      3
     110       1       0       0       0                          01010000D      1
     110       0       0       1       0                                 0D      2
     124       2       0       0       0                          00000000D      3
     124       0       0       4       0                                 0D      4
     100       6       0       0       0               3          01010000D      5
     100       0       0       1       0                                 0D      6
     120       7       0       0       0                          01010000D      7
     120       0       0       1       0                                 0D      8
     126       8       0       0       0                          01010500D      9
     126       0       0       2       0                                 0D     10
     124      10       0       0       0                          00000000D     11
     124       0       0       5       0                                 0D     12
     100      15       0       0       0              11          01010000D     13
     100       0       0       1       0                                 0D     14
...
     144    2540       0       0       0                          00000000D   2201
     144       0       0       1       0                                 0D   2202
110,0.,0.,2.34,0.,0.,41.71007874;                                        1P      1
124,-1.,6.98296672221876E-015,0.,1.7550056904631,                        3P      2
-4.27569270827966E-031,-6.12303176911189E-017,-1.,2.34,                  3P      3
-6.98296672221876E-015,-1.,6.12303176911189E-017,                        3P      4
2.34000000000001;                                                        3P      5
100,2.34,0.877502845,0.,0.887502845,0.,0.887502845,0.;                   5P      6
120,1,5,0.,6.28318530717959;                                             7P      7
126,1,1,1,0,1,0,0.,0.,1.,1.,1.,1.,3.316125569,6.07949799,0.,             9P      8
3.316125569,6.283185307,0.,0.,1.,0.,0.,1.;                               9P      9
124,1.,-1.4432899320127E-014,-1.41252699986858E-015,                    11P     10
3.30776600708679E-015,-1.4738210651899E-014,-1.,                        11P     11
-6.83861036455924E-015,1.60142233753444E-014,                           11P     12
-1.41252699986848E-015,6.83861036455925E-015,-1.,                       11P     13
4.68347296355334;                                                       11P     14
100,2.341736482,0.,0.,0.887350923,0.,0.869007037,0.179494932;           13P     15
126,1,1,1,0,1,0,0.,0.,1.,1.,1.,1.,3.316125569,6.283185307,0.,           15P     16
4.01425727,6.283185307,0.,0.,1.,0.,0.,1.;                               15P     17
...
0.174532925,0.,0.,1.,0.,0.,1.;                                        2191P   2534
126,1,1,1,0,1,0,0.,0.,1.,1.,1.,1.,1.,0.174532925,0.,                  2193P   2535
0.006979342,0.174532925,0.,0.,1.,0.,0.,1.;                            2193P   2536
102,4,2187,2189,2191,2193;                                            2195P   2537
102,4,689,1491,2169,1539;                                             2197P   2538
142,1,2185,2195,2197,1;                                               2199P   2539
144,2185,1,0,2199;                                                    2201P   2540
S      1G      3D   2202P   2540                                          T      1
```

A.2 Sample STEP file

Excerpts taken from lens.step used in camera example.

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION (( 'STEP AP203' ),
       '1' );
FILE_NAME ('lens.step',
       '2000-05-26T08:37:07',
       ( 'MIT' ),
       ( 'MIT' ),
       'SwStep 1.0',
       'SolidWorks 98025',
       '' );
FILE_SCHEMA (( 'CONFIG_CONTROL_DESIGN' ));
ENDSEC;

DATA;
#1= ORIENTED_EDGE ( 'NONE', *, *, #2179, .T. ) ;
#2= VERTEX_POINT ( 'NONE', #379 ) ;
#3= VERTEX_POINT ( 'NONE', #386 ) ;
#4= ORIENTED_EDGE ( 'NONE', *, *, #1421, .T. ) ;
#5= VERTEX_POINT ( 'NONE', #387 ) ;
#6= VERTEX_POINT ( 'NONE', #388 ) ;
#7= ORIENTED_EDGE ( 'NONE', *, *, #2182, .T. ) ;
#8= ORIENTED_EDGE ( 'NONE', *, *, #2223, .T. ) ;
#9= VERTEX_POINT ( 'NONE', #406 ) ;
#10= ORIENTED_EDGE ( 'NONE', *, *, #1480, .F. ) ;
#11= VERTEX_POINT ( 'NONE', #407 ) ;
#12= VERTEX_POINT ( 'NONE', #408 ) ;
#13= ORIENTED_EDGE ( 'NONE', *, *, #2186, .T. ) ;
#14= ORIENTED_EDGE ( 'NONE', *, *, #2226, .T. ) ;
#15= VERTEX_POINT ( 'NONE', #422 ) ;
#16= ORIENTED_EDGE ( 'NONE', *, *, #1520, .T. ) ;
#17= EDGE_LOOP ( 'NONE', ( #258, #270, #1380, #1400, #1439 ) ) ;
#18= EDGE_LOOP ( 'NONE', ( #345, #348, #349, #352 ) ) ;
#19= VERTEX_POINT ( 'NONE', #427 ) ;
#20= ADVANCED_FACE ( 'NONE', ( #434 ), #435, .T. ) ;
...
#373= CIRCLE ( 'NONE', #374, 0.0002539999999999972800 ) ;
#374= AXIS2_PLACEMENT_3D ( 'NONE', #375, #376, #377 ) ;
#375=    CARTESIAN_POINT    (    'NONE',         (    0.02204561170688685400,    -
0.004777294587481071200, 0.05791200000000000500 ) ) ;
#376= DIRECTION ( 'NONE',    ( 1.000000000000000000, 2.408492542386578100E-015,
1.365923996832145200E-014 ) ) ;
#377= DIRECTION ( 'NONE',   ( 1.365923996832146200E-014, 0.000000000000000000, -
1.000000000000000000 ) ) ;
#378=     CARTESIAN_POINT     (     'NONE',          (     0.02204561170688685400,
0.005027435756746172900, 0.05795610663712740700 ) ) ;
#379=     CARTESIAN_POINT     (     'NONE',          (     -0.01968457905069176700,
2.410586057779600100E-018, 0.06069903514268503600 ) ) ;
...
#2303= EDGE_CURVE ( 'NONE', #1521, #1251, #1361, .T. ) ;
ENDSEC;
END-ISO-10303-21;
```

A.3 Sample VRML file

Excerpts taken from lens.wrl used in camera example.

```
#VRML V2.0 utf8

NavigationInfo {
    avatarSize 0
}
Collision {
    collide FALSE
    children [
       Group {
           children [
              DirectionalLight {
                  direction 0 0 -1
              },
              Group {
                  children [
                     Shape {
                         appearance
                            Appearance {
                                material
                                   Material {
                                       emissiveColor    0.615686 0.592157 0.776471
                                   }
                            }
                         geometry
                            IndexedFaceSet {
                                coord
                                   Coordinate {
                                       point     [       -0.0196008       0.00502744
0.0452561,
…
0.0218278 0.00470966 0.0468865,
                                                                  0.0195022   0.00475643
0.0467915,
                                                                  0.0218278   0.00475871
0.0467801 ]
                                   }
                                solid     FALSE
                                creaseAngle      0.5
                                coordIndex       [ 0, 1, 2, -1, 3, 4, 5, -1,
                                                  6, 7, 8, -1, 9, 10, 11, -1,
                                                  12, 13, 14, -1, 15, 16, 17, -1,
                                                  18, 19, 20, -1, 21, 22, 23, -1,
                                                  24, 25, 26, -1, 27, 28, 29, -1,
                                                  30, 31, 32, -1, 33, 34, 35, -1 ]
                            }
                     }
                  ]
              }
           ]
       }
    ]
}
```