

**CONSTRAINT-ENABLED DESIGN INFORMATION REPRESENTATION FOR
MECHANICAL PRODUCTS OVER THE INTERNET**

by

Yan Wang

B.S. in E.E., Tsinghua University, 1996

M.S. in E.E., Chinese Academy of Sciences, 1998

Submitted to the Graduate Faculty of
School of Engineering in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

University of Pittsburgh

2003

UNIVERSITY OF PITTSBURGH

SCHOOL OF ENGINEERING

This dissertation was presented

by

Yan Wang

It was defended on

August 1, 2003

and approved by

Bopaya Bidanda, Professor, Department of Industrial Engineering

Michael R. Lovell, Associated Professor, Department of Mechanical Engineering

Ming-En Wang, Assistant Professor, Department of Industrial Engineering

Raymond R. Hoare, Assistant Professor, Department of Electrical Engineering

Dissertation Director: Bartholomew O. Nnaji, Professor, Department of Industrial Engineering

© Copyright by Yan Wang 2003
All Rights Reserved

ABSTRACT

CONSTRAINT-ENABLED DESIGN INFORMATION REPRESENTATION FOR MECHANICAL PRODUCTS OVER THE INTERNET

Yan Wang, Ph.D.

University of Pittsburgh, 2003

Global economy has made manufacturing industry become more distributed than ever before. Product design requires more involvement from various technical disciplines at different locations. In such a geographically and temporally distributed environment, efficient and effective collaboration on design is vital to maintain product quality and organizational competency. Interoperability of design information is one of major barriers for collaborative design. Current standard CAD data formats do not support design collaboration effectively in terms of design information and knowledge capturing, exchange, and integration within the design cycle. Multidisciplinary design constraints cannot be represented and transferred among different groups, and design information cannot be integrated efficiently within a distributed environment. Uncertainty of specification cannot be modeled at early design stages, while constraints for optimization are not embedded in design data.

In this work, a design information model, Universal Linkage model, is developed to represent design related information for mechanical products in a distributed form. It incorporates geometric and non-geometric constraints with traditional geometry and topology elements, thus allows more design knowledge sharing in collaborative design. Segments of

design data are linked and integrated into a complete product model, thus support lean design information capturing, storage, and query. The model is represented by Directed Hyper Graph and Product Markup Language to preserve extensibility and openness. Incorporating robustness consideration, an Interval Geometric Modeling scheme is presented, in which numerical parameters are represented by interval values. This scheme is able to capture uncertainty and inexactness of design and reduces the chances of conflict in constraint imposition. It provides a unified constraint representation for the process of conceptual design, detailed design, and design optimization. Corresponding interval constraint solving methods are studied.

DESCRIPTORS

Collaborative Design	Computer-Aided Design
Constraint Representation	Data Model
Design	Extensible Markup Language
Feature Representation	Geometric Modeling
Interoperability	Interval Analysis
Parametric Family	Persistent Naming
Product Markup Language	Reliable Computation
Robustness	Semantic ID
Uncertainty	Universal Linkage
Xlink	XML Schema
XPath	

ACKNOWLEDGEMENTS

Sincerely I wish to express my gratitude to my advisor Professor Bartholomew O. Nnaji for his years of guidance and encouragement. His invaluable support and mentoring allow me to keep research endeavor and are priceless contribution to this work.

Special thanks to the committee members, Professor Bopaya Bidanda, Professor Michael R. Lovell, Professor Ming-En Wang, and Professor Raymond R. Hoare for their commitment and insightful comments.

I would like to extend my thanks to the faculty and staff of Department of Industrial Engineering for providing indispensable resources and maintaining the homely environment. I also thank all old and new members of the Automation and Robotics Laboratory for the enjoyable companionship and their help, particularly, Justin Kidder, Celestine Aguwa, Obinna Muogboh, Kyoung-Yun Kim, Salil Desai, Adaeze Mbaezue, Pamela Ajoku, Amer Momani, and David Manley.

Finally, I am grateful to my family members for their love and understanding.

TABLE OF CONTENTS

1.0	INTRODUCTION	1
1.1	Role of Internet in Product Design	3
1.2	Importance of Capturing Knowledge in Design	6
1.3	Research Objectives and Overview	7
2.0	DESIGN KNOWLEDGE REPRESENTATION FOR CAD	11
2.1	Current Standard Formats	12
2.1.1	The Initial Graphics Exchanges Specification (IGES)	13
2.1.2	The Standard for the Exchange of Product Model Data (STEP)	13
2.2	General Data Models	15
2.3	Design Knowledge Representation	17
2.3.1	General Knowledge Representation Languages	19
2.3.2	Design Modeling Languages	21
2.3.3	Requirements for Design Information and Constraint Representation	23
3.0	UNIVERSAL LINKAGE MODEL	27
3.1	Information Elements of UL Model	28
3.2	Directed Hyper Graph	30
3.3	Universal Linkage among Entities	35
4.0	SYNTAX AND SEMANTICS OF PRODUCT MARKUP LANGUAGE	37
4.1	The Syntax of PML	39
4.2	The Schema of PML	40
4.3	Graph Decomposition	44
4.4	Demonstration	52
5.0	DESIGN FEATURE AND CONSTRAINT REPRESENTATION	57
5.1	Design Feature Representation	61
5.1.1	Dual representation of features	64
5.1.2	Feature dependency	71
5.2	Geometric Constraint Representation	74
5.2.1	Robustness in Geometric Computation	76
5.2.2	Interval-value numerical constraints	78
5.3	Non-geometric Constraint Representation	87
5.4	Entity ID Persistency	89
5.4.1	Parametric family	92
5.4.2	Semantic ID	97
5.4.3	Curve, Edge, and Point Mapping	107

6.0	INTERVAL GEOMETRIC MODELING	112
6.1	Preliminaries of Traditional Interval Analysis.....	114
6.2	Concepts of Interval Geometric Modeling (IGM)	115
6.2.1	Interval Definitions in IGM	117
6.2.2	Sampling Relation between Real Number and Interval Number.....	119
6.3	Geometry Description in IGM	123
6.3.1	Modeling Uncertainty in IGM	123
6.3.2	Solving Under-constrained Problems	125
6.3.3	Solving Over-constrained Problems	126
6.4	Solving Equations in Interval Geometric Modeling	127
6.4.1	Interval Linear Equations.....	128
6.4.2	Interval Nonlinear Equations	129
6.4.3	Interval Inequalities	135
6.4.4	A Numerical Example.....	136
6.5	Design Refinement.....	138
6.5.1	Interval Subdivision	139
6.5.2	Constraint Re-specification.....	146
7.0	IMPLEMENTATIONS AND TESTS	149
7.1	Service Architecture of Pegasus	149
7.2	UL-PML Scheme in Collaborative Design.....	156
7.2.1	PML Modeler.....	157
7.2.2	Lean Information Transfer Based on HTTP	159
7.2.3	Lean Information Transfer Based on CORBA	160
7.2.4	Distributed Design Information Integration.....	165
7.2.5	Mapping Between Native CAD Data Models and PML Model.....	168
7.2.6	Constraint Propagation and Management.....	173
8.0	SUMMARY AND FUTURE WORK	177
	APPENDIX I – XML SYNTAX	184
	APPENDIX II – XML NAMESPACE SYNTAX.....	188
	APPENDIX III – XLINK SYNTAX.....	189
	APPENDIX IV – XPATH SYNTAX.....	190
	APPENDIX V – XPOINTER SYNTAX	192
	APPENDIX VI – EXAMPLES OF PML SCHEMA	193
	BIBLIOGRAPHY.....	202

LIST OF TABLES

Table 1: Examples of Entities	28
Table 2: Examples of Relations	29
Table 3: Examples of design features	66
Table 4: Categories of common geometric constraints.....	75
Table 5: Coordinates of ending vertex with different radii.....	79
Table 6: Numerical results of the bracket example	137
Table 7: Initial result of Section 6.4.4.....	142
Table 8: Subdivision level 1	142
Table 9: Subdivision level 2	142
Table 10: Subdivision level 3	143
Table 11: Service sequence in a client/server transaction	155
Table 12: Selective topology transferred to <i>Mould2</i>	161
Table 13: Constraint examples in <i>mold1.xml</i> , <i>mold2.xml</i> , and <i>constr.xml</i>	166
Table 14: Research Summary	182

LIST OF FIGURES

Figure 1: Geometric entities and non-geometric entities in DHG	30
Figure 2: Static relations and dynamic relations in DHG	31
Figure 3: An example of aggregation relation in DHG	32
Figure 4: An example of association relation in DHG	33
Figure 5: An example of generalization relation in DHG.....	33
Figure 6: An example of geometric constraint in DHG.....	34
Figure 7: An example of non-geometric constraint in DHG	34
Figure 8: A triangle with dimensional constraints.....	34
Figure 9: DHG representation of the triangle with dimensional constraints in Figure 8.....	35
Figure 10: Universal linkage between files	36
Figure 11: A point in PML.....	40
Figure 12: Schema of POINT referring to coordinates.....	42
Figure 13: Schema of LINE.....	43
Figure 14: Syntax of reference ID	44
Figure 15: Graph decomposition algorithm.....	46
Figure 16: Assumed topological hierarchy	47
Figure 17: Tree structure of entities in Figure 8 after graph decomposition	48
Figure 18: PML representation of the triangular part in Figure 8 and Figure 9	49
Figure 19: A tetrahedron.....	50
Figure 20: DHG model of the tetrahedron in Figure 19	50
Figure 21: PML model of the tetrahedron in Figure 19.....	51
Figure 22: A part to be designed by two groups.....	53
Figure 23: The body section of the part.....	53
Figure 24: The head section of the part	54
Figure 25: Universal linkage by URI.....	54
Figure 26: Tree of geometric and non-geometric constraints.....	60

Figure 27: Priori feature of protrusion in DHG	67
Figure 28: Posteriori feature of protrusion in DHG.....	68
Figure 29: A solid feature example.....	69
Figure 30: Feature definition procedure in Figure 29.....	70
Figure 31: PML description of feature information of Figure 29.....	71
Figure 32: Algorithm to list dependent features of a feature for reference dependency	73
Figure 33: Algorithm to list features that a feature depends on for reference dependency	73
Figure 34: An example of numerical errors.....	79
Figure 35: Constraint examples in a piston and its assembly	81
Figure 36: Piston features and geometric constraints in PML.....	85
Figure 37: Simple link geometric constraint.....	86
Figure 38: Examples of non-geometric constraints	88
Figure 39: An example of naming persistency problem.....	90
Figure 40: Example of intersect continuity.....	94
Figure 41: An Example of face bounded by surfaces and edge bounded by surfaces.....	101
Figure 42: Examples of intersecting surfaces	102
Figure 43: Syntax of IDs for topological and geometric entities.....	106
Figure 44: Range of a point specified by interval numbers	117
Figure 45: Relations between intervals.....	118
Figure 46: A 2D triangle geometry specified by intervals.....	124
Figure 47: Constraint-driven geometry in interval modeling	124
Figure 48: An example of under-constrained geometry in bracket design.....	126
Figure 49: An example of over-constrained geometry in bracket design.....	127
Figure 50: Algorithm of extended Gauss-Seidel method for solving linear equations (6.6).....	129
Figure 51: Linear enclosure of nonlinear interval function	132
Figure 52: <code>RootIsolation</code> procedure based on Descartes' rule of signs.....	133
Figure 53: Constraint equations of Figure 49 (b) in separable form.....	136
Figure 54: Convergence of Interval calculation in the bracket example	137
Figure 55: Variation allowance of the bracket.....	138
Figure 56: The solution set represented as a 2D region.....	140
Figure 57: Two-dimensional interval vector subdivision	140

Figure 58: Subdivide procedure for power interval elevation.....	141
Figure 59: Comparison of different levels of subdivisions.....	145
Figure 60: Refined bracket design by subdivision.....	146
Figure 61: Relations of two constraint subsets	147
Figure 62: Service triangular relationship.....	151
Figure 63: Pegasus system architecture	152
Figure 64: Services provided by Service Manager.....	153
Figure 65: Peer-to-peer relationship among service providers	155
Figure 66: Sequence diagram of a service	156
Figure 67: Architecture of PML modeler	158
Figure 68: Interface of PML modeler	158
Figure 69: Lean information transfer base on HTTP.....	159
Figure 70: A pair of moulds in collaborative design	161
Figure 71: The first mould designed at the server site.....	162
Figure 72: Design library for data sharing.....	162
Figure 73: The second mould designed at the client site	163
Figure 74: Updated design of the first mould	164
Figure 75: Updated second mould by translating corresponding references	164
Figure 76: Lean information transfer based on CORBA	165
Figure 77: Design constraints in <i>mold1.xml</i>	167
Figure 78: Design constraints in <i>mold2.xml</i>	167
Figure 79: Design constraints in <i>constr.xml</i>	168
Figure 80: PML model as a medium for selective information transfer between CAD systems	169
Figure 81: The architecture of the ACIS modeler	169
Figure 82: A jig model in SAT format in the ACIS modeler	171
Figure 83: The translated jig model in PML format in PML native modeler at server site.....	171
Figure 84: The jig model in PML format received at client site.....	172
Figure 85: The translated jig model in the ACIS modeler.....	172
Figure 86: Process of constraint format query from constraint library.....	175

1.0 INTRODUCTION

The emergence of Internet technologies and their widespread proliferation has had a tremendous impact on industry. The Internet provides a convenient medium for faster business transactions. High-speed business information exchange over Internet shrinks the span of both time and space. Free information flow facilitates a global free market. The global market has a trend to shift its gravity to e-business. For example, US companies invested \$9.5 billion on the Internet over the first 10 months of 1999. European venture capitalists (VCs) invested \$333.9 million in Internet companies over the same period [1]. With business-to-business e-commerce expecting to top \$1.3 trillion by 2003, the Internet is a key driving force in the new millennium as manufacturers strive to optimize their supply chain [2]. Nevertheless, the Internet brings challenges to manufacturers. Within the spectrum of product management activities, faster discovery of customer needs, greater customization of the products to meet the customer needs, faster new product testing, and shorter product life cycles are issues that manufacturers are facing. One of the keys to improving the performances stated above is faster and better product design. How to shorten product design cycle time is the major question to answer.

Computer-Aided Design (CAD) systems are crucial tools for engineers in various fields, such as mechanical, electrical, software, chemical, architectural, and civil engineering. The birth of interactive CAD tools can be traced back to 1960s [3]. In the past four decades, Mechanical Computer-Aided Design (MCAD) systems have been evolving from 2-Dimensional models to 3-Dimensional models, from wire frame modeling to surface and solid modeling. Some new

techniques such as feature-based design and variational geometry have had computer play a significant role in product development. In the current highly networked business and engineering environment, network oriented collaborative design systems become a development trend for future CAD systems to support faster and better product design.

Despite the advancements in computer technology, there are still some beleaguering problems for CAD tools in terms of the efficiency and effectiveness of information exchange between human being and computer, as well as computer to computer. The exchange channels are far from engineers' expectation about CAD tools. For example, special engineering skills are required to use CAD tools for design. Currently the computer is incapable of walking the human being through the design process. Furthermore, lack of common data format causes islands of automation in Computer Integrated Manufacturing (CIM). Ease of communication between users and computers as well as among computers is the ultimate goal of the future CAD systems, which is the issue of *interoperability*.

The hub of the conventional mechanical CAD systems is the geometric modeler. The task of computers focuses on manipulation of geometric information, whereas non-geometric technical information (e.g., material properties, functional requirements, and manufacturing methods) and administrative information (e.g., bill of materials, process planning and scheduling, and cost estimation) are mostly neglected. Existing design information modeling methods impede collaborative design. First, current CAD systems have high risks of degrading integration during design. Mechanical design needs to extensively consider various issues of material properties, tool selection, tolerance, and manufacturing/assembly process, etc. Computer-Aided Drafting falls short in capturing these aspects. Second, current CAD systems do not support direct constraint imposition. The raw data of geometric shape, dimensions, features, etc. are

entered by specially trained CAD engineers. Other design partners cannot add constraints to the design and integrate the non-geometric requirements and specifications in design data. Third, no CAD tools exist to effectively aid the conceptual design and propagate specifications and data to detailed design and downstream activities. The trial-and-error approach makes the design cycle longer for new product conceptualization.

1.1 Role of Internet in Product Design

Global market calls for collaboration among designers and manufacturers. The number of multi-national companies has increased from 7000 in 1969 to 24000 in 1995 [4]. In manufacturing industries, product design and manufacturing process has been much more distributed than ever before. The business pressure toward outsourcing forces corporations to design complex products collaboratively. Ford Motor Company estimates that suppliers add 60% of a vehicle's value, and automotive companies are increasingly relying on these suppliers to participate substantively in vehicle design. Defense Advanced Research Projects Agency (DARPA) estimates that the supply chain accounts for more than 50% of weapon system and major subsystem production costs [5]. In such a geographically and temporally distributed environment, efficient and effective design collaboration should be guaranteed to maintain product quality and organizational competency.

Customers, who are the driving force of manufacturing evolution, are continuously increasing their expectations about lead-time, quality, and price. Mass customization is taking over mass production. Diversity of products requires producers' quick responses. Challenges exist in cutting costs of design and manufacturing, while retaining high quality. Currently the

cost of product design contributes to a significant proportion of its operation costs for a manufacturing company. For example, General Electric's GE90 engine for the Boeing 777 aircraft cost \$2 billion to develop. Ford spends \$3-6 billion on developing a new model automobile [6]. There is tremendous cost leverage available through improved collaborative design.

In spite of time and space restrictions, the Internet enables communication among design team members, as well as with other teams such as material procurement, manufacturing, assembly, quality control, and customer services. Customers can directly contact design personnel and participate in remote design of products. Stakeholders of supply, manufacturing, product test, maintenance, recycling, and others are able to contribute their expertise at early product design stages so as to reduce the risk of failure and shorten the design cycle time.

Specifically, there are several interoperability issues to be considered in collaborative design tools. First, collaborative design over Internet requires an industry standard for CAD data. To complete effective information exchange, a CAD data exchange standard should be established by the CAD industry. There are many CAD file formats currently used in industry, such as IGES (Initial Graphics Exchange Specification), DWG, DXF (Data eXchange Format), VDAIS (Vereinigung Deutsche Automobilindustrie IGES Subset), SET (Standard d'Exchange et de Transfert), STEP (STandard for the Exchange of Product model data), and VRML (Virtual Reality Modeling Language). These commonly used standard CAD files in industry capture only the static geometric information and part of the administrative information. Other information that contains designer's intent such as constraints and other dynamic relationships is lost during CAD file translation. The use of pure visible geometric graphics, which is supported by standard data translations, does not allow users to modify solid models that lack parameters or features,

which represent the history of modeling. As a result, teams with different CAD packages cannot work on design projects together efficiently. To exchange all useful product information, a more powerful data format should be developed to integrate various design information.

Second, the information infrastructure that supports the Internet-based product development should be established to assist the cooperation of various Computer Aided Engineering (CAE) systems. Current CAD data formats were designed for standalone systems. All information about components and assemblies has to be available locally in order to be processed. Transferring CAD information among design collaborators requires large amounts of data to be moved around, which is inefficient under the constraints and limits of communication bandwidth. Furthermore, corporations do not wish to expose complete design data to customers or suppliers for information security purpose. A collaborative design data model should support lean information processing. It should be compliant with industry standards of programming, communication, networking, system management, and interfaces between applications and system services. It should also have good compatibility and interoperability with current CAE systems.

In this dissertation, a new scheme for capturing design information within the context of the Internet services and transactions is developed. To maximize the future CAD systems' openness, flexibility and integrity with the Internet, this data scheme intends to be portable across different Internet protocols, network configurations and operating systems. The performance and throughput of collaborative design systems could vary based on the requirements of application. This data scheme has a distributed style, which supports the required scalability and extensibility of the systems.

1.2 Importance of Capturing Knowledge in Design

In modern society, skill specialization creates domain experts, which makes the design process less smooth than it was before. Designers are by no means merely exchanging graphical and physical shapes, but are exchanging knowledge about design and design process, including specifications, design rules, constraints, rationale, etc. As design becomes increasingly knowledge-intensive and collaborative, the need for intelligent computational design tools to support the representation, use, and integration of knowledge among distributed designers becomes more critical. Design data should contain the knowledge that is used and generated in a design project. It is essential to ensure that one can represent and reason with what is captured in design.

Knowledge is defined as the fact or condition of knowing something with familiarity gained through experience or association as quoted by the Merriam-Webster's dictionary. Information is the valuable data from the subject's point of view and knowledge is organized information. How to represent engineering design data and knowledge is one of the important topics to address.

In general, design data includes [7]: (1) Product data, which covers the requirement specification, functional diagram, sketches/drawings, calculations, graphs, etc., as well as production plans, user manuals, maintenance instructions, etc. in the entire product life cycle; (2) Process data, which includes the rationale behind product data such as the information to support arguments and decisions related to the various stages of the product and alternatives, along with various aspects of the business involved; (3) Process administration data, which includes the planned and actually applied resources (who did what, when and how). Ideally, design knowledge should be embedded in design data for storage, transfer, and reuse.

Design knowledge covers a variety of mental powers, including laws, rules and formulas pertaining to the function and behavior of human, material, object and physical space. Its broad spectrum includes general laws such as Newton's laws of motion and Hook's law, specific rules such as spatial relations for assembly, human related ergonomic issues, process and environment related material properties, cost related durability and reliability, etc.

Though design knowledge is important, the current standard CAD file formats do not capture it well. For example, STEP is only capable of modeling geometric and topological elements, tolerance, a small portion of features, and administrative information. The dynamic constraints concerning parameters, engineering relationships, functionality, etc. cannot be represented, which hinders design knowledge transferring interoperably.

This dissertation focuses on the representation and manipulation of dynamic design information, which includes product data and design rules that are used to capture the variant information besides the static one. To a large extent, this type of information is added into design data in terms of internal or external constraints. The data and knowledge in electronic format should be recognizable to different parties within the computer supported collaborative design environment for a successful design.

1.3 Research Objectives and Overview

Within the context of Internet-based collaborative design, there are special requirements for interoperable design information representation. Information incompleteness, improcessability, and inconsistency are major problems to solve. There are needs for representing more design knowledge in CAD data, transferring selective design information among design collaborators,

carrying design information in a network oriented data scheme, supporting consistent interpretation for different systems, modeling uncertainty and inexactness of design, and enabling multidisciplinary constraint imposition and integration.

The research objective of this dissertation is to develop a design information model, Universal Linkage (UL) model, to tackle issues related to interoperability for collaborative design systems. Research issues include collaborative design data scheme, lean design information modeling, Dual-Rep design feature representation, geometric and non-geometric constraints integration, semantic naming and linkage, and Interval Geometric Modeling.

(1) *Collaborative design data scheme*: An UL-PML scheme is developed to capture geometric and non-geometric entities and relations among them. Unlike current CAD neutral formats and models, the UL model is able to capture not only static geometric information, but more importantly design constraints which reflect the dynamic relations among geometric entities, thereby more design knowledge and rationale. This model captures both static and dynamic relations among entities. Static relations represent structural and topological associations and dynamic relations are constraints defined by designers. Graphically, Directed Hyper Graph (DHG) symbolizes UL model. Computationally, Product Markup Language (PML) [8] represents this model. PML has the format of Extensible Markup Language (XML) [9], which is an emerging Internet information transferring standard. PML inherits XML's good extensibility, flexibility and portability. This research focuses on the feasibility of building information interoperability (PML) based upon data interoperability (XML). It includes a new scheme for design knowledge and Internet data exchange integration, PML semantics and schema in the mechanical design domain, and extensible representation of geometric and non-geometric constraints.

(2) *Lean design information modeling*: To enable selective design information exchange and sharing, a linkage reference structure is developed in the UL model that allows physically distributed entities to be linked to build a logically integrated set of design information. Relations of basic entities can be established across the boundary of data files, which overcomes the shortcoming of current standalone CAD file formats in information transferring. This allows design collaborators to share design information without transmitting large amounts of raw data, thereby supporting intelligent information sharing. This introduces a new way of distributed design data modeling, storage, and query with entity-level granularity.

(3) *Dual-Rep design feature representation*: To support implicit modeling and to enhance the existing design feature representation methods, a Dual-Rep feature representation method is developed in the UL model. This method models intentional and geometric feature independently for both global and local features such that feature construction and evaluation are both modeled.

(4) *Geometric and non-geometric constraints integration*: To capture more design intent, constraints are modeled in the UL-PML scheme as dynamic relations in an extensible form. Symbolic constraints are represented in descriptive ways, which eliminate ambiguity and uncertainty. Numerical constraints are represented by interval values, which reduce inconsistency due to numerical errors. From both symbolic and numerical aspects, CAD models' completeness, reliability, and robustness are improved.

(5) *Semantic naming and linkage*: To maintain persistent reference and linkage among entities, a geometry-based semantic ID method is developed such that topological entities are identified by geometry and geometric entities are named based on surfaces. Hierarchical namespace is introduced to reduce the interference between IDs. This method adds semantics of geometry and

topology into IDs thereby increasing the stableness of entity reference. It builds the identification framework for the distributed UL model, and can enhance the naming persistency of current CAD systems.

(6) *Interval Geometric Modeling (IGM)*: A new geometric modeling scheme based on interval representation and analysis is developed to improve model's robustness and represent design uncertainty and inexactness. IGM allows all parameters of geometric modeling (coordinates as well as parametric constraints) to be non-trivial-width interval values instead of fixed values. Interval numerical constraints then can be used for the process of conceptual design, detailed design, and design optimization. It models soft constraints, thus reducing the chance of conflicts during constraint imposition. It releases the restriction of under-constrained and over-constrained issues for variational geometry. Constraint-driven interval geometric modeling supports more design interaction for optimization and decision-making. IGM establishes a generic approach for interoperable numerical constraint representation and integration for the entire design cycle.

In this dissertation, Chapter 2 presents current different knowledge representation schemes and data models for design. Based on the special requirements for design data models, Chapter 3 describes the new UL model and DHG representation. Chapter 4 describes the basic syntax and semantics of PML, and the schema of PML in the context of mechanical design is defined. This includes geometric and non-geometric entities and the relations among them. Chapter 5 describes how features can be represented in the UL-PML scheme. Representation issues of intentional features and geometric features, symbolic and numerical constraints, parametric families, and naming persistency are discussed. Chapter 6 presents the IGM for numerical constraint model. Chapter 7 shows the implementation and proof of the new concepts and the UL model.

2.0 DESIGN KNOWLEDGE REPRESENTATION FOR CAD

Functional, material, manufacturing, maintenance, and other information about a product need to be transferred among design stakeholders. An integrated product model is vital in network-based collaborative design. Current CAD systems use different data structures and file formats. Although some standard neutral formats have been developed to support file translation, they cannot capture all original product-related information. Most of them only support static geometric information, that is, the physical shape of a product. However, product design cannot be completely captured by its geometric data. More importantly, design intent including functionality, cost, materials, tolerances, etc. determines the actual shape of the product. From this viewpoint, design is a decision-making process based on the designer's knowledge. Sometimes it is practical to postpone a decision to a later stage of the design and planning process [10].

Currently it is common in a collaborative design environment that members of a design group use different kinds of modeling systems. Different design tools are used for different stages of the design. Input and output information have several formats. To allow efficient communication and collaboration, these pieces of design information should be logically integrated and consistently represented for different CAD systems. Current CAD data formats were designed for standalone CAD systems. Transferring a large amount of data by network communication channels with limited bandwidth is inefficient and the quality of service for remote geometric computation and manipulation cannot be guaranteed.

Design is a process of knowledge reuse and generation for the designer. The designer's intent is the reflection of his or her design knowledge. Attaching detailed product information besides geometry and topology is essential for sound CAD model with explicit design knowledge. The subsystem of Knowledge Representation (KR) for design is crucial in the integrated design system. The responsibility of KR system is to select appropriate symbolic structures to represent knowledge, and to select appropriate reasoning mechanisms both to answer questions and to assimilate new information, in accordance with the truth theory of the underlying representation language [11].

Generally there are two kinds of CAD modeling systems. One of them is explicit modeling, in which only static geometric information is recorded at any time during the modeling process. The other is procedural or implicit modeling, wherein the product is modeled in a sequence of instructions, and the history of construction is embodied in the CAD file. Implicit modeling requires less geometric computation involvement of human users and more design process information than explicit modeling. Most of the CAD tools have migrated from explicit modeling to implicit modeling. A good CAD data model should support implicit modeling and capture design process information as much as possible.

2.1 Current Standard Formats

There are different commonly used formats for CAD models. To attain the objective of product data sharing on different platforms, standard CAD file formats are required. Various industries have embraced the effective implementation of the Standard for the Exchange of Product Model Data (STEP) to achieve this objective. CAD file standardization was initiated in

1979 by an industrial group led by Boeing, General Electric, and the National Bureau of Standards (now the National Institute of Standards and Technology (NIST)). This work resulted in the Initial Graphics Exchanges Specification (IGES) version 1 and was adopted by the American National Standards Institute (ANSI) in 1981.

2.1.1 The Initial Graphics Exchanges Specification (IGES)

IGES is the precursor of product data exchange standards, similar to the French standard SET and the German VDAFS for automobile surface data exchange. It is a U.S. ANSI standard whose purpose is simply to exchange flat-file-structured CAD data between systems. IGES is executed in a batch-like operation. It was developed using a bottom-up approach with a goal of addressing as many entities as possible. That is, the format for elements (geometry, attributes, etc.) was defined first, with an application for the data in mind. Software developers attempt to match their own internal data element representations based on their interpretation of the IGES data element specification. Users often face difficulties when these interpretations are not accurate or an entity definition is ambiguous; therefore, conformance to IGES is sometimes subjective [12].

Recognizing these limits, the U.S. IGES group initiated a project in 1984 called Product Data Exchange Specification (PDES) to rectify the problems with IGES. International Standards Organization (ISO) later embraced PDES as the basis for its international standard (ISO 10303), which is commonly known as STEP.

2.1.2 The Standard for the Exchange of Product Model Data (STEP)

The objective of STEP is to provide a neutral mechanism capable of describing product data throughout the life cycle of a product, independent of any particular system. This kind of

system makes it suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases and archiving.

The STEP (ISO 10303) parts can be grouped into the following five main categories: description methods, implementation and conformance methodology, integrated-information resources, abstract-test suites, and application protocols (AP): (1) The description methods group forms the underpinning of the STEP standard. This includes overview, which contains definitions that are universal to the STEP, and EXPRESS language, which is used to describe data modes; (2) The implementation methods group describes the mapping from STEP formal specifications to a representation used to implement STEP. The conformance testing methodology framework provides information on methods to test software product conformance to the standard. It also acts as guidance for creating abstract-test suites, and describes the responsibilities of testing laboratories; (3) The integrated information resources group contains the generic-STEP-data models. These data models can be considered the building blocks for STEP, and they can help AP integration and interoperability; (4) The abstract-test suits consist of test data and criteria that are used to assess the conformance of a STEP software product; (5) The application protocols describe the more detailed and complex data models for specific product applications. They not only describe what data is to be used, but also describe how the data is to be used in the model.

In STEP applications, resource models, application protocols, and EXPRESS information modeling language are to be implemented. The resource models contain the low-level entities and features, such as geometry, topology, form features, product structure configuration management, and tolerances. The application protocols describe the scope and information requirements for a particular application of STEP, usually by commodity (such as machined

parts, sheet metal, castings, composites, etc.). The APs break STEP into more manageable and comprehensible "chunks" that can be more readily implemented within a computer environment. AP development and implementation is a major distinguishing feature between IGES and STEP. EXPRESS, a computer-interpretable data definition language, is built based on the Entity Relationship (ER) model, which contains the relationships of generalization and specialization.

Though STEP is becoming standard in industries, it still cannot capture parametric and variational information [13]. This kind of relationship information among geometric entities is an important part of design constraints. To fully represent design data, current information models for CAD should be expanded so as to contain more relations of design entities.

2.2 General Data Models

The objective of data and information models is to describe a Universe of Discourse (UoD) in certain ways that the information of the UoD can be transferred. The task of information modeling is to provide a sound basis for mapping between the portion of the world of interest and a representation of it that can be used as a specification for defining a database and/or application. Various product information models have been proposed and some have been used in industry. For example, the ER model [14] and its extended version - Enhanced Entity Relationship (EER) model [15] are the basic data models in relational database systems. ER/EER views the world as consisting of entities with attributes and relationships among them, including association, specialization, generalization, inheritance, and categories.

Integration DEFinition for Information Modeling (IDEF1X) is used to produce a logical graphical information model, which represents the structure and semantics of information within

an environment or system. IDEF1 was originally developed under the Integrated Computer Aided Manufacturing (ICAM) program by Hughes Aircraft and D Appleton Company [16, 17], built upon relational theory and entity-relationship modeling concepts. IDEF1X is the extended version. Similar to the EER model, the relationships in IDEF1X include connection (association), categorization, etc. But IDEF1X has more structural constraints to embed semantics.

Nijssen's Information Analysis Method (NIAM) [18] is a binary-relationship approach, based on the concept of information exchange between the user and the computer, using elementary sentences (conceptual grammar). In NIAM, object and role correspond to entity and relationship in ER. It attempts to build the semantics of the object into the syntax of the data structure. Restricting rules such as uniqueness constraint, total constraint, equality constraint, exclusion constraint, and subset constraint, are applied on objects.

The information models of EER, IDEF1X, and NIAM emphasize structural relationships, thus connections of entities can be built. The restriction of these models for applications in CAD systems is that the structural relations and constraints of these models are invariant [19], whereas variant relations among geometric entities in CAD are widely applied to represent design constraints. Therefore an information model, which accommodates variant relationship among geometric entities, is needed to enable smooth interaction between CAD systems.

To find an appropriate way to model design data, we need to ruminate the nature of design. Design is an information-processing activity that creates a description of an engineered artifact, guided by some set of specifications and some set of constraints [20]. It is an intelligent process of old knowledge application and new knowledge generation. The behavior of design performed by a design engineer is essentially based on his/her knowledge. The sketches or drawings

represent the design knowledge of the designer, which are constrained by design rationales. The design data is the knowledge about the product, which is represented in a computer comprehensible format. Design is a knowledge intensive activity.

2.3 Design Knowledge Representation

Knowledge can be divided into two categories: declarative knowledge and procedural knowledge. Declarative knowledge is knowing something is the case. Such knowledge is generally a matter of knowing facts, or laws, or terminology peculiar to the subject. Knowledge about tasks, on the other hand, is often more procedural in character; that is, is knowing how to do something [21].

The notion of the *representation of knowledge* is at heart an easy one to understand. It simply has to do with writing down, in some language or communicative medium, description or pictures that correspond in some salient way to the world or a state of the world [22]. Under the assumption of *knowledge representation hypothesis* [23], any process capable of reasoning intelligently about the world must consist in part of a field of structures, of a roughly linguistic sort, which in some fashion represents knowledge and beliefs that process may be said to possess. Moreover, these structural ingredients, independent of what external observers take them to be, play an essential and causal role in engendering the behavior that shows the knowledge. Any system, whether it be human or artificial, that manifests intelligent behavior, is assumed to contain a substructure of knowledge base that encodes knowledge, and another substructure of inference engine that manipulates the knowledge. Thus, one can presume from

the hypothesis that any KR language contains two aspects, namely syntactic and inferential aspects.

There are many research efforts on knowledge representation and interchange in the area of Artificial Intelligence (AI). Generally speaking, there are five approaches in KR, that is, Logics, Production Rules, Semantic Networks, Frames, and Artificial Neural Networks [24]. Because of its declarative nature, a logical language has the advantage of natural semantics, expressive power, economy of storage, generality, flexibility, and maintainability. But it has the disadvantage on computational inefficiency, undecidability, default reasoning, and abduction. Production rules have been used extensively in expert systems [25]. It has the similar pros and cons as logic-based representation languages. In the above two kinds of representation, knowledge is organized around relatively simple and independent elements (propositions in logics and facts & rules in production rules). Different pieces of knowledge are stored independently of each other with no strong interconnections between them. This is against the intuition that information in human memory is highly interconnected. Though they have attractive property of good expressiveness, computational untractability adds shadows on the application prospect. Comparatively, semantic networks and frame-based representation languages emphasize more on the structures of knowledge. The semantic network was initially created to represent the semantics of English words [26]. Then It was used to represent knowledge, including all sorts of non-semantic things (e.g., propositions, physical object structure) [27,28]. Knowledge is expressed in terms of objects and the relationship among them, graphically corresponding to nodes and arcs. Object-centered frame-based representation languages [29, 30] organize knowledge in a more structured fashion for the chunks of knowledge than it is in logic. At the same time, the declarative and procedural aspects of a given chunk are

tightly connected. Hierarchically, *class-frame* and *instance-frame* build the structure of knowledge. Inheritance is one of the major relations among objects. The frame is the predecessor of the object in the object-oriented concept, which now has been widely used.

In AI field, KR has been studied for decades. Most of the researches consider how to model and represent general knowledge rather than certain specific areas, In the next section, different languages for general knowledge representation are introduced.

2.3.1 General Knowledge Representation Languages

KL-ONE [31,32] is based on semantic networks formalism. The primitive semantic network was unable to distinguish assertional information and definitional information [33]. The graphs in semantic networks were open to many possible interpretations. Beginning with the KL-ONE, description logic (also called terminological logic, taxonomic logic, frame-based [description] language, concept language, term subsumption language, KL-ONE-like language, and structured inheritance networks) required a precise syntax and semantics for the representation language. Assertions are made relative to a *context*, and they therefore do not affect the concept structure. In addition, KL-ONE distinguishes two types of concepts, *generic* and *individual* concepts. Generic concepts are descriptions of classes of individuals, whereas individual concepts are descriptions of individual objects, attributes, relationships etc. From this aspect, KL-ONE is similar to the hierarchical frame representations.

KRYPTON [34] is a mixed representation system which grew out of KL-ONE. It uses a network/frame-style language for forming terms and a first-order predicate language for making statements. Thus, KRYPTON separates definitional and assertional information by splitting the operations into two components: a terminological one (TBox) and an assertional one (ABox).

CLASSIC [35] is a description logic with an ancestry of extensive theoretical work tracing back over to KL-ONE. It was intended to be built with a compact logic with a variety of inferences, which are completion inferences, contradiction detection, classification, subsumption, and rule application. CLASSIC can be envisioned as a deductive, object-oriented database system. It has been implemented to aid conceptual modeling for configuration of telecommunication equipment [36, 37].

In KRL [30], the formalism for declarative knowledge is based on structured conceptual objects with associated descriptions. It was an attempt to integrate procedural knowledge with a broad base of declarative forms. The control structure is based on multiprocessing with explicit (user-provided) scheduling and resource control. The system is so complex that it finally collapsed.

KODIAK [38] is a hybrid language of frames and semantic networks. Like KL-ONE, the primary structure of KODIAK is the *concept*. However, there is no notion of role, slot, or case. Instead, the idea of having a slot or role is replaced by a primitive epistemological relation --- *manifest*.

RML-Telos family [39, 40, 41] includes an object-centered framework, which supports aggregation generalization, and classification; a novel treatment of attributes; an explicit representation of time; and facilities for specifying integrity constraints and deductive rules.

Due to domain and community dependency of knowledge, researches on knowledge interchange are being conducted for knowledge sharing and reuse. Knowledge Interchange Format (KIF) is a computer-oriented language for the interchange of knowledge among disparate programs [42]. Ontolingua [43] and Knowledge Query and Manipulation Language (KQML) [44] are developed for agent-based knowledge sharing and communication.

EDDL+TDDL [45, 46, 47] is a framework for modeling and analyzing domain knowledge at a conceptual abstraction level. Within this framework, domains are modeled using two different representation levels, namely an epistemological one and a terminological one. The epistemological level defines an external, user-oriented and domain-dependent representation based on the EDDL language. The terminological level defines an internal, machine-oriented and domain-independent representation based on the TDDL language, which is a decidable member of the KL-ONE family. The two levels are linked by protoDL.

The problem of the representation schemes above is that they are designed for general knowledge. In mechanical design, design knowledge is applied in the design process. Physically, this knowledge is embedded in the process of the design and the design data. Design knowledge representation requires a special format of hierarchical structure. It should be able to model objects and relations, object properties, classes and instances, etc. [48]. Thus, a dedicated representation mechanism is needed to model design knowledge and design data so as to represent the information occurred in design efficiently and effectively.

2.3.2 Design Modeling Languages

Research on modeling languages for design has been carried out for years. The purpose is to represent enough design knowledge in a computer-comprehensible way so that the knowledge can be retrieved by computer and reapplied to new design, thereby leading to intelligent and easy-to-use CAD tools.

IDDL [49, 50, 51, 52] is a hybrid language of predicates, frames and production rules. It has the concepts of entities, relationships among entities, and attributes of entities and relationships. These are represented by objects, first-order predicates, and functions, respectively. Objects are denoted as constants and variables of first-order predicate logic.

Predicates are used to express logical relationships among entities, and they construct the if-then rule paradigm. A function can be defined over a set of both objects and predicates. Calling a function corresponds to sending a message to a set of objects in the object-oriented programming paradigm.

EDM [19, 53] is developed based on sets and predicate logic. It has three base forms: domains (sets of values), aggregations (sets of named domains, e.g., variables) and constraints (general relations that are defined as procedures). All constraints are fully specified and executable.

DKSL [54] is implemented using a frame-based KR scheme. In addition to the features of conventional frame systems, it supports the notion of context as a "dictionary" mapping from terms to frames. Contexts may be created by the user, and may be nested. A System Context contains basic definitions, and a User Context stores user-defined frames. There are no explicit classes or "meta-frames" in DKSL. Rather, a prototype-based approach is used, wherein any entity can be an exemplar with which other frames can be cloned. Without classes, inheritance mechanism is done by clones.

CML [55] is a general-purpose declarative modeling language for representing physical knowledge required for compositional modeling, which formulates a behavior model of a physical system by composing descriptions of symbolic and mathematical properties of individual components for early-stage design. It is translatable to the KIF [42].

The above languages inherit the AI approaches of KR. Though they have good expressiveness of logic relations for general design knowledge, they still have limitations on representing geometric and spatial relations among entities. And most importantly they lack the ability to keep the relations persistent so that they can be transferred among CAD systems.

There are two types of relations among geometric entities to be captured. One is the *static relation* that exhibits the basic structural or topological information of entities, such as the aggregation relation between a line and its two end points. Another is the *dynamic relation* that is added by the designer as constraint, such as the distance between two points, or concentricity of two holes. The dynamic relations can be changed without altering the topological information of a part or an assembly. The mechanical design activity deals with both static and dynamic relations at the same time. Design is the process of problem solving subject to various dynamic constraints. Parametric design is an improvement of the CAD with dynamic constraints. But the lack of interoperability for dynamic constraints among different 3D CAD packages reduces the power of parametric design.

2.3.3 Requirements for Design Information and Constraint Representation

Spooner [56] has a list of requirements for object-oriented CAD data models. Data must be modeled as objects organized into aggregation and generalization hierarchies. The data model must support definition of object intentions as well as extensions. It must be possible to define properties of objects. The data model must allow definition of operations (methods) for objects. The data model must support inheritance of properties and operations. It must be possible to represent relationships between objects. The data model must allow the intentions and extensions of objects to be modified (dynamic schemas). It should possess the properties of support for strong typing in the data model, full support for recursive object structures, equivalent support for aggregation and generalization, efficient and flexible update capabilities for objects, multiple inheritance, support for methods and procedures, and specification and enforcement of data integrity constraints.

Eastman and Fereshetian [19] proposed criteria to evaluate product models in CAD/CAM development. Good models should provide full abstract data types that include object behaviors, the ability for modeling multiple specialization, composite objects, relations within compositions, relations on object structure, relations between variables, variant relations needed for schema evolution and state of integrity. The models should also support integrity management of external applications needed for applications management, management of partial integrity needed for iterative design, and schema evolution needed for design evolution and refinement.

From the viewpoint of interoperability, the ideal representation language for mechanical design should have the following properties. It is declarative in nature and self-explanatory. It should be able to capture the inherent properties and relations among objects explicitly. Those relations include functional, structural, and performance relations, as well as parametric, engineering and other constraints. Properties and relations should maintain good persistency during information transferring. The language should be semantically comprehensive. The engineering meaning of design can be clearly uttered. The language should be both modularly self-contained and flexible so that various objects and their relations with partial integrity can be captured, stored, and queried in an arbitrary manner. Additionally, the language should be extensible. When new entities and relations are needed, it should be able to be extended. At last, to encourage openness, this language should also be simple enough and comprehensible to both humans and machines.

In the UoD of mechanical design, design knowledge mostly appears as constraints during the design. Design for manufacturability, assemblability, profitability, quality, safety, and recyclability, etc. (DFX) essentially are domain knowledge practice at the early design stage so

as to reduce the time of product development. Within these domains, feature, functionality, manufacturability, constraint, etc. are information elements, which consist of the domain model of design. The information flow within a design team largely depends on the attachment of different constraints. The design constraint is the adhesive in the whole design process. It models the dynamic relations among geometric entities and captures designer's intent explicitly. Constraint representation is one of the most important aspects for design knowledge and information models. Constraints consist of a wide spectrum of domains, including geometric and topological relations among geometric objects and features, spatial relations among assembled parts, restrictions on configurations because of manufacturability, assemblability, material characteristics, ergonomics, reliability, etc.

Parametric design is an improvement of CAD with features and dynamic geometric constraints. Geometric constraints are internally represented by different schemes (mathematical equations, predictive logic, graphs, etc.), which capture dimensions and dependencies of geometric entities and features. The physical shape of an object is determined by the results of constraint solving. But the different internal constraint representations of parametric CAD systems are not easily interchangeable. Besides geometric constraints, constraints concerning other engineering issues such as material, tolerance, manufacturing, safety, and reliability cannot be captured by these CAD systems.

Therefore, a more open model for design information is needed which will effectively and thoroughly represent product data and design constraints. It should be able to model geometric objects as well as dynamic constraints defined by designers such that all relevant product information can be carried and exchanged seamlessly. Hence a Universal Linkage model is

developed for this purpose to model geometric and non-geometric entities and constraints explicitly.

3.0 UNIVERSAL LINKAGE MODEL

As mentioned in Chapter 2, various relations are important in recording design information. Besides the hierarchical structure of the geometric entities, which represent static relations, dynamic relations among entities are important as well. This chapter will describe a Universal Linkage (UL) model that captures both static and dynamic relations. Graphically this model can be presented by Directed Hyper Graphs (DHG). In Chapter 4, a textual presentation of UL model in Product Markup Language (PML) is specified in detail.

To build an information model for CAD, three fundamental questions should be answered. (1) What kind of information elements are to be captured? (2) How would these elements be represented? (3) How can information be retrieved from these elements? These three questions are dealing with information abstraction, representation, and deduction. These three aspects comprise the information structure of CAD systems.

Pure relational approach abstracts information objects in a structured manner, thus information can be easily retrieved and modified using external operations. Object-oriented approach categorizes information objects in a modular way, such that objects are self-contained micro-systems whereas connections among objects are simplified. Object-oriented models can be descriptive object-oriented in which only structures and relations of entities are captured, or procedural object-oriented in which both objects' structures and behaviors are modeled. Besides modeling declarative knowledge similar to relational approach, object-oriented approach can

also model procedural knowledge, in which information abstraction, representation, and deduction are integrated.

Structurally procedural object-oriented models are much more complicated than declarative ones. It is also difficult to achieve high portability and openness for procedural object-oriented models. The new UL model does not take the procedural approach in order to ensure good interoperability. To make the model simple but comprehensive enough, the UL model adopts a hybrid approach of declarative object-oriented and relational modeling.

3.1 Information Elements of UL Model

The UL model has the fundamental elements of *entities* and *relations*. Entities are abstract representation of any objects in the real world. They include geometric entities, topological entities, entities of material, tolerance, mathematics etc. Examples are shown in Table 1.

Table 1: Examples of Entities

Geometric Entities	Non-Geometric Entities			
	Topology	Material	Reliability	Manufacturing
Point	Vertex	Density	MTBF	Cutting speed
Vector	Edge	Polythene	Hazard Rate	Feed rate
Line	CoEdge	Yield Strength	Safety Factor	
Curve	Face	Stress	S-N Ratio	
Plane	Shell	Friction Coef		
Sphere	Body	Specific Heat		

The information elements used in UL model are defined as follows.

Definition 3.1: An *entity* is an object that exists as a distinguishable unit in the Universe of Discourse for design. It possesses unique attributes.

Definition 3.2: An *attribute* is a characteristic property or feature that is associated with an entity and identifies or modifies the entity.

Definition 3.3: A *relation* is a logical or natural association between two or more entities.

The relations among entities are categorized into two types: *static* relations and *dynamic* relations. Static relations are basically the structural relations among entities within a part or among different assembled parts. They represent static geometric and topological relations. Static relations include aggregation, which transforms a relationship between objects into a higher-level object [57, 58], and generalization, which refers to an abstraction in which a set of similar objects is regarded as a generic object [59]. In CAD information models, geometry-related relations mostly are aggregation relations while non-geometric (e.g., administration, material) relations include both of aggregation and generalization. Dynamic relations are specified operationally by designers, which appear as various kinds of constraints. Examples of relations are shown in Table 2.

Table 2: Examples of Relations

Static Relations	Dynamic Relations
<i>Consist-of</i>	<i>Distance between two planes</i>
<i>is-a-kind-of</i>	<i>Parallel</i>
<i>associated-with</i>	<i>Angle between two lines</i>

Unlike ER model, which only captures static relations, the UL model differentiates static and dynamic relations because dynamic relations are crucial for constraint representation.

3.2 Directed Hyper Graph

Graphically, the UL model can be represented by Directed Hyper Graph (DHG), in which a *node* denotes an entity and an *arc* stands for a relation. There are two categories of entities in UL model, geometric entities and non-geometric entities.

Definition 3.4: A *geometric entity* is an entity that is geometrically perceptible and can form a concrete shape in a 3-dimensional Euclidean space. It is represented by an elliptical node in DHG, as shown in Figure 1-a.

Definition 3.5: A *non-geometric entity* is an entity that is not geometric and not geometrically tangible in a 3-dimensional Euclidean space. It is represented by a rectangular node in DHG, as shown in Figure 1-b.

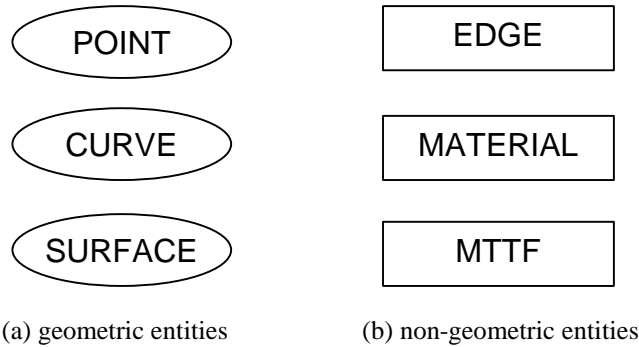


Figure 1: Geometric entities and non-geometric entities in DHG

There are two categories of relations in the UL model, static relations and dynamic relations.

Definition 3.6: A *static relation* is a relation that indicates the essential and inherent affiliation of entities in order to form a physical object. It is represented by an arc with solid line in DHG. Three types of static relations are aggregation, generalization, and general association.

To distinguish aggregation and generalization from general association, the start head of arc is a diamond for aggregation relation and is a triangle for generalization relation, which is illustrated in Figure 2-a.

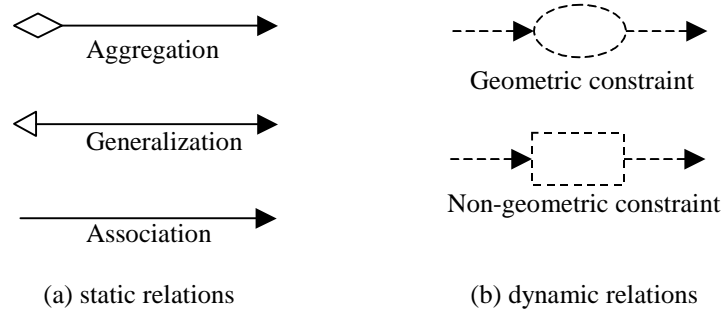


Figure 2: Static relations and dynamic relations in DHG

Definition 3.7: A *dynamic relation* is a relation that specifies the extrinsic affiliation among entities that indicates additional connection or preference. It is represented by an arc with a dash line in DHG. Two types of dynamic relations are the geometric dynamic relation and the non-geometric dynamic relations.

Definition 3.8: A *constraint* is a relation of dependency, limitation, or restriction among entities, which reflects a special requirement from designer.

Dynamic relations are constraints added externally by design participants. We use the terms dynamic relation and constraint interchangeably. To differentiate two types of constraints, a special kind of entities are defined as *constraint entities*. A constraint entity is drawn in dash line and attached on the corresponding constraint arc graphically in DHG, shown as in Figure 2-b.

Definition 3.9: An *initial entity* of a relation is the starting (source) entity of the directed arc corresponding to the relation.

Definition 3.10: A *terminal entity* of a relation is the ending (destination) entity of the directed arc corresponding to the relation.

Definition 3.11: A *constraint entity* is a special entity which indicates the type and characteristics of a dynamic relation.

The entities and relations have the following properties:

- (1) All types of relations are antireflexive.
- (2) Aggregation and generalization relations have transitive properties.
- (3) The direction of an arc implies the asymmetric unitary relation. If an arc has both ends arrowed, the relation has the symmetric binary property.
- (4) A constraint entity can be associated with one, two, or more entities, that is, a relation of constraint can be specified to one or more objects.

Figure 3 shows an example of aggregation static relation in DHG. *line0* is an instance of a *LINE*. It consists of two points, *point0* and *point1*, i.e., *line0* is referring to two points.

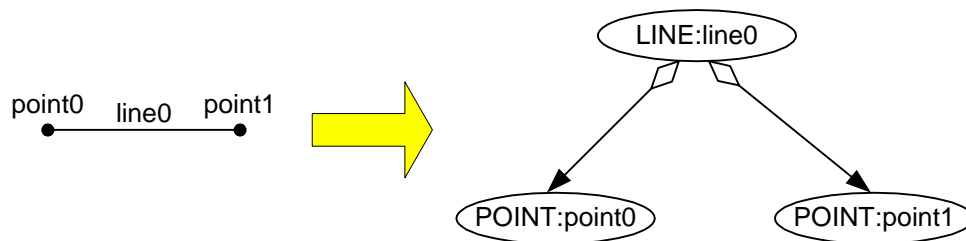


Figure 3: An example of aggregation relation in DHG

Figure 4 shows an example of general association relation in DHG. *vertex0* is a topological entity and is referring to a geometric entity *point0*.

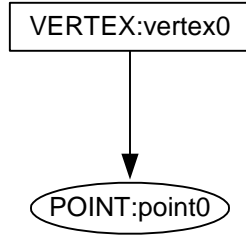


Figure 4: An example of association relation in DHG

Figure 5 shows an example of generalization static relation in DHG. *SURFACE* is the general entity of *PLANE*, or *PLANE* is a special kind of *SURFACE*. Unlike aggregation and association, generalization is mostly used in the meta-level of product modeling. It defines the relation between two abstract classes.

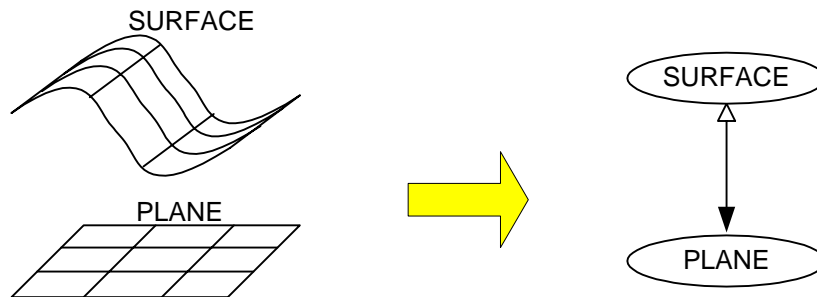


Figure 5: An example of generalization relation in DHG

Figure 6 shows an example of geometric dynamic relation in DHG. *line2* is parallel to *line1*, and the distance from *line2* to *line1* is d . Here *line1* is the terminal entity. The directions of constraint arcs are unitary.

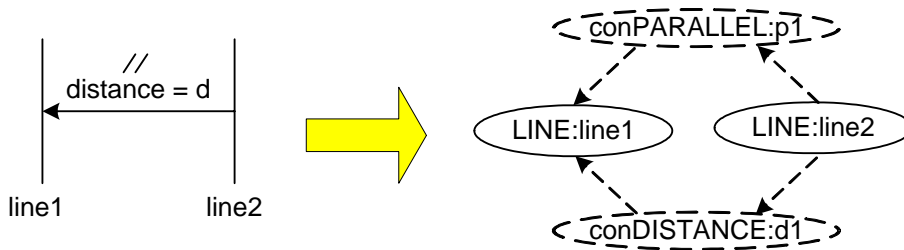


Figure 6: An example of geometric constraint in DHG

Figure 7 shows an example of non-geometric dynamic relation in DHG. The material of *part1* is aluminum. It is represented by a material constraint entity that is referring to the part.

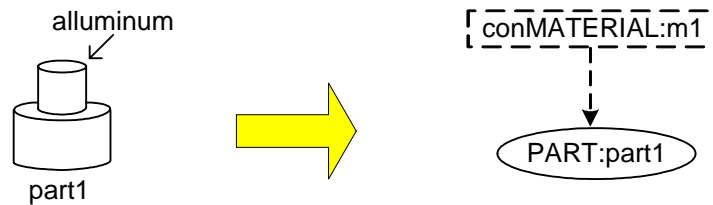


Figure 7: An example of non-geometric constraint in DHG

As a comprehensive example, Figure 8 shows a triangular sheet metal part with dimensional constraints. Its geometric and topological information as well as constraints can be modeled in DHG as in Figure 9.

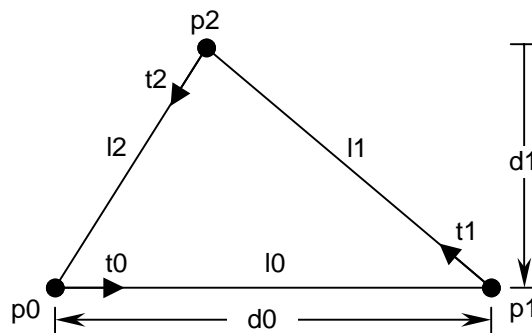


Figure 8: A triangle with dimensional constraints

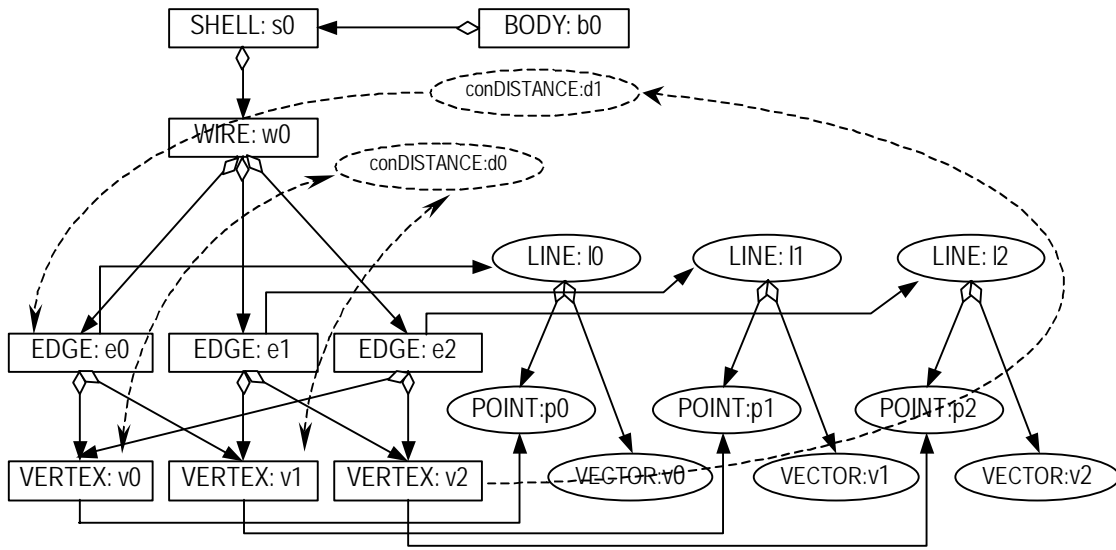


Figure 9: DHG representation of the triangle with dimensional constraints in Figure 8

3.3 Universal Linkage among Entities

Current CAD file formats were designed for standalone computers by which all design information about one part/assembly is stored in one file. Thus, they lack flexibility on design information retrieving and reuse. They do not support partial data queries. If only part of design data is needed, it cannot be retrieved without querying the whole file. Thus, fractions of design data cannot be reused unless the whole CAD file is imported. In a collaborative design environment, the design tasks of different parts or sections are usually completed by different working groups. To enable the seamless composition of product from different groups, new modeling technique is needed to support the integration of distributed design information.

Besides differentiating the static and dynamic relations among entities, another key feature of UL model is that the relations among entities are not restricted within one data file. The relations of entities located in different files and domains can be created as well. Relations are linkages among information elements. A simple linkage model allows physically distributed entities to be linked, and a logically integrated set of design information thus can be built. This feature solves the flexibility problem of current CAD data modeling for collaborative design.

In the UL model, the relation among entities can be extended across file boundaries so as to increase flexibility and modularity of CAD models. Universal links of entities may be built to support distributed CAD data. This model will take advantages of the Internet connection and assist collaborative design in a distributed design environment. As illustrated in Figure 10, relations of entities (both static and dynamic) in different domains and physical locations can be linked together. One can easily refer to entities in other data files, either located on the same computational machine or other locations through the Internet.

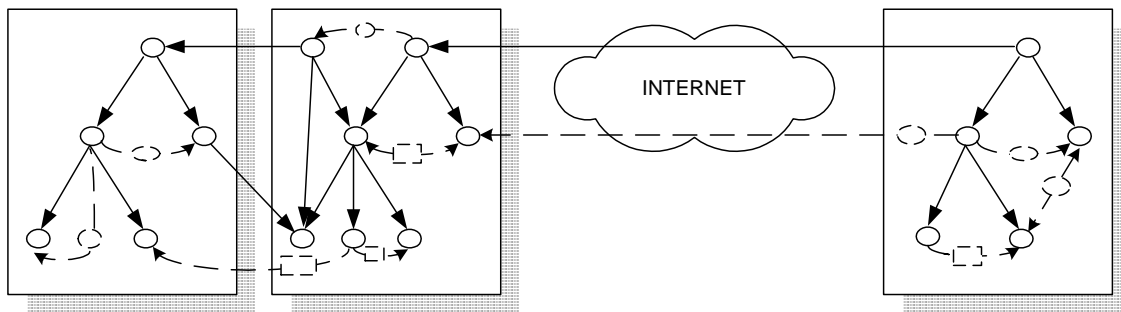


Figure 10: Universal linkage between files

Graphically, the UL model can be illustrated by DHG. Textually, UL models are represented in Product Markup Language (PML) and processed by computers. Chapter 4 describes the syntax and semantics of PML.

4.0 SYNTAX AND SEMANTICS OF PRODUCT MARKUP LANGUAGE

To encourage future information flow and CAD application over the Internet, a system independent data format is vital. It will be advantageous if this ideal format is network-oriented at the implementation level, i.e., compatible to the Internet protocols and standards. At the semantic level, this format should be object-oriented, which extensively supports data abstraction in a well-developed style that itself evolved from the frame representation of knowledge. This format should also be able to model and represent geometric and non-geometric constraints explicitly in comparison to the existing format.

With the emergence of Extensible Markup Language (XML), data exchange over Internet can have a uniform format. XML is a simple, flexible, and structured text format derived from Standard Generalized Markup Language (SGML) (ISO8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is playing an increasingly important role in the exchange of wide varieties of data over the Internet, such as MathML [60] for mathematics, CML [61] for molecules, SMIL [62] for multimedia, SVG [63] for graphics, ebXML [64] for electronic business, OFX [65] for financial data exchange, and WML [66] for wireless applications etc. There are more than 400 XML application areas in the world [67].

An XML-based modeling language, Product Markup Language (PML), is developed for mechanical product information modeling, which is CAD and computer system independent. Inherited from XML, PML has the following general characteristics: (1) Simplicity: the file is a hierarchically tree-structured text. Each object is represented by a node in the tree in the format

of characters and markup tags. This makes it easily readable and comprehensible by machines.

(2) Extensibility: By the nature of markup, PML can be extended for new information if necessary. When new concepts or notations are to be used, a modeling system can extend its language scope by adding new elements in. It provides good scalability for modeling systems.

(3) Portability and interoperability: The language tends to separate system-independent content and system-dependent format of product information so that useful information about product will not be lost during data exchange and translation. PML can include more information in product files. It has the capability to include engineering information, such as materials, tools selection, cutting path, and managerial information, such as order number, cost, as well as geometric information from different levels.

(4) Object-oriented: The inherent hierarchical tree structure of the language enables good encapsulation such that modular transparency is guaranteed for the top-down approach of design. Products are modeled by PML, which describes the information about the product explicitly, such as geometries, functions, features, materials and contexts. Theoretically all information about product can be modeled in PML.

(5) Compatible with Information Infrastructure: XML is regarded as the future of web technology. PML is compatible to web standards. Compatibility is indispensable when building an open and interoperable system.

Some research has been done on the application of XML in CAD/CAM area. Ratchev *et al.* [68] developed a decision-making environment for distributed product and facility prototyping in an extended enterprise. XML is used for conveying design and manufacture messages across traditional technology boundaries. Kahn *et al.* [69] are working on a framework for transforming EXPRESS into XML and viewing with standard WWW browsers. Burkett [70] proposed a mapping between EXPRESS and XML Data Type Definition (DTD). NIST's Design Repository

project [71, 72] created XML mappings for the function and flow in order to support representation of artifact function models in software systems.

The above research represents geometry based on existing neutral formats (STEP or VRML). They do not consider that the relations among geometric entities represent the major part of design knowledge. The problem of design information incompleteness is not resolved. The major advancement of XML for information modeling is that it has standard syntax. Thus the interoperability of semantics can be separated from the interoperability of syntax. Taking advantage of XML to model design entities and relations is one of the promising directions for solving CAD interoperability issues.

4.1 The Syntax of PML

The syntax of PML strictly follows that of XML to ensure the usability and interoperability. The compliance to industrial computation and communication standard is the premise of computational interoperability at the machine level. The syntax of XML in Extended Backus-Naur Form is listed in Appendix I, which is specified at the World Wide Web consortium [73].

Markup is text that is added to the data of a document in order to convey information about it [74]. There are four kinds of markup in SGML: descriptive markup (tags), references, markup declarations, and processing instructions. XML descriptive markup consists of tags and attributes. Matching tags must mark the beginning and the end of each element. Attributes, which are embedded in the start tag, must provide additional information about the element. Unlike HTML, in which tag set is under the control of the creators of HTML browser, XML puts

control of the tag set in the user's hands. Users can create new tags as needed, which makes an XML file extensible. Figure 3 shows an example of point modeled in PML following the syntax of XML. It shows that *Point1* is a point, where its x, y and z coordinate are 1.0, 1.0 and 0.0 respectively. Tag set <point> and </point> specifies the geometric meaning of symbol *Point1* and its attributes of x, y and z.

```
<POINT id="point1" x="1.0", y="1.0", z="0.0">
</POINT>
```

Figure 11: A point in PML

4.2 The Schema of PML

XML provides a type of syntax for modeling data. It offers a user-defined and extensible format to model data and information for different application areas. To enable an XML-style language to be used in a particular area, additional efforts should be carried out to define the semantics of this language. Therefore, specifying what tags will be used in PML is one of the major tasks in defining PML. This includes what kinds of elements to be used to model geometric and non-geometric entities, what types of attributes to be specified for each entity, how to capture the relations among entities, etc.

There are two ways to specify the structure of instance documents and the data type of each element and attribute in XML, *Data Type Definition* (DTD) which is inherited from SGML, and *Schema* which is developed recently. Some disadvantages of DTD make people turn to develop

Schema. DTD has a different syntax from XML; thus, two processing systems are needed to process XML and DTD separately. Furthermore, DTD supports a limited capability for specifying data types. For example, DTD cannot add value range constraints on elements. DTD does not support all current available data types in databases. Comparatively, Schema has advantage over DTD. Schema uses the same syntax as XML. It is object-oriented and extensible in nature. It has enhanced data type definition to specify element sets, multiple elements with the same name but different contents, etc. It supports attribute grouping, user defined types, namespace, etc. The PML Schema is defined according to W3C's Schema working draft [75].

Figure 12 shows two examples of PML schemas used to define entities. The left-hand side schema file defines geometric point. A geometric point should contain four attributes, which are coordinate x, y, z, and an identification name. The coordinate attributes are defined in the right-hand side schema file, which are referred by the schema of point. Reference between schema files can be built to ensure modularity. Figure 13 is the schema of line entities, which shows that a line is defined by either two points or a point and a vector. Appendix VI lists more examples of PML schemas.

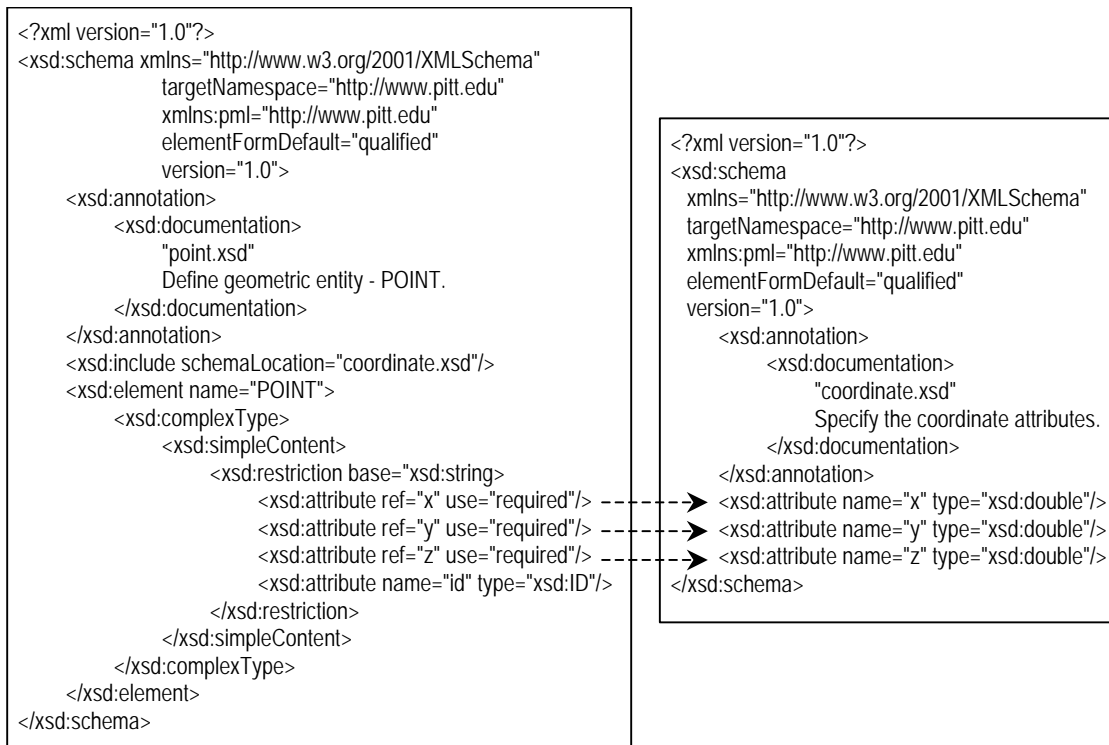


Figure 12: Schema of POINT referring to coordinates

```

<?xml version="1.0"?>
<xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.pitt.edu"
  xmlns:pml="http://www.pitt.edu"
  elementFormDefault="qualified"
  version="1.0">
  <xsd:annotation>
    <xsd:documentation>
      "line.xsd"
      Define geometric entity - LINE.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:include schemaLocation="refPoint.xsd"/>
  <xsd:include schemaLocation="refVector.xsd"/>
  <xsd:element name="LINE">
    <xsd:complexType>
      <xsd:choice>
        <xsd:sequence>
          <xsd:element ref="refPOINT"/>
          <xsd:element ref="refPOINT"/>
        </xsd:sequence>
        <xsd:sequence>
          <xsd:element ref="refPOINT"/>
          <xsd:element ref="refVECTOR"/>
        </xsd:sequence>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

Figure 13: Schema of LINE

The relation of entities in UL model is symbolized by the protocols of XML Xlink [76]. There are two kinds of links in Xlink: *simple* and *extended*. Simple links offer shorthand syntax for a common and outbound link with exactly two participating resources. Extended links offer full Xlink functionality, such as inbound and third-party arcs, as well as links that have arbitrary numbers of participating resources.

In PML, static relations are modeled by simple links and dynamic relations are modeled by simple or extended links. Links can be *local* within one file, or *remote* between files. A reference is constructed by a *reference ID*, which include a Uniform Resource Identifier (URI) specifying

the name and the location of the referred data file and the referred entity ID. If no URI is specified, the reference is a local one. The syntax of the reference ID is shown in Figure 14.

```

<reference_id> ::= # <entity_id> | <URI> # <entity_id>
<entity_id> ::= <part_id> | <assembly_id> | <topology_id> | <geometry_id> |
               <constraint_id>
<topology_id> ::= <body_id> | <shell_id> | <wire_id> | <face_id> | <edge_id> |
                  <coedge_id> | <vertex_id>
<geometry_id> ::= <surface_id> | <curve_id> | <point_id> | <vector_id>

```

Figure 14: Syntax of reference ID

To model the data structure of DHG by a tree structure of PML, a mapping process is needed. The mapping from DHG to PML tree is done under the guidance of graph decomposition rules, which are described in the following section.

4.3 Graph Decomposition

In DHGs, entities have hierarchical structure of static relations, as well as other dynamic relations. To model the hyper-graph structure with a tree-structured PML, the *graph decomposition* procedure should be carried out. The purpose of graph decomposition is to disintegrate the graph structure of the data model into a tree structure by introducing virtual entities to mirror some geometric or non-geometric entities. Thus, the graph structure can be mapped to the tree structure of PML.

Definition 4.1: A *mirror* of an entity is a virtual entity that reflects the referred entity, thereby containing all the attributes of the original entity.

The principles of graph decomposition are listed as follows:

- (1) Entities are represented by elements/nodes in PML.
- (2) Relations are represented by links in PML.
- (3) The bondage of a mirror with its original entity is represented by a simple link that is from the mirror to the original entity.
- (4) An aggregate relation is represented in a parent-child relation of elements/nodes in PML in which the parent is the initial entity and the child is the mirror of terminal entity.
- (5) An association relation is represented in a parent-child relation of elements/nodes in PML in which the parent is the initial entity and the child is the mirror of terminal entity.
- (6) A dynamic relation (constraint) is represented by a simple link, which is from the constraint entity to the constrained entity if only one entity is involved in the relation.
- (7) A dynamic relation (constraint) is represented by an extended link whose children specify the initial entities and terminal entities of the relation if two or more entities are involved in the relation.

The graph decomposition algorithm is listed in Figure 15, assuming that the topological hierarchy is as in Figure 16.

```

INPUT: Directed Hyper Graph G = (V, E)
OUTPUT: PML Tree T

Add root node TR of T
TR add child TG (Geometry)
TR add child TT (Topology)
TR add child TC (Constraint)

Search the topological node 'BODY' in G
Add a node A corresponding to 'BODY' as a child of TT
Run the following procedure P with input <'BODY', A>
  P: On input node <M, I>
    START P
      Mark M in G
      FOR each unmarked node N with a path from M
        IF N is a topological entity
          Add a node J corresponding to N
            as a child of TT
          Add a mirror node of J as the child of I
            with a simple link referring to J
          Run P on input <N, J>
        ENDIF
        IF N is a geometric entity
          Add a node J corresponding to N
            as a child of TG
          Add a mirror node of J as the child of I
            with a simple link referring to J
          Run P on input <N, J>
        ENDIF
        IF N is a constraint entity
          Add a node J corresponding to N
            as a child of TC
          Add an extended link locator node LOC1
            referring to M as a child of J
          Add an extended link locator node LOC2
            referring to N as a child of J
          Add an extended link arc node starting
            from LOC1 to LOC2 as a child of J
          IF there is a path from N to M
            Add an extended link arc node starting
              from LOC2 to LOC1 as a child of J
          ENDIF
        ENDIF
      ENDFOR
    END P

```

Figure 15: Graph decomposition algorithm

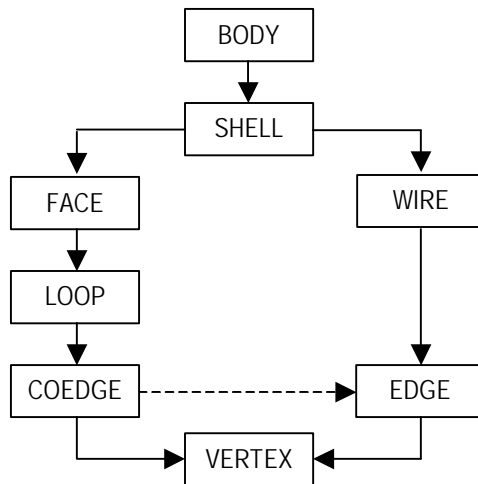


Figure 16: Assumed topological hierarchy

Following the rules of graph decomposition, the DHG of the triangular sheet metal part example in Figure 9 can be transformed to a tree structure, as shown in Figure 17. Thus the design information is expressed in PML and can be easily parsed by computer systems, as shown in Figure 18. Elements with a prefix of “con” are constraint entities. Elements with a prefix of “ref” are mirror entities. For example, *conDISTANCE* is a distance constraint entity, and *refPOINT* is the mirror of *POINT*. The tree structure of PML documents allows computers to do geometric and non-geometric edition, operation, query, and other manipulation efficiently. All relevant product information is stored in a PML tree. The PML file can be read into a CAD system and the information can be translated into the system’s internal representation.

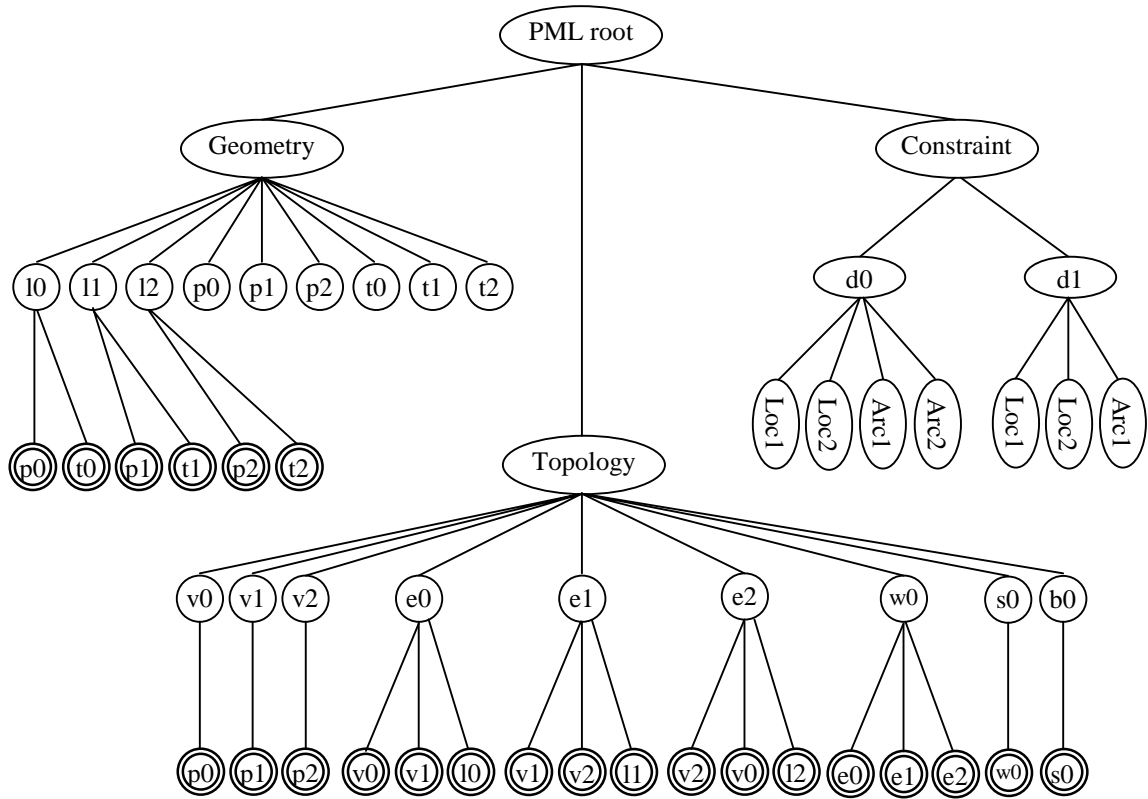


Figure 17: Tree structure of entities in Figure 8 after graph decomposition

```

<?xml version="1.0"?>
<pml:PART id="part0" xmlns:pml="http://www.pitt.edu" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink" xsi:schemaLocation="http://www.pitt.edu/line.xsd">
  <pml:GEOMETRY>
    <pml:POINT id="p0" x="0.0" y="0.0" z="0.0"/>
    <pml:POINT id="p1" x="20.0" y="0.0" z="0.0"/>
    <pml:POINT id="p2" x="12.0" y="10.0" z="0.0"/>
    <pml:VECTOR id="t0" x="20.0" y="0.0" z="0.0"/>
    <pml:VECTOR id="t1" x="-8.0" y="10.0" z="0.0"/>
    <pml:VECTOR id="t2" x="-12.0" y="-10.0" z="0.0"/>
    <pml:LINE id="l0">
      <pml:refPOINT xlink:type="simple" xlink:href="#p0" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVECTOR xlink:type="simple" xlink:href="#t0" xlink:show="embed" xlink:actuate="onLoad"/> </pml:LINE>
    <pml:LINE id="l1">
      <pml:refPOINT xlink:type="simple" xlink:href="#p1" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVECTOR xlink:type="simple" xlink:href="#t1" xlink:show="embed" xlink:actuate="onLoad"/> </pml:LINE>
    <pml:LINE id="l2">
      <pml:refPOINT xlink:type="simple" xlink:href="#p2" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVECTOR xlink:type="simple" xlink:href="#t2" xlink:show="embed" xlink:actuate="onLoad"/> </pml:LINE>
  </pml:GEOMETRY>
  <pml:TOPOLOGY>
    <pml:VERTEX id="v0">
      <pml:refPOINT xlink:type="simple" xlink:href="#p0" xlink:show="embed" xlink:actuate="onLoad"/> </pml:VERTEX>
    <pml:VERTEX id="v1">
      <pml:refPOINT xlink:type="simple" xlink:href="#p1" xlink:show="embed" xlink:actuate="onLoad"/> </pml:VERTEX>
    <pml:VERTEX id="v2">
      <pml:refPOINT xlink:type="simple" xlink:href="#p2" xlink:show="embed" xlink:actuate="onLoad"/> </pml:VERTEX>
    <pml:EDGE id="e0" pml:startParam="0" pml:endParam="20">
      <pml:refVERTEX xlink:type="simple" xlink:href="#v0" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVERTEX xlink:type="simple" xlink:href="#v1" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refCURVE xlink:type="simple" xlink:href="#l0" xlink:show="embed" xlink:actuate="onLoad"/> </pml:EDGE>
    <pml:EDGE id="e1" pml:startParam="0" pml:endParam="12.8062484748657">
      <pml:refVERTEX xlink:type="simple" xlink:href="#v1" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVERTEX xlink:type="simple" xlink:href="#v2" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refCURVE xlink:type="simple" xlink:href="#l1" xlink:show="embed" xlink:actuate="onLoad"/> </pml:EDGE>
    <pml:EDGE id="e2" pml:startParam="0" pml:endParam="15.6204993518133">
      <pml:refVERTEX xlink:type="simple" xlink:href="#v2" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVERTEX xlink:type="simple" xlink:href="#v0" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refCURVE xlink:type="simple" xlink:href="#l2" xlink:show="embed" xlink:actuate="onLoad"/> </pml:EDGE>
    <pml:WIRE id="w0">
      <pml:refEDGE xlink:type="simple" xlink:href="#e0" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refEDGE xlink:type="simple" xlink:href="#e1" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refEDGE xlink:type="simple" xlink:href="#e2" xlink:show="embed" xlink:actuate="onLoad"/> </pml:WIRE>
    <pml:SHELL id="s0">
      <pml:refWIRE xlink:type="simple" xlink:href="#w0" xlink:show="embed" xlink:actuate="onLoad"/> </pml:SHELL>
    <pml:BODY id="b0">
      <pml:refSHELL xlink:type="simple" xlink:href="#s0" xlink:show="embed" xlink:actuate="onLoad"/> </pml:BODY>
  </pml:TOPOLOGY>
  <pml:CONSTRAINT>
    <pml:conDISTANCE xlink:type="extended" pml:lowerBound="19" pml:upperBound="21">
      <pml:LOC1 xlink:type="locator" xlink:label="start" xlink:href="#v1"/>
      <pml:LOC2 xlink:type="locator" xlink:label="end" xlink:href="#v0"/>
      <pml:ARC1 xlink:type="arc" xlink:from="start" xlink:to="end" xlink:actuate="onRequest"/>
      <pml:ARC2 xlink:type="arc" xlink:from="end" xlink:to="start" xlink:actuate="onRequest"/> </pml:conDISTANCE>
    <pml:conDISTANCE xlink:type="extended" pml:lowerBound="9" pml:upperBound="11">
      <pml:LOC1 xlink:type="locator" xlink:label="start" xlink:href="#v2"/>
      <pml:LOC2 xlink:type="locator" xlink:label="end" xlink:href="#e0"/>
      <pml:ARC1 xlink:type="arc" xlink:from="start" xlink:to="end" xlink:actuate="onRequest"/> </pml:conDISTANCE>
  </pml:CONSTRAINT>
</pml:PART>

```

Figure 18: PML representation of the triangular part in Figure 8 and Figure 9

3D solid models can also be represented in the UL-PML scheme. For example, a tetrahedron in Figure 19 is modeled in DHG as in Figure 20 and in PML in Figure 21.

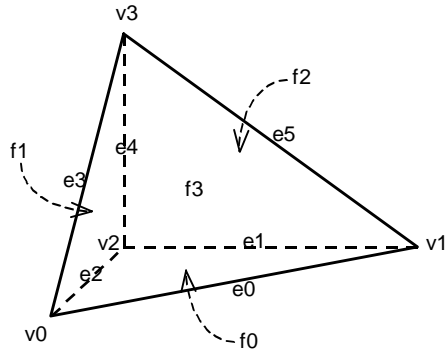


Figure 19: A tetrahedron

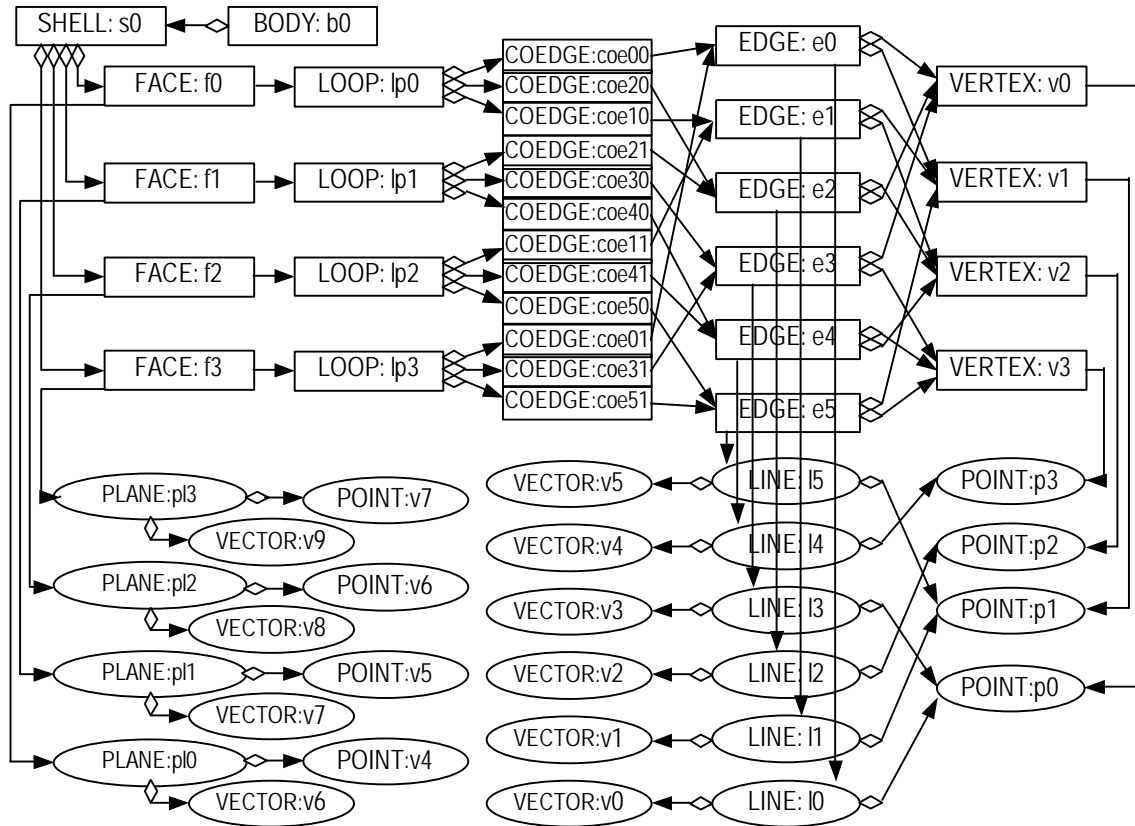


Figure 20: DHG model of the tetrahedron in Figure 19

```

<?xml version="1.0"?>
<pml:PART id="part0" xmlns:pml="http://www.pitt.edu" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink" xsi:schemaLocation="http://www.pitt.edu/line.xsd">
  <pml:GEOMETRY>
    <pml:POINT id="p0" x="0.0" y="0.0" z="1.0"/>
    <pml:POINT id="p1" x="1.0" y="0.0" z="0.0"/>
    <pml:POINT id="p2" x="0.0" y="0.0" z="0.0"/>
    <pml:POINT id="p3" x="0.0" y="1.0" z="0.0"/>
    <pml:POINT id="p4" x="0.0" y="0.0" z="1.0"/>
    <pml:POINT id="p5" x="1.0" y="0.0" z="0.0"/>
    <pml:POINT id="p6" x="0.0" y="0.0" z="0.0"/>
    <pml:POINT id="p7" x="0.0" y="1.0" z="0.0"/>
    <pml:VECTOR id="t0" x="1.0" y="0.0" z="-1.0"/>
    <pml:VECTOR id="t1" x="-1.0" y="0.0" z="0.0"/>
    <pml:VECTOR id="t2" x="0.0" y="0.0" z="1.0"/>
    <pml:VECTOR id="t3" x="0.0" y="1.0" z="-1.0"/>
    <pml:VECTOR id="t4" x="0.0" y="-1.0" z="0.0"/>
    <pml:VECTOR id="t5" x="-1.0" y="1.0" z="0.0"/>
    <pml:VECTOR id="t6" x="0.0" y="-1.0" z="0.0"/>
    <pml:VECTOR id="t7" x="-1.0" y="0.0" z="0.0"/>
    <pml:VECTOR id="t8" x="0.0" y="0.0" z="-1.0"/>
    <pml:VECTOR id="t9" x="0.577350" y="0.577350" z="0.577350"/>
    <pml:LINE id="l0">
      <pml:refPOINT xlink:type="simple" xlink:href="#p0" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVECTOR xlink:type="simple" xlink:href="#t0" xlink:show="embed" xlink:actuate="onLoad"/> </pml:LINE>
    <pml:LINE id="l1">
      <pml:refPOINT xlink:type="simple" xlink:href="#p1" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVECTOR xlink:type="simple" xlink:href="#t1" xlink:show="embed" xlink:actuate="onLoad"/> </pml:LINE>
    <pml:LINE id="l2">
      <pml:refPOINT xlink:type="simple" xlink:href="#p2" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVECTOR xlink:type="simple" xlink:href="#t2" xlink:show="embed" xlink:actuate="onLoad"/> </pml:LINE>
    <pml:LINE id="l3">
      <pml:refPOINT xlink:type="simple" xlink:href="#p0" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVECTOR xlink:type="simple" xlink:href="#t3" xlink:show="embed" xlink:actuate="onLoad"/> </pml:LINE>
    <pml:LINE id="l4">
      <pml:refPOINT xlink:type="simple" xlink:href="#p3" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVECTOR xlink:type="simple" xlink:href="#t4" xlink:show="embed" xlink:actuate="onLoad"/> </pml:LINE>
    <pml:LINE id="l5">
      <pml:refPOINT xlink:type="simple" xlink:href="#p1" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVECTOR xlink:type="simple" xlink:href="#t5" xlink:show="embed" xlink:actuate="onLoad"/> </pml:LINE>
    <pml:PLANE id="pl0">
      <pml:refPOINT xlink:type="simple" xlink:href="#p4" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVECTOR xlink:type="simple" xlink:href="#t6" xlink:show="embed" xlink:actuate="onLoad"/> </pml:PLANE>
    <pml:PLANE id="pl1">
      <pml:refPOINT xlink:type="simple" xlink:href="#p5" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVECTOR xlink:type="simple" xlink:href="#t7" xlink:show="embed" xlink:actuate="onLoad"/> </pml:PLANE>
    <pml:PLANE id="pl2">
      <pml:refPOINT xlink:type="simple" xlink:href="#p6" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVECTOR xlink:type="simple" xlink:href="#t8" xlink:show="embed" xlink:actuate="onLoad"/> </pml:PLANE>
    <pml:PLANE id="pl3">
      <pml:refPOINT xlink:type="simple" xlink:href="#p7" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVECTOR xlink:type="simple" xlink:href="#t9" xlink:show="embed" xlink:actuate="onLoad"/> </pml:PLANE>
  </pml:GEOMETRY>
  <pml:TOPOLOGY>
    <pml:VERTEX id="v0">
      <pml:refPOINT xlink:type="simple" xlink:href="#p0" xlink:show="embed" xlink:actuate="onLoad"/> </pml:VERTEX>
    <pml:VERTEX id="v1">
      <pml:refPOINT xlink:type="simple" xlink:href="#p1" xlink:show="embed" xlink:actuate="onLoad"/> </pml:VERTEX>
    <pml:VERTEX id="v2">
      <pml:refPOINT xlink:type="simple" xlink:href="#p2" xlink:show="embed" xlink:actuate="onLoad"/> </pml:VERTEX>
    <pml:VERTEX id="v3">
      <pml:refPOINT xlink:type="simple" xlink:href="#p3" xlink:show="embed" xlink:actuate="onLoad"/> </pml:VERTEX>
    <pml:EDGE id="e0" pml:startParam="0" pml:endParam="20">
      <pml:refVERTEX xlink:type="simple" xlink:href="#v0" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVERTEX xlink:type="simple" xlink:href="#v1" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refCURVE xlink:type="simple" xlink:href="#l0" xlink:show="embed" xlink:actuate="onLoad"/> </pml:EDGE>
    <pml:EDGE id="e1" pml:startParam="0" pml:endParam="12.8062484748657">
      <pml:refVERTEX xlink:type="simple" xlink:href="#v1" xlink:show="embed" xlink:actuate="onLoad"/>

```

Figure 21: PML model of the tetrahedron in Figure 19

4.4 Demonstration

UL-PML scheme models product information in a distributed fashion, thus it allows physically dispersed CAD data in a collaborative design environment to be integrated logically. It does not require all geometric and non-geometric data of a part or an assembly reside in one CAD system. Within the limit of network and communication bandwidth, one can break the traditional large CAD files into small pieces, thus partial data query and transferring are supported by this scheme. Having a standard XML format, PML can be easily processed for reading, writing, storing, query, and transferring based on current computational standards and network protocols, which possibly makes it widely acceptable by different CAD systems.

Unlike current CAD files with the information granularity for transferring at the component level, UL-PML scheme allows CAD data communication at the basic geometric and non-geometric entity level. For example, a connector in Figure 22 is to be designed by two groups, the head section by one group and the body section by another. While the body section is being designed at one location (in Figure 23), the head section (in Figure 24) data file at another location is referring to the top face of the body by linkage specified by URIs as in Figure 25. One section of a part can be linked to another section during the component design. In a similar way, an assembly file can also refer to the distributed files containing several components.

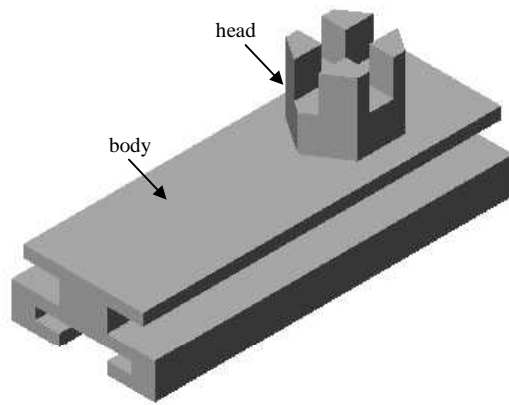


Figure 22: A part to be designed by two groups

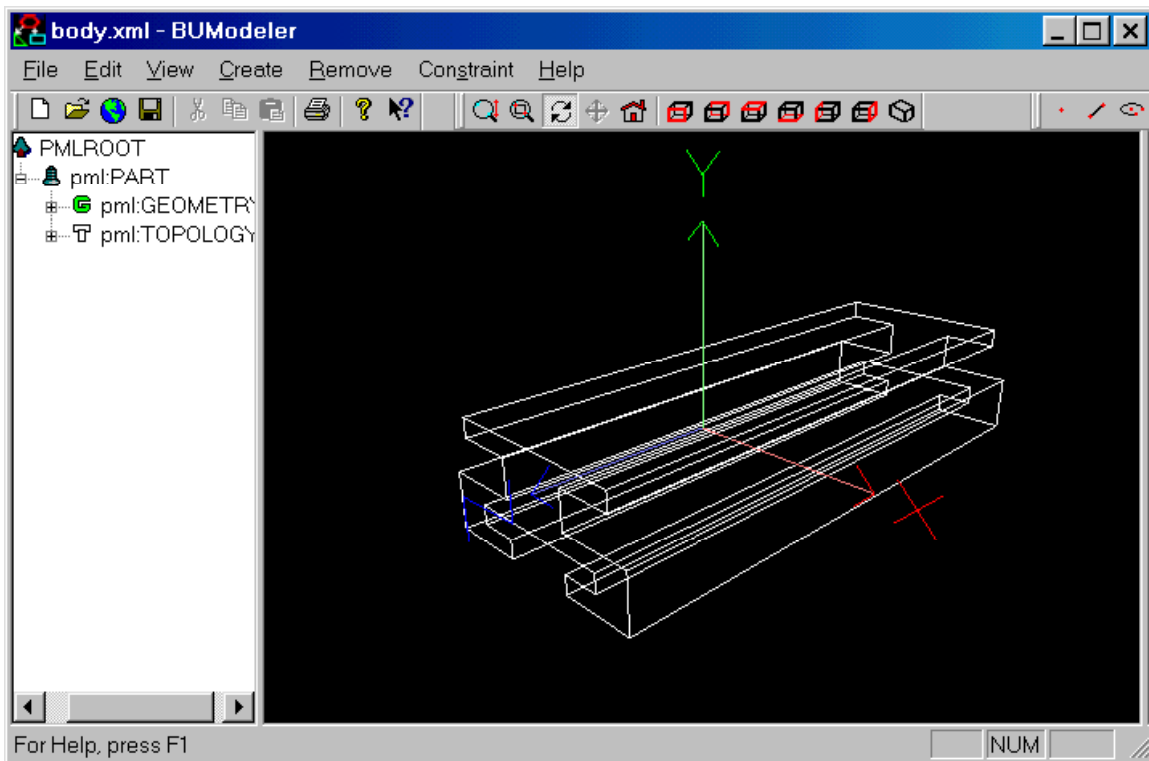


Figure 23: The body section of the part

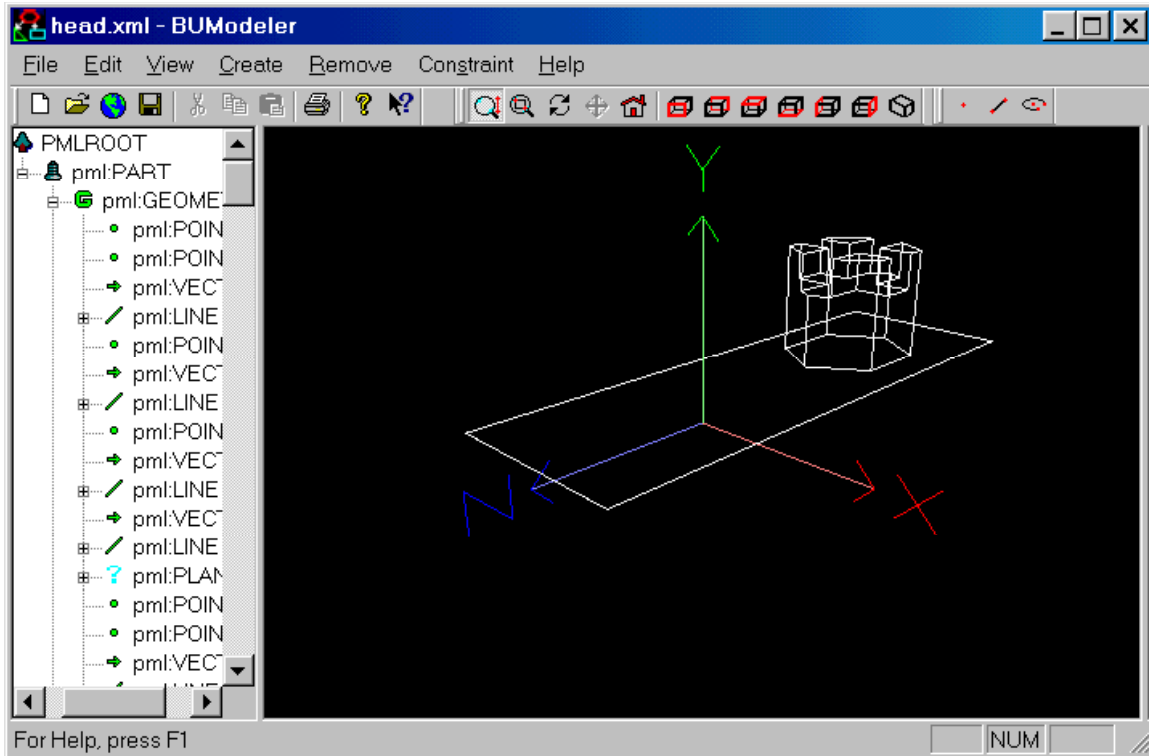


Figure 24: The head section of the part

```

.
.
.
<pml:FACE id="face16">
  <pml:refFACE xlink:type="simple" xlink:href="http://www.pitt.edu/~yawst4/pg/body.xml#face16"
    xlink:show="embed" xlink:actuate="onLoad"/>
.
.
.

```

Figure 25: Universal linkage by URI

In summary, UL model captures geometric and non-geometric relations among entities by uniform and explicit links in PML such that references between entities can be made across the boundary of files and physical locations in a distributed design environment. To main syntax-level interoperability, PML uses standard XML syntax. Schemas of PML are defined for entities and relations. Tree-structured PML allows design information to be easily processed. Graph decomposition method is developed to map graph-structured entities and relations to PML tree. The properties of UL model include:

(1) UL model does not require that one data file contain all information relevant to the designed product. Supporting physical distribution, it makes partial design information storage and retrieval easy to realize. This will increase the efficiency during design information query.

(2) The design information can be stored modularly without compromising the integrity of the whole product. This eases the requirements on computational time and storage space. Thus it provides good flexibility for scalable designer systems. It also encourages reuse of designed components/sections, thus reuse of design knowledge.

(3) The linkage ensures product data's logical integrity though it is physically distributed. Link relations among entities in UL model create a distributed information framework, thus collaborative design can be easily realized over the Internet.

(4) The design data elements and constraints are connected within the model by links. The linkage makes the design data model open and extensible. Information can be generated and linked together in the network virtual space.

(5) With lean product information transferring, design collaborators can share necessary design information without losing control of intellectual properties. This scheme thus enables easy management of trust relation and design information security in a collaborative design environment.

(6) The geometric and non-geometric constraint representation in UL model incorporates more design knowledge in design data. It provides a more comprehensive support for optimization and decision-making at different design stages.

(7) The explicit capturing of multidisciplinary constraints, especially non-geometric constraints, allows a more complete information representation than current standard formats. Thus it prevents design information loss and reduces the design cycle time.

Based on this general UL-PML scheme, a high-level data model is developed to represent features and constraints in order to support distributed feature-based parametric modeling, which is described in the following chapter.

5.0 DESIGN FEATURE AND CONSTRAINT REPRESENTATION

Feature-based parametric modeling is the new approach used in most of the modern CAD systems to derive geometric forms. Features are used in geometry construction instead of low-level geometric entities, such that shapes can be built with terminologies that are more intuitive and meaningful for human designers and engineers, and faster for model reconstruction and reuse. Features contain design information of model construction history besides the geometry boundary. Parameter information is recorded for constraint and specification driven design, and for ease of model re-evaluation in design variation. Geometry and form information is stored by geometric and topological entities at the low level, whereas design intent is recorded at the high level by features and parameters. Feature-based parametric modeling facilitates geometry construction process. Nevertheless, it signifies the interoperability problem of design information and knowledge capturing.

During the process of design, requirements from different stakeholders are imposed on design as specifications or constraints, either geometric or non-geometric. As defined in Section 3.2, constraints represent dynamic relations among entities specified by users. A simple but comprehensive enough scheme to represent constraints is vital for design knowledge and information representation in a collaborative design environment.

Constraint should not only be looked as the complementary part of design. It is the result of logic reasoning activity of engineers and other design participants during the design process. It is

the specifications and constraints from different aspects that finalize the physical form of a design.

Geometric constraints are the fundamental constraints to be captured in mechanical product design. The study of geometric constraints representation can be traced back to the origin of CAD research. Constrained geometries are sets of loci that satisfy certain constraints, thus they can be constructed systematically by computer systems.

Different types of geometric constraint solving methods and associated representation methods for CAD have been proposed. Generally there are four approaches. The numerical approach [77, 78, 79, 80, 81, 82] translates geometric constraints into a system of mathematical equations. These equations then can be solved numerically by Newton-Raphson or Homotopy methods directly, or by minimizing the sum of squares for all equations indirectly. The artificial intelligent approach [83, 84, 85, 86, 87, 88] represents geometric constraints by facts and rules. Constraint problems are solved by the aid of geometric reasoning. The symbolic approach [89, 90, 91, 92] translates geometric constraints into a system of easily solvable nonlinear equations with symbolic algebraic methods, such as Grobner's bases or the Wu-Ritt method, before numerically solving them. The constructive approach [93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105] represents constraints as graphs internally. Constraint system is solved either by top-down decomposition or by bottom-up clustering of the constraint graphs along with degrees of freedom analysis.

To support different constraint solving methods in various CAD systems, a neutral model for feature and constraint representation should be included in enriched CAD data. Current neutral CAD data formats utilize the explicit modeling method to represent geometric entities. Implicit geometric relations (such as dimensions and constraints in parametric design tools)

cannot be represented. Furthermore, features, which capture design process and history in parametric design, cannot be represented in these neutral formats. To ensure various design information and rationale is captured explicitly in CAD data, modification and extension of neutral CAD data formats are needed.

Some research efforts have been granted to include parametric information in STEP. The program of Enabling Next GENERation mechanical design (ENGEN) [106] was sponsored by Defense Advanced Projects Research Agency (DARPA) and PDES, Inc., and National Institute of Standards and Technology (NIST) Parametric Group [107]. Though some form features and geometric constraints are modeled in the above research, the representation method is not generic enough to consider both implicit and explicit modeling, and to include geometric and non-geometric constraints. Design features represent the history of construction, which contains design intent. To allow other design participants to understand the design intent behind the shape, and to do modification directly on the same geometry in different CAD systems, design features and the transition from implicit model to explicit model should be included in CAD neutral formats to enhance interoperability.

Different disciplines have their own domain specifications or constraints. Design constraints consist of specifications in both geometric aspect (e.g., dimensions, parallelism, and concentricity) and non-geometric aspect (e.g., functionality, materials, process requirements, and ergonomics). During the process of design, geometric constraints are imposed on the geometry to find the loci and generate the desired physical shape, while non-geometric constraints are first processed by designers based on design knowledge and interpreted to the corresponding geometric constraints. The physical shape of a design is determined by geometric constraints directly and non-geometric constraints indirectly. Based on different interests, constraints can be

categorized in different ways. An example of constraint categorization is shown in Figure 26. Geometric constraints are looked as low-level constraints and non-geometric constraints are high-level constraints in design specifications.

It is important to capture non-geometric constraints explicitly in product data in order to prevent information misinterpretation or loss. For example, the diameter of a shaft could be determined by the limit of machining tools, the dimensions of mating parts, the level of bearable load, or the strength of the material. The diameter alone cannot represent the actual specification. The explicitly specified non-geometric constraints need to be modeled to retain the source of geometric interpretation. There are also some other non-geometric constraints that cannot be generally interpreted into geometric information, such as design related material properties, manufacturing processes, and working environments.

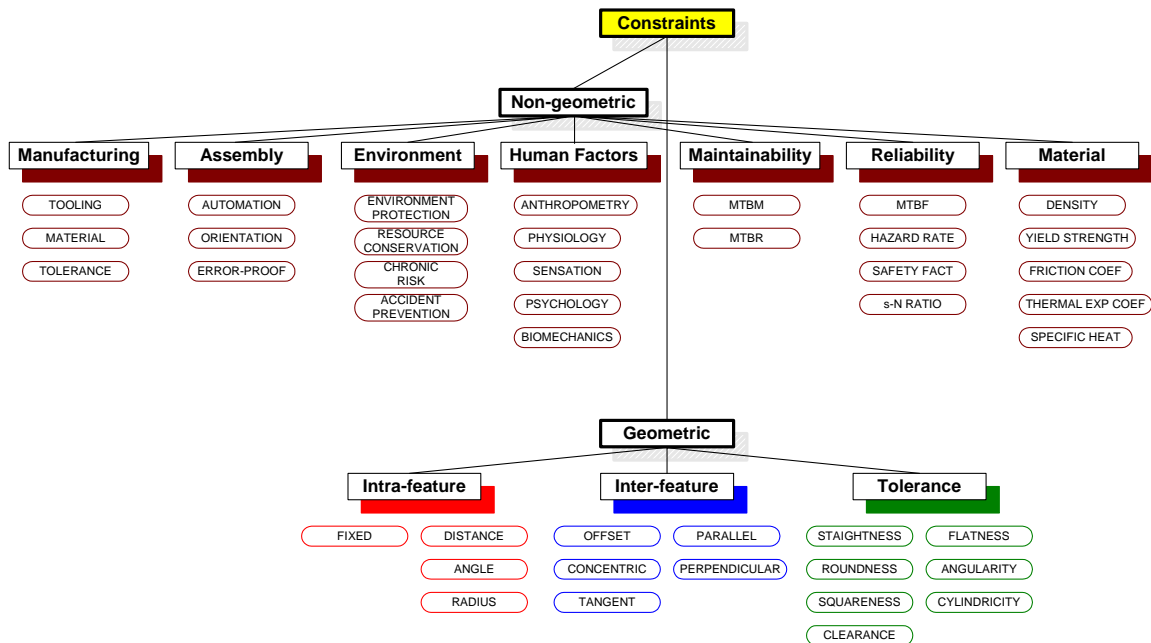


Figure 26: Tree of geometric and non-geometric constraints

The UL-PML scheme developed in this research support features and constraints in parametric design. It captures design features and their relations to low-level entities. It also incorporates non-geometric constraints to preserve design information integrity.

5.1 Design Feature Representation

Features represent regions of interests for different application purposes. For example, function realization and geometry definition are important during design. Material removal methods, tool selection, and tolerances are the major concerns in manufacturing. Spatial and kinematical relationships are of interest for assembly. The taxonomy of features is application specific. Here, the design feature or form feature that is applied in CAD model construction is the main domain of discussion.

Within a feature-based parametric modeling environment, geometric shapes or forms are constructed by high-level units – design features. The feature-based construction procedure represents design intent and variation information, which are useful for downstream activities, such as design modification, model validation, and manufacturing. Feature-based design has been widely used in current CAD systems. Features become indispensable tools to aid designers to express ideas and histories of design. Information about features should be modeled as part of transferable product data for heterogeneous CAD modeling systems.

In geometric modeling systems, design features or form features can be represented in two levels. One is termed *implicit* or *unevaluated*, where features are defined by construct procedures and parameters. Another is called *explicit* or *evaluated*, by which features are defined by low-level geometric and topological elements.

Most of research in form feature modeling uses implicit representation. For example, in ASU Features Testbed Modeler [108, 109], features are defined in terms of various parameters and rules about geometric shape. Interaction between features includes spatial relationship and volume-based CSG tree and Boolean operations. This modeling scheme only uses predefined geometric information, thus it makes feature classification not flexible enough.

E-REP [110, 111, 112, 113, 114] distinguishes generated features, datum features, and modifying features and regards a CAD model as being built entirely by a sequence of feature insertion, modification, and deletion description. This description then is translated to explicit entity representation. This approach allows feature-based modeling to be independent of current different CAD systems. But at the same time, features are isolated with entities. The constructional procedures do not directly associate with entities.

Middleditch and Reade [115] proposed a hierarchical structure for feature composition and emphasized the construct relationship of features, but failed to build the connection between features and low-level entities.

Some research represents features explicitly. Based on current framework of STEP standards, the ENGEN Data Model (EDM) [106, 116] extended STEP's current explicit entity representation by adding some predefined local features such as round and chamfer. EDM took a bottom-up approach only and considered the low-level entity construction process, but did not consider implicit modeling aspects in a parametric modeling environment.

The most commonly used CAD systems use a mixed representation of CSG and B-Rep. It is vital that a widely acceptable CAD data model should be able to capture feature-level information as well as geometric and topological entities and relations. Pratt and Anderson [117] also advocate that the future CAD data modeling standard should support both explicit modeling

and implicit modeling. A hybrid of descriptive and procedural representation will accommodate requirements from different aspects.

PDES's Form Feature Information Model (FFIM) [118, 119] adopted a dual representation of explicit and implicit features. Explicit features are represented generally by face lists, while implicit features are categorized into depression, protrusion, passage, deformation, transition, and area features. The limitation of FFIM is that parameters and constraints are not supported, and the relation between explicit features and low-level entities is not explicitly modeled.

Some researchers used a hybrid CSG/B-Rep structure. Roy and Liu [120] constructed CSG using form primitives and form features. A face-edge type data structure is used at the low-level B-Rep. These two data structures are linked by reference faces. Wang and Ozsoy [121] used primitive features and form features to build CSG structure. Dimension and orientation information are represented as constraint nodes in CSG tree. A face-edge type data structure is used for lower level entities. The connection between two structures is built by pointers from set operator nodes in CSG to B-Rep data structure and from faces to feature faces. Gomes and Teixeira [122] also developed a CSG/B-Rep scheme, in which CSG represents the high-level relationships between features, and the B-Rep model describes the details. An additional Feature Topological Structure in parallel with the B-Rep model defines volume form features.

The above hybrid representations build CSG trees using pre-defined features. Although connections between features and low-level entities are built, these hybrid approaches are not generic enough. Some local operations such as chamfer, fillet, and thread cannot be implemented purely in CSG context. Feature identification and mapping procedures in different modeling systems may not be easy if some systems do not contain a particular feature. Thus, the definition of a feature itself needs to be captured.

From a more general point of view, an intentional feature [123] captures the process of how a feature is defined, and is more flexible than geometric features. An intentional feature is an abstraction for accessing groups of geometric elements with certain attributes, while a geometric feature is a physical collection of geometric elements. It is advantageous to model features in a procedural way in terms of intentional features, which separates feature construction and validation. The design feature representation scheme proposed here is a combination of intentional and geometric aspects of features.

5.1.1 Dual representation of features

A definition of feature in terms of information representation for modeling procedures is needed to delineate the scope of feature information elements. Kim and O'Grady [124] proposed an abstract representation in which features are defined as building blocks of part with certain operators, but did not show detailed relations between features and entities. Relations between features and low-level entities are important to make a feature representation generally acceptable by current CAD systems. We define that a design feature is a *relation* between *priori* properties (profile, orientation, attributes, etc.) and *posteriori* properties (derived geometric and topological entities and their relations). The collection of priori properties is called *priori feature*, and the collection of posteriori properties is called *posteriori feature*. Priori features consist of construction intents and procedures, while posteriori features have evaluated geometric shape information.

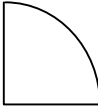
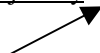
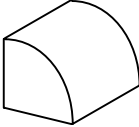



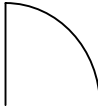

If E is a set of low-level entities (geometry and topology), and R is a set of relations between entities, a CAD model D can be defined as a set of points in the E-R space, denoted as $D = (E, R)$. Furthermore, E can be subdivided into spaces of topology and geometry, $E = T \cup G$, where T is the set of topological entities, and G is the set of geometric entities. R can be

subdivided into spaces of static relation and constraint, $R = S \cup C$, where S is the set of static or structural relations (generalization, aggregation, association), and C is the set of constraints or dynamic relations. Priori features and posteriori features are subspaces of D , and they contain information of T , G , S , and C . If F_i is the set of priori features and F_o the set of posteriori features, $F_i \subset D$, $F_o \subset D$, i.e., $F_i \subset E \times R$, $F_o \subset E \times R$. Feature evaluation is the mapping function $f: F_i \rightarrow F_o$. Design feature F is defined as the relation f . F can also be denoted as (F_i, F_o) . The relations between features and low-level entities thus are built, which can be summarized as $F = ([T_i, G_i] \times [C_i, S_i], [T_o, G_o] \times [C_o, S_o])$, where i and o respectively denote entities or relations belonging to priori and posteriori features. Some examples of features are listed in Table 3.

While the explicit modeling method builds models using elements of T , G , C , and S directly, feature-based modeling composes models in a more structured way by using collections of $\{T, G, C, S\}$. During the process of modeling, entity specifications for priori features are independent of those for posterior features (i.e., $T_i \cap T_o = \emptyset$ and $G_i \cap G_o = \emptyset$). Thus, feature definition is separated with feature evaluation, which allows construction procedure, history, and other design information be captured along with geometry.

In the UL model, priori features are modeled by introducing a new type of entities - feature entities. Priori features (e.g., protrusion, cut, hole, sweep, chamfer, and fillet) are sets of low-level entities and relations that express the construct procedures. The relation between feature entities and topological and geometric entities in priori feature definition are defined as aggregation. Similar to low-level entities, feature entities can be referred as both abstract class and instance. Design feature entities are categorized as geometric entities, and can be represented in DHG. For example, the priori feature of protrusion in Table 3 is represented as in Figure 27.

Table 3: Examples of design features

	Priori Features	Posteriori Features
Protrusion	<p><u>Profile</u>  <u>Trajectory</u> </p> <p>T_i: face \rightarrow loop, edge, vertex G_i: surface \rightarrow line, curve, point, vector C_i: profile dimension, sweep distance S_i: association, aggregation</p>	 <p>T_o: face \rightarrow loop, edge, vertex G_o: surface \rightarrow line, curve, point, vector C_o: dimension / distance, parallelism S_o: association, aggregation,</p>
Cut	<p><u>Profile</u>  <u>Trajectory</u> </p> <p>T_i: loop \rightarrow edge, vertex G_i: line, curve \rightarrow point, vector C_i: profile location, dimension, sweep distance S_i: association, aggregation</p>	 <p>T_o: face \rightarrow loop, edge, vertex G_o: surface \rightarrow line, curve, point, vector C_o: dimension / distance, parallelism S_o: association, aggregation</p>
Fillet	<p><u>FilletEdge</u> </p> <p>T_i: edge \rightarrow vertex G_i: line, curve \rightarrow point, vector C_i: dimension (radii of fillet) S_i: association, aggregation</p>	 <p>T_o: face \rightarrow loop, edge, vertex G_o: surface \rightarrow line, curve, point, vector C_o: dimension / distance, parallelism S_o: association, aggregation</p>

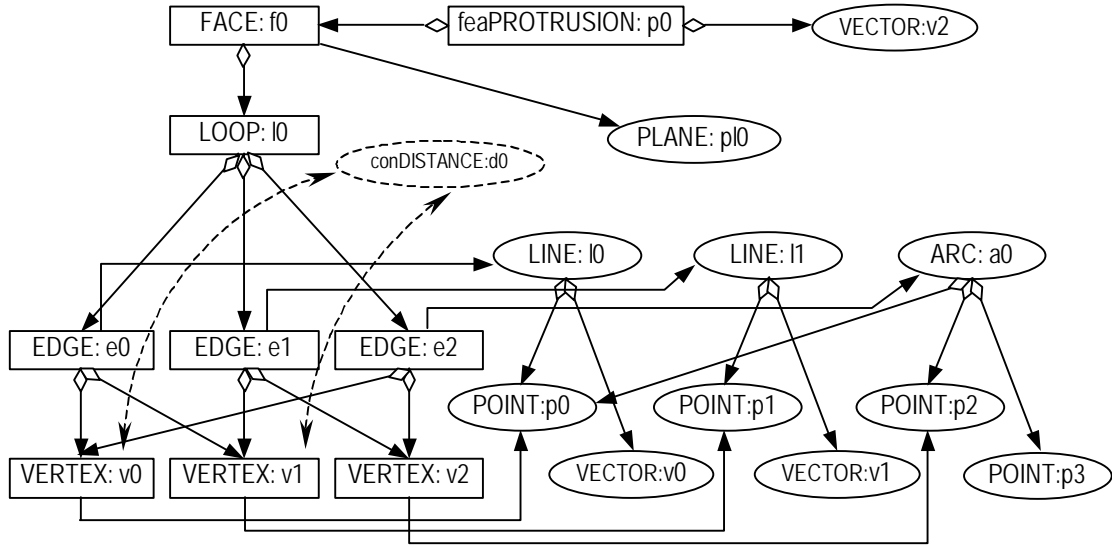


Figure 27: Priori feature of protrusion in DHG

Posteriori features are modeled in the form of collections of low-level entities and their association with high-level feature entities. The boundary topological entities of the models are the connections between geometry and feature. For example, in 3D solid models, a face entity is the pivot of connection between evaluated entities and feature entities, thus priori feature and posteriori feature. The relation between feature entities and face entities in posteriori feature definition are defined as general association. Any new face generated in a feature evaluation is associated with the feature. The posteriori feature of protrusion in Table 3 is illustrated in Figure 28. Through feature entities, two levels of feature representation (i.e., priori features and posteriori features) are linked.

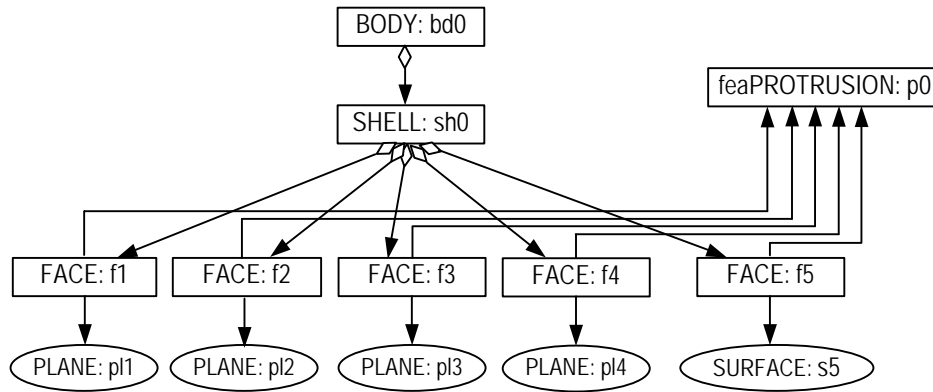


Figure 28: Posteriori feature of protrusion in DHG

In this model, low-level entities of a priori feature are independent of those of the corresponding posteriori feature. It is possible that two sets of entities represent one geometric form. This dual representation scheme makes a priori feature separated from its posteriori counterpart, therefore feature construction is independent of feature evaluation and validation.

For example, the solid part in Figure 29 is constructed by four features: protrusion, extrusion cut, hole, and fillet. The construct procedure is illustrated in Figure 30. The priori features are specified by some low-level entities, either independently defined or evaluated from previous steps, with aggregation relations. Then the feature is evaluated. The generated low-level entities are associated with the priori features, i.e., *feaPROTRUSION*, *feaCUT*, *feaHOLE*, and *feaFILLET*. Some features are specified independent of evaluated entities, e.g., protrusion and cut. In this case, two sets of entities are referring to the same geometry. For example, face *f0* associated with plane *p10* and face *f1* associated with plane *p11* in Figure 30 (a) are referring the same surface, while edges of ring *r0* and intersecting edges between face *f2* and *f7*, *f8* in Figure 30 (b) are referring the same curves. These redundancies are very necessary to preserve information of design intentions. Some features are specified based on evaluated entities from

previous steps, e.g., chamfer and fillet. In Figure 30 (d), edge e_8 is generated at the protrusion creation.

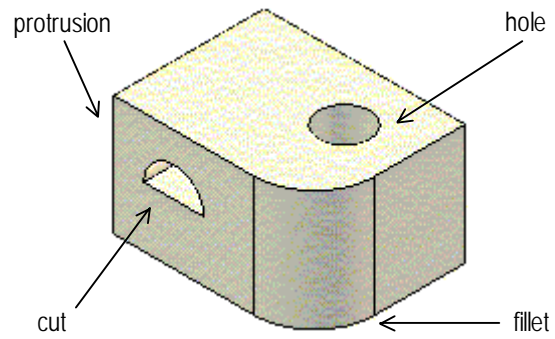
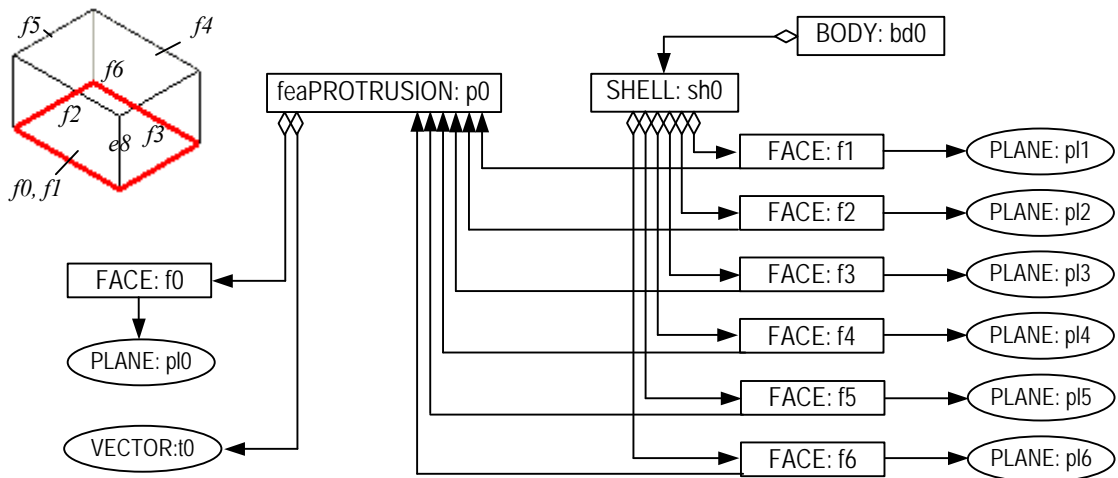
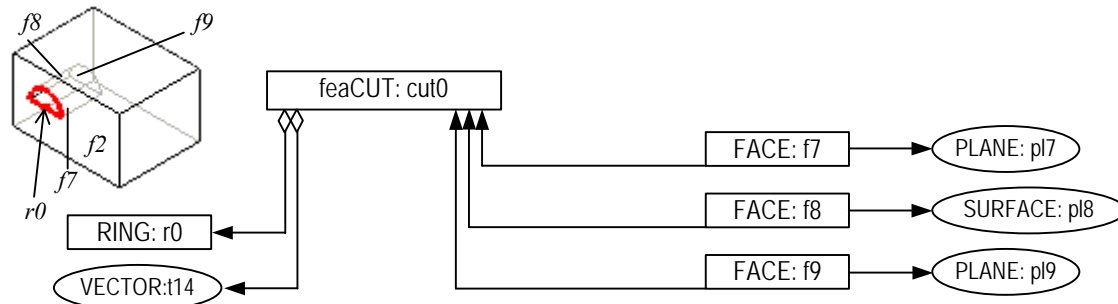


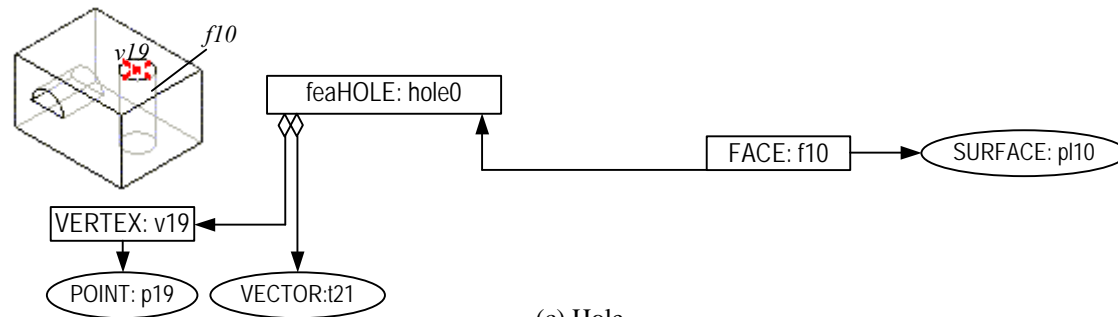
Figure 29: A solid feature example



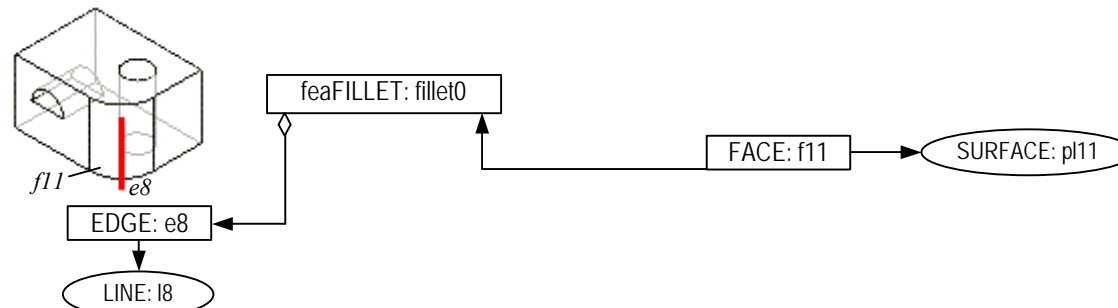
(a) Protrusion



(b) Cut



(c) Hole



(d) Fillet

Figure 30: Feature definition procedure in Figure 29

```

<pml:GEOMETRY>
.....
</pml:GEOMETRY>
<pml:TOPOLOGY>
<pml:FACE id="f0">
  <pml:refLOOP xlink:type="simple" xlink:href="#lp0" xlink:show="embed" xlink:actuate="onLoad"/>
  <pml:refSURFACE xlink:type="simple" xlink:href="#p10" xlink:show="embed" xlink:actuate="onLoad"/> </pml:FACE>
<pml:RING id="r0">
  <pml:refEDGE xlink:type="simple" xlink:href="#e12" xlink:show="embed" xlink:actuate="onLoad"/>
  <pml:refEDGE xlink:type="simple" xlink:href="#e13" xlink:show="embed" xlink:actuate="onLoad"/> </pml:RING>
.....
</pml:TOPOLOGY>
<pml:FEATURE>
<pml:feaPROTRUSION id="p0" depth="40.0">
  <pml:PROFILE>
    <pml:refFACE xlink:type="simple" xlink:href="#f0" xlink:show="embed" xlink:actuate="onRequest"/> </pml:PROFILE>
  <pml:TRAJECTORY>
    <pml:refVECTOR xlink:type="simple" xlink:href="#t4" xlink:show="embed" xlink:actuate="onRequest"/> </pml:TRAJECTORY>
</pml:feaPROTRUSION>
<pml:feaCUT id="cut0" category="extrusion" type="blind" depth="12.0">
  <pml:PROFILE>
    <pml:refRING xlink:type="simple" xlink:href="r0" xlink:show="embed" xlink:actuate="onRequest"/> </pml:PROFILE>
  <pml:TRAJECTORY>
    <pml:refVECTOR xlink:type="simple" xlink:href="#t14" xlink:show="embed" xlink:actuate="onLoad"/> </pml:TRAJECTORY>
</pml:feaCUT>
<pml:feaHOLE id="hole0" type="through_to_next" diameter="16" depth="">
  <pml:PROFILE>
    <pml:refVERTEX xlink:type="simple" xlink:href="#v19" xlink:show="embed" xlink:actuate="onLoad"/> </pml:PROFILE>
  <pml:TRAJECTORY>
    <pml:refVECTOR xlink:type="simple" xlink:href="#t15" xlink:show="embed" xlink:actuate="onLoad"/> </pml:TRAJECTORY>
</pml:feaHOLE>
<pml:feaFILLET id="fillet0" type="simple">
  <pml:FILLET_EDGE radius1="10" radius2="10">
    <pml:refEDGE xlink:type="simple" xlink:href="#e8" xlink:show="embed" xlink:actuate="onRequest"/> </pml:FILLET_EDGE>
  </pml:feaFILLET>
</pml:FEATURE>
<pml:CONSTRAINT>
.....
</pml:CONSTRAINT>

```

Figure 31: PML description of feature information of Figure 29

As seen before, dual representation scheme captures both intentional features and geometric features. Though redundancy requires more storage space in CAD systems, it is worthwhile in order to preserve the design procedure and construct history. Besides the association between features and low-level entities the relation between features (i.e., feature dependency) is also an important part of design history.

5.1.2 Feature dependency

There are two types of relations between features, *chronicle dependency* and *reference dependency*. Chronicle dependency records the construction process and history of design. It

captures feature operations in design step-by-step. In the example of Figure 30, cut feature *cut0* is added after protrusion *p0*, while hole feature *hole0* is built after cut *cut0*. The hierarchical structure of PML trees provides a convenient way to model the chronicle aspect of modeling. The sequence of child nodes of the PML tree node *FEATURE* shows the constructing sequence. Reference dependency occurs when previous low-level entities of posteriori features are referenced by new priori feature specification. Some local feature operations use entities generated from previous evaluation as part of their specification, such as thread, chamfer, and fillet. The reference dependency among features is captured in terms of the reference relation between entities in priori features and posteriori features. In the example of Figure 30, edge *e8* was generated by protrusion feature *p0*. When feature *fillet0* is defined, *e8* is part of the priori specification. Fillet *fillet0* is reference dependent on protrusion *p0*. Reference dependency can be retrieved in a DHG model using the algorithms in Figure 32 and Figure 33.

```

INPUT:  Directed Hyper Graph G = (V, E)
        Feature node Nfeat
OUTPUT: NodeList L containing feature nodes that are
        dependent of Nfeat

FOR each unmarked face node Nface with
  an association path to Nfeat
  run TEST on input <Nface>
  mark Nface
  FOR each unmarked edge node Nedge with
    an aggregation path from Nface
    run TEST on input <Nedge>
    mark Nedge
    FOR each unmarked vertex node Nvertex with
      an aggregation path from Nedge
      run TEST on input <Nvertex>
      mark Nvertex
    ENDFOR
  ENDFOR
ENDFOR

TEST: on input <N>
START TEST
  FOR each unmarked feature Nfeat0 that has
    an aggregation path to N
    L.add(Nfeat0)
    Mark Nfeat0
  ENDFOR
END TEST

```

Figure 32: Algorithm to list dependent features of a feature for reference dependency

```

INPUT:  Directed Hyper Graph G = (V, E)
        Feature node Nfeat
OUTPUT: NodeList L containing feature nodes that
        Nfeat depends on

FOR each topological node N with
  an aggregation path from Nfeat
  IF N has an association path to feature Nfeat0
    L.add(Nfeat0)
  ENDIF
ENDFOR

```

Figure 33: Algorithm to list features that a feature depends on for reference dependency

To summarize, features are important information about design intent and history, which are widely used in feature-based design. The UL-PML scheme is capable of capturing feature

information. The dual representation of priori and posteriori features allows global and local feature operations to be modeled, thus the feature definition is separated from the feature evaluation. Two dimensional dependency relations between features are captured as well to be part of design intent information.

5.2 Geometric Constraint Representation

Geometric constraints are fundamental relations needed to construct geometric shapes in a parametric or variational way. Current standard formats use the explicit modeling method, therefore geometric relations among entities are not captured explicitly. Rather, these relations are modeled implicitly. For example, if two lines are parallel, they have the same directional vectors instead of explicitly constrained with “parallel”. It is impossible to differentiate intentional parallel from accidental parallel. Further, if two directional vectors are (1.0, 0.0, 0.0) and (1.00000001, 0.0, 0.0), the question whether they are equal or not is system dependent. The small difference may be generated unintentionally because of numerical errors with floating-point arithmetic, or it may be intentionally specified by the designer.

To preserve design intent and maintain information integrity, it is essential that product data include geometric relations among entities, such as coincidence, concentric, parallelism, coplanar, and perpendicularity, such that these specifications and constraints can be recorded and transmitted. These relations should be modeled explicitly and included in current explicit modeling scheme. EDM [106, 116] classified constraints into three classes: predefined constraints which are common and well-known; free form constraints which are expressed by

string; and construct constraints which capture the construction process. But it did not show how constraints are modeled at entity level and relations are built to support parametric modeling.

Geometric constraints include a variety of types. The commonly used ones can be categorized as in Table 4. Geometric constraints are not mutually exclusive, which means that one constraint relation may be represented by another constraint. For example, perpendicularity can be represented as an angle of 90 degrees. This implies that constraint representation scheme should be flexible enough and extensible.

Table 4: Categories of common geometric constraints

Dimension	Position	Orientation	Symmetry	Tolerance
Distance	Fixed	Angle	Line symmetry	Dimension
Radius	Coincidence	Horizontal	Plane symmetry	Straightness
Diameter	Concentric	Vertical		Flatness
	Point on curve	Curve parallel		Circularity
	Curve on surface	Surface parallel		Cylindricity
	Curve tangent	Collinear		Of a line
	Surface tangent	Coplanar		Of a surface
		Perpendicular		Angularity
				Perpendicularity
				Parallelism
				Position
				Concentricity
				Circular runout
				Total runout

In the UL-PML scheme, geometric constraints are only modeled at the topological and geometric entity level, since form or shape is the major concern of geometric constraints. Each instance of a constraint is defined as a constraint entity. The unidirectional relation between a constraint and a topological or geometric entity is dynamic and represented as a path in DHG model. A constraint entity can have relations with one, two, or more topological/geometric entities.

There are two types of geometric constraints. One is *numerical* constraint, such as distance and angle, which gives numerical information; the other is *symbolic* constraint, such as coincidence and parallel, which gives logical information. In many geometric operations, the result of numerical computation must be used to infer symbolic facts. The geometric reasoning process thus depends on the precision of numerical values, which in turn depends on the system's error tolerance and computational algorithms. Different systems have different implementations, which causes errors during geometry interpretation.

Both symbolic and numerical constraints are modeled explicitly in the UL-PML scheme. The inclusion of symbolic constraints eliminates ambiguity and uncertainty, which specifies geometric relations semantically. For numerical constraints, an interval-value representation is proposed to specify allowance of numerical values to avoid inconsistency. From both aspects, the robustness of geometric computation can be improved.

5.2.1 Robustness in Geometric Computation

During geometric computation, numerical results about geometric entities are usually tested against specified constraints for verification and validation purposes. Numerically, 0.99999999 and 1.00000000 may be same in some systems but not in others. Similarly, an angle of 89.99999999 may be considered perpendicular in some systems but not so in others. Conceptually, geometric objects are within a *continuous* Euclidean space, yet they are modeled and computed within a *discrete* domain of computation. Representing an infinite number of real numbers by a finite number of bits requires approximation. In geometric computation, some geometric properties such as incidence, separation, tangency, and perpendicularity are derived based on numerical calculation. Similar to other numerical computation based on floating-point

arithmetic, geometric computation is not as accurate and reliable as we expect, especially when irrational numbers are involved.

The precision of binary representation in floating-point computation always has limits. The outcome of the computation thus might largely depend on the detailed algorithm implementation and sequence of calculations, which are highly system-dependent. Uncertainty is associated with approximate arithmetic computations, and different logical inferences may be made in different systems. Consequently, robustness is one of the interoperability issues between CAD systems.

The numerical errors may come from rounding or cancellation [125]. Not all decimal numbers can be represented in binary format exactly. For example, the decimal number 0.1 cannot be represented exactly but is approximately $1.1001100110011001101 \times 2^{-4}$ in floating-point format. This results in rounding errors. Multiplication operations generally require double number of bits for the arithmetic. After that, the results are rounded off to normal precision. This may generate rounding errors as well. When subtracting nearly equal quantities, the most significant digits in the operands match and cancel each other, which generates errors due to the cancellation. There are two kinds of cancellation: *catastrophic* and *benign*. Catastrophic cancellation occurs when the operands are subject to rounding errors. For example, consider $b = 3.34$, $a = 1.22$, and $c = 2.28$. The exact value of $b^2 - 4ac$ is 0.0292. But b^2 rounds to 11.2 and $4ac$ rounds to 11.1, hence the final answer 0.1 has a significant error, which is introduced by earlier multiplication. Benign cancellation occurs when subtracting exactly known quantities, which has small relative error.

The severity of the robustness problem in geometric computation has been studied by some researchers [126, 127, 128, 129, 130]. Three strategies have been proposed to improve robustness and consistency, which are exact arithmetic, symbolic reasoning, and reliable

computation. The exact arithmetic approach [131, 132, 133, 134, 135] uses exact numbers (e.g., integers) as necessary numerical values and symbolic computation on algebraic geometry to calculate other values with variable precision. The symbolic reasoning [136, 137] represents geometric entities and infers geometric relations symbolically, and no numerical calculation is involved; thus, consistency is maintained. The reliable computation [138, 139, 140, 141] uses interval arithmetic such that the exact real result of an arithmetic calculation is enclosed within a floating-point interval.

5.2.2 Interval-value numerical constraints

To improve the modeling robustness, an interval-value constraint scheme is proposed to specify numerical values. A numerical constraint is given by a lower bound and an upper bound. For instance, if a distance between two points are given in the format of lower and upper bound, it will allow CAD systems to interpret and validate constraints within certain error range.

Figure 34 shows an example of numerical errors. A regular polygon of 360 sides is built to inscribe a circle. Starting from vertex A, the coordinates of starting vertex and ending vertex of each side are calculated sequentially based on the previous calculated vertex. The starting vertex A of the first side is supposed to coincide with the ending vertex Z of the last side. But as the radius of the circle increases, a gap between A and Z appears and the gap is increased as the size of the circle increases. The coordinates of Z are listed in Table 5, where the coordinates of A are $(0.0, 0.0)$. If different systems have different error tolerances, inconsistent interpretation will be derived.

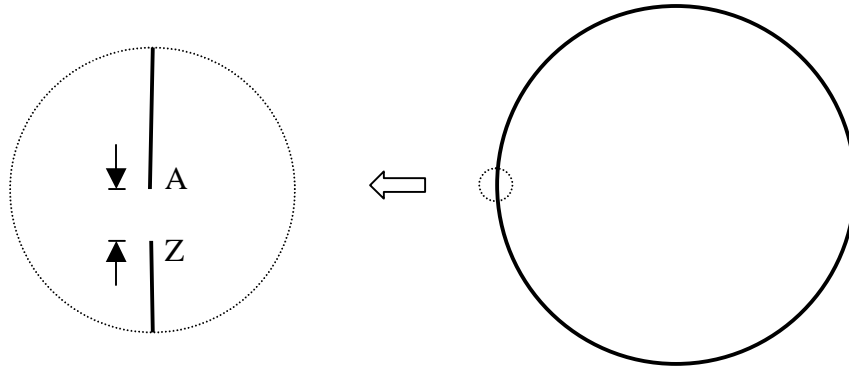


Figure 34: An example of numerical errors

Table 5: Coordinates of ending vertex with different radii

Point Z	Radius = 1	Radius = 100	Radius = 10000	Radius = 1000000
x=	0.000000	-0.000000	0.000000	0.000000
y=	-0.000000	-0.000000	-0.000004	-0.000359

Numerical errors caused by rounding and cancellation are inevitable. Thus a real value constraint, such as $distance = 10.0$, will not guarantee to be satisfied in different systems. To ensure symbolic meanings to be derived consistently from numerical results, some flexible allowances should be given as numerical constraints for consistent interpretation. In the previous example, if a coincidence constraint is given by $-0.000 \leq |\bar{p}_A - \bar{p}_Z| \leq 0.001$, i.e., an interval value $[-0.000, 0.001]$ is given in the distance constraint, the two vertices A and Z will be coincident with different radii of the circle. Interval values for numerical constraints increase the robustness of geometric computation.

Two types of intervals are considered in a numerical constraint. One is *trivial-width* interval, and the other is *non-trivial-width* interval. A trivial-width interval gives a narrow floating-point value bound to have the real value included within it. This interval gives an approximation of the real value that cannot be represented by floating-point values. The width of

the interval gives an estimate of floating-point precision. For example, real value 0.1 is represented by floating-point interval $[1.10011001100110011001100 \times 2^{-4}, 1.10011001100110011001110 \times 2^{-4}]$ that is equivalent to $[0.099999994, 0.100000009]$ in decimal.

A non-trivial-width interval has a wide bound and gives more allowance to entities. If a calculated value is within the interval, this constraint is satisfied. The interval value prevents topological inconsistency due to error propagation. The width of the interval gives the tolerance of errors. For example, in Figure 34, if the distance between A and Z is within the interval $[0.000, 0.001]$, coincidence can be derived. If the distance is not within the interval, the constraint is not satisfied. Inconsistency error then occurs. Interval-value constraints increase the robustness for constraint verification and validation.

There is a new issue generated during constraint verification and validation test when interval values are used. That is how to choose the proper width of an interval value. In the example of Figure 34, if the radius of the circle increases continuously, the distance value will go beyond the interval $[0.000, 0.001]$ and an inconsistency error will occur eventually. Choosing the width of an interval thus is a tricky part of imposing numerical constraints. There are two types of errors associated with choosing interval width. If the width of a constraint interval value is too small, most of the tests will fail because of numerical approximation, which generates unnecessary errors of inconsistency, which is called *Type I error*. If the width of a constraint value is too big, some of the tests that were supposed to fail now will pass, which generates unnecessary errors of inconsistency too, which is called *Type II error*. Choosing interval width of constraint values will be influenced by the uncertainty of application type, accuracy requirement, software system implementation, and computation hardware precision. It could largely depend on users' experiences.

To summarize, it is advantageous that numerical constraints are represented by interval values, which reduce the chances of inconsistency due to numerical errors, and symbolic constraints are represented in descriptive ways, which eliminate ambiguity and uncertainty. In the UL-PML scheme, specific constraint entities are defined using schema, thus geometric constraints can be included in an integrated product model.

Figure 35 gives some examples of modeling symbolic and numerical geometric constraint for a feature-based piston design. Geometric constraints include constraints in priori features such as the radius r within the profile of *revolve* feature in Figure 35 (b), constraints in posteriori features such as the distance d of the *cut* feature in Figure 35 (c), as well as inter-feature constraints such as *concentric* of faces $f1$ and $f2$ for the assembly in Figure 35 (d).

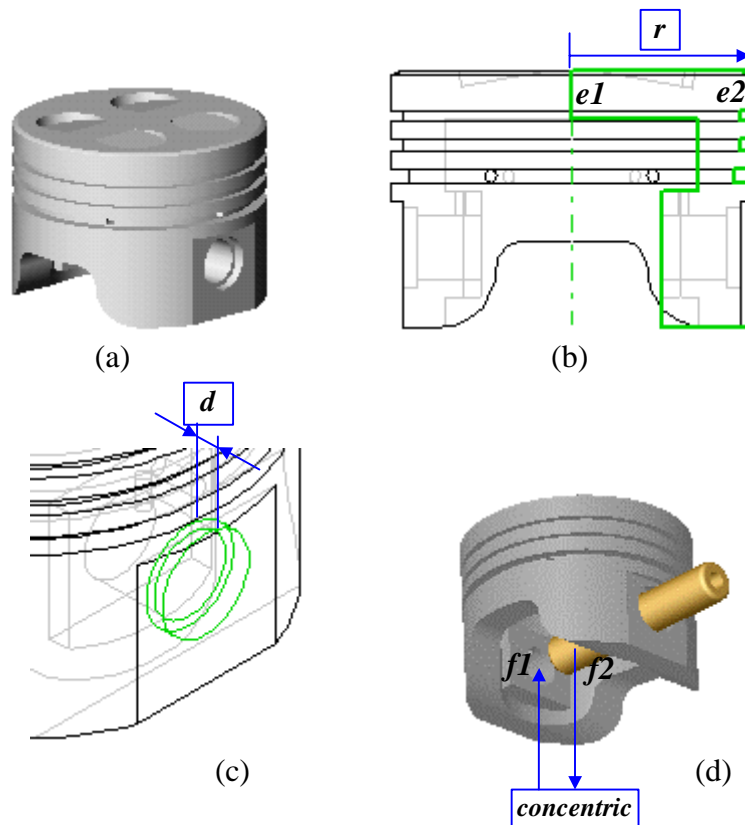


Figure 35: Constraint examples in a piston and its assembly

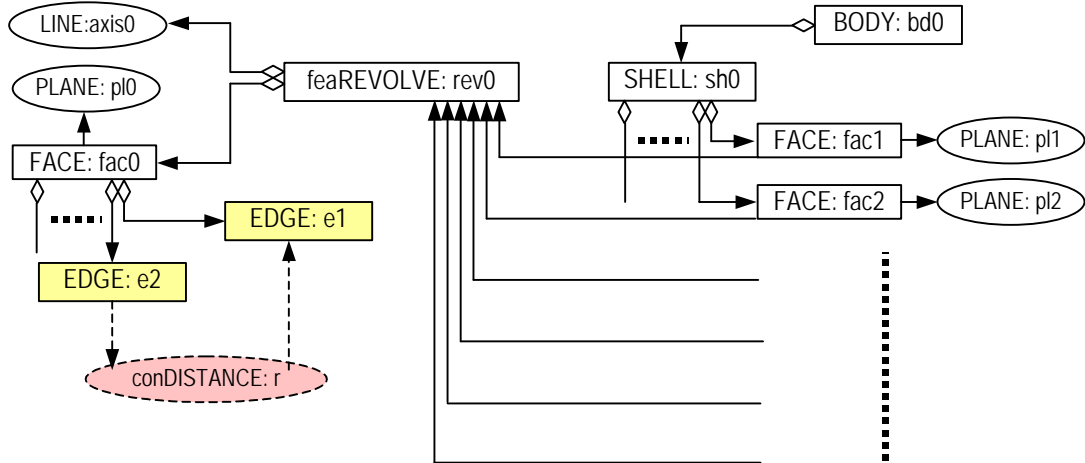
In the UL-PML scheme, this piston is modeled based on features including *revolve*, *cut*, *hole*, *pattern*, etc., as shown in Figure 36 (a). The constraints in Figure 35 (b), (c), and (d) are modeled in PML as in Figure 36 (b), (c), and (d) respectively. Symbolic constraints are represented by constraint entities while numerical constraints have interval value allowances in computation.

```

<pml:PART id="piston">
  <pml:GEOMETRY>
    .....
  </pml:GEOMETRY>
  <pml:TOPOLOGY>
    .....
  <pml:FACE id="f1">
    <pml:refLOOP xlink:type="simple" xlink:href="#lp0" xlink:show="embed" xlink:actuate="onLoad"/>
    <pml:refSURFACE xlink:type="simple" xlink:href="#pl0" xlink:show="embed" xlink:actuate="onLoad"/> </pml:FACE>
  <pml:EDGE id="e4">
    <pml:refCURVE xlink:type="simple" xlink:href="#l12" xlink:show="embed" xlink:actuate="onLoad"/> </pml:EDGE>
  <pml:RING id="r2">
    <pml:refEDGE xlink:type="simple" xlink:href="#e13" xlink:show="embed" xlink:actuate="onLoad"/> </pml:RING>
    .....
  </pml:TOPOLOGY>
  <pml:FEATURE>
    <pml:feaREVOLVE id="rev0" angle="360.0">
      <pml:PROFILE>
        <pml:refFACE xlink:type="simple" xlink:href="#f1" xlink:show="embed" xlink:actuate="onRequest"/> </pml:PROFILE>
      <pml:AXIS>
        <pml:refEDGE xlink:type="simple" xlink:href="#e4" xlink:show="embed" xlink:actuate="onRequest"/> </pml:AXIS>
    </pml:feaREVOLVE>
    <pml:feaCUT id="cut0" category="extrusion" type="blind" depth="12.0">
      <pml:PROFILE>
        <pml:refRING xlink:type="simple" xlink:href="r2" xlink:show="embed" xlink:actuate="onRequest"/> </pml:PROFILE>
      <pml:TRAJECTORY>
        <pml:refVECTOR xlink:type="simple" xlink:href="#t14" xlink:show="embed" xlink:actuate="onLoad"/> </pml:TRAJECTORY>
    </pml:feaCUT>
    .....
  </pml:FEATURE>
</pml:PART>

```

(a) features

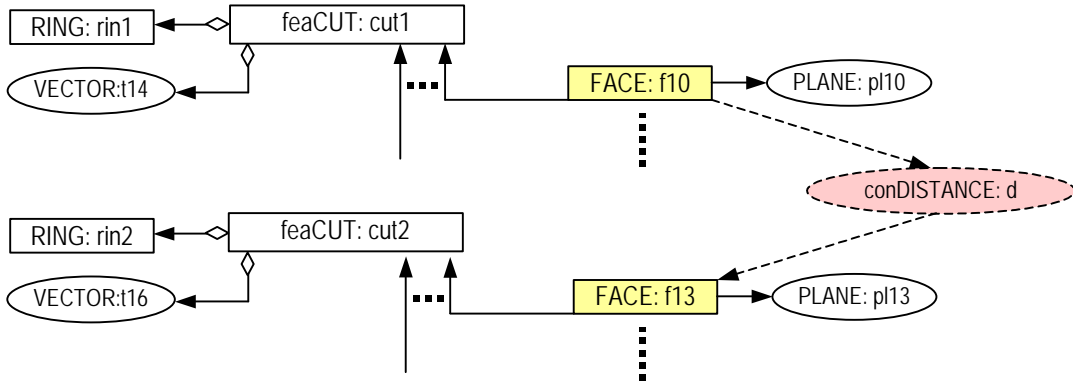


```

<pml:PART id="piston">
  <pml:GEOMETRY>
    .....
  </pml:GEOMETRY>
  <pml:TOPOLOGY>
    .....
    <pml:EDGE id="e1">
      <pml:refVERTEX xlink:type="simple" xlink:href="#v0" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVERTEX xlink:type="simple" xlink:href="#v1" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refCURVE xlink:type="simple" xlink:href="#line0" xlink:show="embed" xlink:actuate="onLoad"/> </pml:EDGE>
    <pml:EDGE id="e2">
      <pml:refVERTEX xlink:type="simple" xlink:href="#v2" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVERTEX xlink:type="simple" xlink:href="#v3" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refCURVE xlink:type="simple" xlink:href="#line1" xlink:show="embed" xlink:actuate="onLoad"/> </pml:EDGE>
    .....
  </pml:TOPOLOGY>
  <pml:FEATURE>
    .....
  </pml:FEATURE>
</pml:PART>
<pml:CONSTRAINT>
  .....
  <pml:conDISTANCE id="r" xlink:type="extended" pml:lowerBound="49.99998720" pml:upperBound="50.00012210">
    <pml:LOC1 xlink:type="locator" xlink:label="start" xlink:href="#e2"/>
    <pml:LOC2 xlink:type="locator" xlink:label="end" xlink:href="#e1"/>
    <pml:ARC1 xlink:type="arc" xlink:from="start" xlink:to="end" xlink:actuate="onRequest"/> </pml:conDISTANCE>
  .....
</pml:CONSTRAINT>

```

(b) *distance* constraint between edges

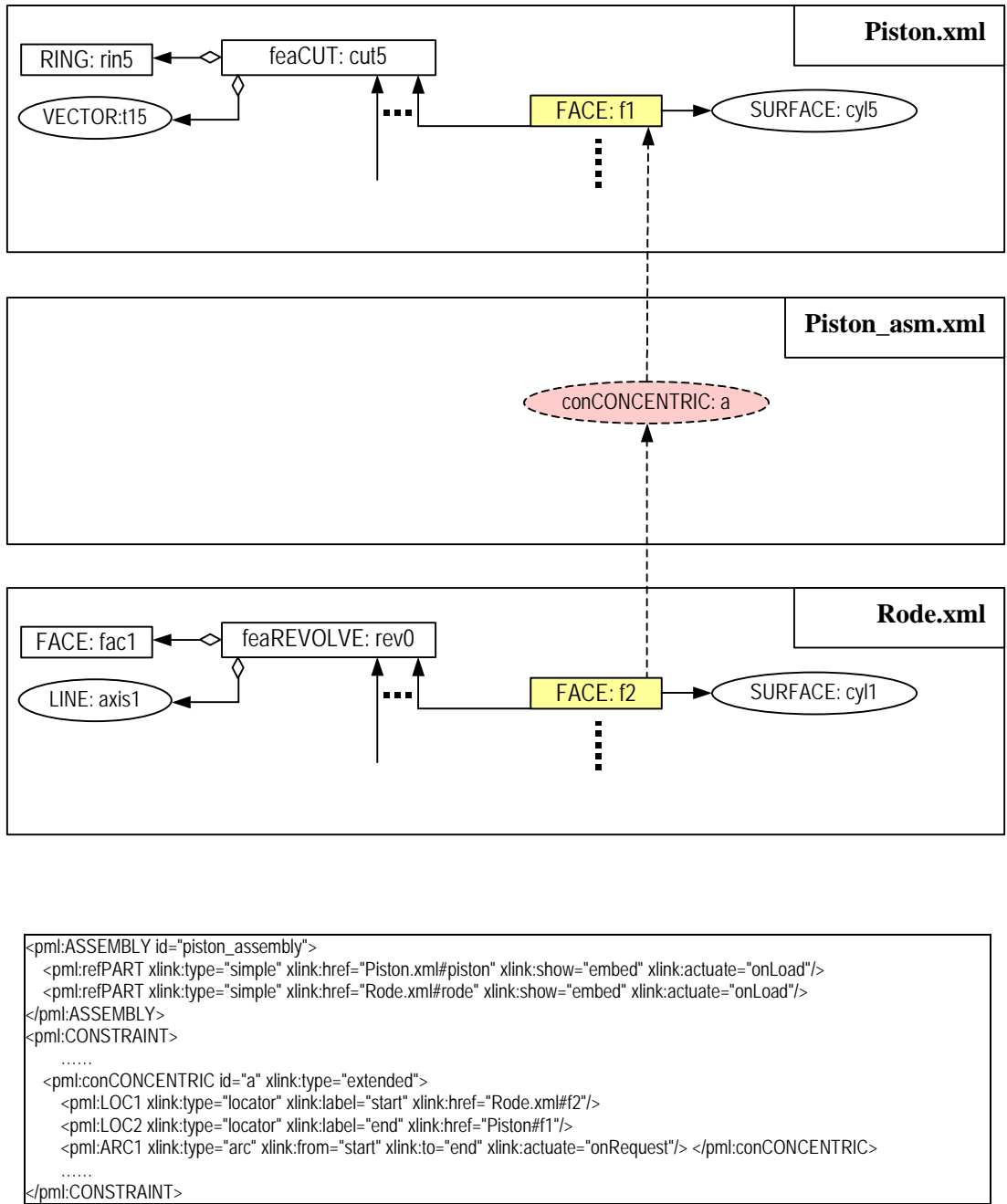


```

<pml:PART id="piston">
  <pml:GEOMETRY>
    .....
  </pml:GEOMETRY>
  <pml:TOPOLOGY>
    .....
    <pml:FACE id="f10">
      <pml:refLOOP xlink:type="simple" xlink:href="#p10" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refSURFACE xlink:type="simple" xlink:href="#plane10" xlink:show="embed" xlink:actuate="onLoad"/> </pml:FACE>
      .....
    <pml:FACE id="f13">
      <pml:refLOOP xlink:type="simple" xlink:href="#p13" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refSURFACE xlink:type="simple" xlink:href="#plane13" xlink:show="embed" xlink:actuate="onLoad"/> </pml:FACE>
      .....
    </pml:TOPOLOGY>
  </pml:FEATURE>
  .....
</pml:FEATURE>
</pml:PART>
<pml:CONSTRAINT>
  .....
  <pml:conDISTANCE id="d" xlink:type="extended" pml:lowerBound="3.99898720" pml:upperBound="4.00010210">
    <pml:LOC1 xlink:type="locator" xlink:label="start" xlink:href="#f10"/>
    <pml:LOC2 xlink:type="locator" xlink:label="end" xlink:href="#f13"/>
    <pml:ARC1 xlink:type="arc" xlink:from="start" xlink:to="end" xlink:actuate="onRequest"/> </pml:conDISTANCE>
    .....
  </pml:CONSTRAINT>

```

(c) distance constraint between faces

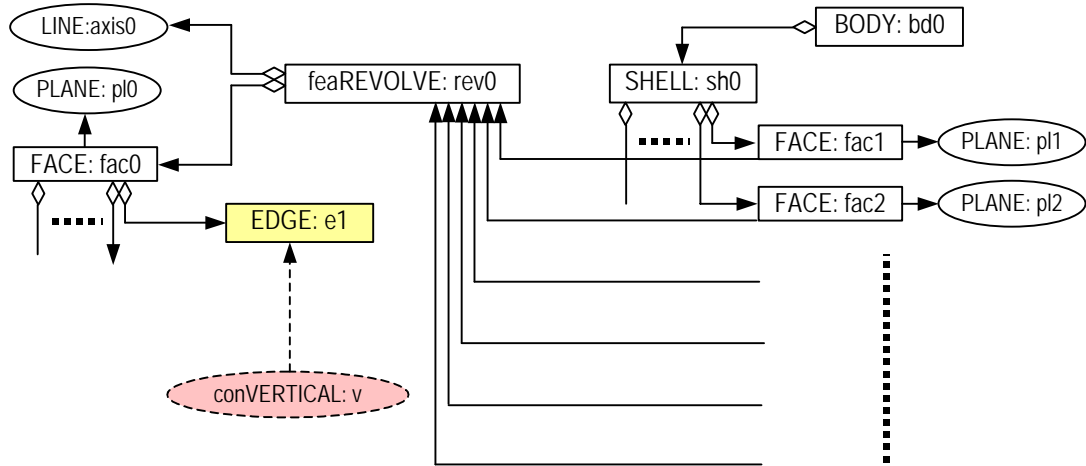


(d) concentric constraint between faces in assembly file

Figure 36: Piston features and geometric constraints in PML

The above constraints are associated with two entities, which are represented by extended links in PML. Constraints can also be associated with one entity. They are represented by simple

links in PML. For example, a *vertical* constraint of edge *e1* in Figure 35 (b) can be modeled as in Figure 37.



```

<pml:PART id="piston">
  <pml:GEOMETRY>
    .....
  </pml:GEOMETRY>
  <pml:TOPOLOGY>
    .....
    <pml:EDGE id="e1">
      <pml:refVERTEX xlink:type="simple" xlink:href="#v0" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refVERTEX xlink:type="simple" xlink:href="#v1" xlink:show="embed" xlink:actuate="onLoad"/>
      <pml:refCURVE xlink:type="simple" xlink:href="#line0" xlink:show="embed" xlink:actuate="onLoad"/> </pml:EDGE>
    .....
  </pml:TOPOLOGY>
  <pml:FEATURE>
    .....
  </pml:FEATURE>
</pml:PART>
<pml:CONSTRAINT>
  .....
  <pml:conVERTICAL id="v">
    <pml:refEDGE xlink:type="simple" xlink:href="#e1" xlink:show="embed" xlink:actuate="onRequest"/> </pml:conVERTICAL>
  .....
</pml:CONSTRAINT>

```

Figure 37: Simple link geometric constraint

5.3 Non-geometric Constraint Representation

Besides geometric constraints, there are a large number of constraints which have no geometric meanings themselves, such as material types, mathematical relations, and manufacturing process specification, as shown in Figure 26. These types of constraints may come from specifications or requirements of different design stakeholders, and are as important as geometric shape to maintain the quality of design. A large amount of design intent is transmitted by these non-geometric constraints, which unfortunately are unable to be captured and transferred along with the geometry by neutral format.

Some of the non-geometric constraints can be translated into geometric constraints. The geometric constraints are the reasoning results from the non-geometric ones. For example, the reliability constraint of a load-bearing shaft can be interpreted as the minimal diameter of the shaft should be greater than certain value, thus resulting in a diameter constraint with certain value. Nevertheless, it is still important to capture the reliability constraints, because some other constraints such as an assembly constraint may end up referring to the same diameter constraint. Therefore, it is critical that original non-geometric constraints be captured explicitly in the product model.

In the UL-PML scheme, non-geometric constraint entities are associated with high-level entities including feature, constraint, part, and assembly. To make it general, non-geometric constraints can be represented symbolically, which means that character strings are attached to entities of features, parts, and assemblies as supplemental information. Domain specific interpreters are needed to assist design system to understand the constraints. The taxonomy of non-geometric constraints is domain dependent. Constraint entities need to be defined for each application domain.

Based on the UL model, constraints can be associated with one or more entities. Some examples of non-geometric constraints are shown in Figure 38. A constraint can be a specific one, such as the *material* associated with the part *piston* and the *math* associated with three distance constraints *d*, *r*, and *l* in this example. It can also be a general one, such as the *op_temp* expressed in character string and associated with the part *piston*.

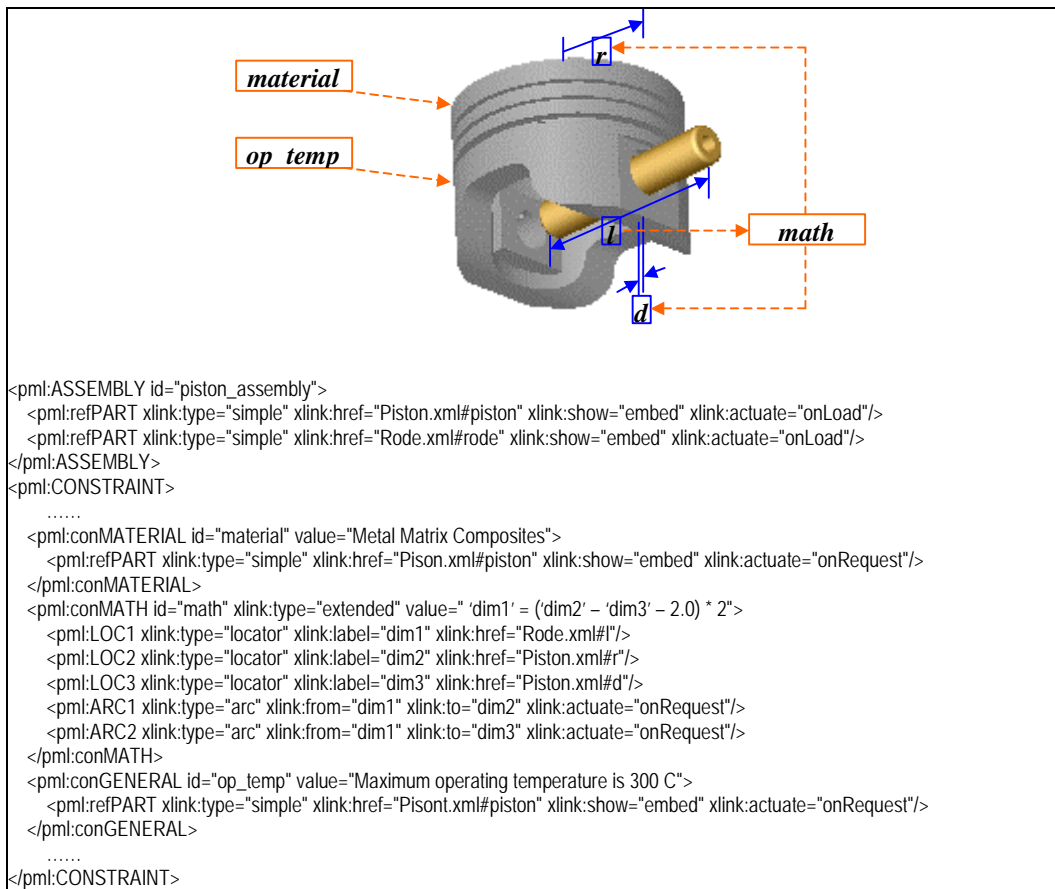


Figure 38: Examples of non-geometric constraints

5.4 Entity ID Persistency

One of the problems associated with feature-based parametric design is the naming persistency of entities, which has not been solved systematically. Topological entities are used as main references to trace geometry and other design information in most commonly used CAD systems, which are largely based on boundary representation. The name of a newly created topological entity is generated sequentially during the process of design. This new entity could be a reference to the new feature of the next step. If parameters assigned in previous steps are modified, or previous features are redefined, the parametric system needs to recreate the model. The change of a feature will directly affect the features that have reference dependencies on it during the model re-evaluation. As a result, some features at later steps may refer to a different entity unexpectedly, or even cannot find the reference. This naming persistency problem exists in current parametric solid modeling systems.

A typical example is shown in Figure 39 (a), where a part is constructed by a *protrusion* and a circular *cut* feature, followed by a *hole* feature. The position of the *hole* is partly determined by the distance s from the center of the *hole* to edge $e1$, which is generated by the *cut*. If the distances from the center of the *cut* to its references are changed, by either from b to d horizontally or from a to c vertically, as shown in Figure 39 (b) and (c) respectively, the distance reference of the *hole* to $e1$ will jump to edge $e2$. This is because the ID of the edge $e1$ was assigned to edge $e2$ after the Boolean operation of the *cut*, and the orientation information of edges is also used in the re-evaluation process. The direct effect of the naming persistency problem is that geometry re-evaluation generates an unexpected shape. It causes inconsistent and unpredictable geometry. The naming persistency problem affects the process of shape

construction within one modeling session. It can also affect the design process between modeling sessions.

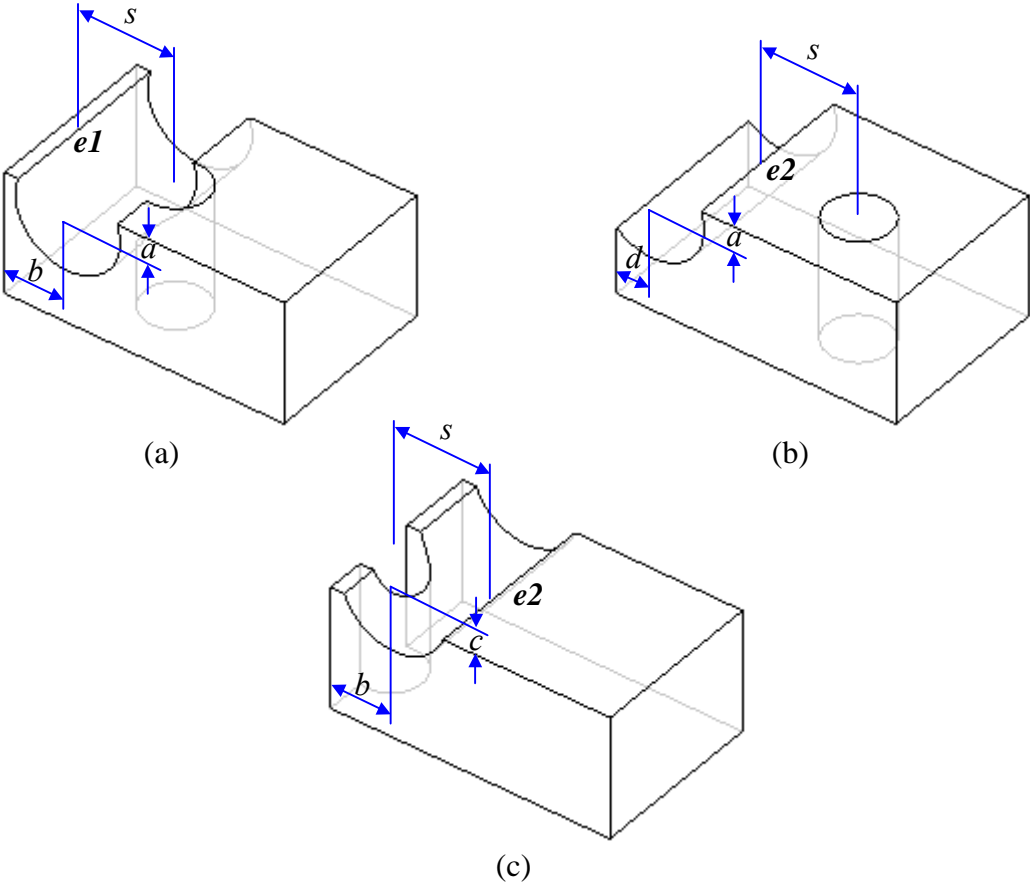


Figure 39: An example of naming persistency problem

In a PML-based distributed modeling environment, the persistency problem can easily cause inconsistency and unpredictability of modeling. The identification of entities is crucial to maintain the persistent and non-volatile linkage among different data files. Within a design session, relations between entities in one file should be sustained. Among different design sessions, linkages among files should be preserved as well. The issue of intra-session and inter-

session persistency should be resolved to allow the UL model to apply in a distributed design environment effectively.

Some heuristic solutions of the persistent naming issue have been proposed. In the research of E-REP [142], a topology-based naming method is used. New topological entities are named based on the referred old entities during the feature construction. For example, in an extrusion, a new edge is named by reference to the sweeping vertex, whereas a new face is named by reference to the sweeping edge. When model is re-evaluated, new entities should be identified and matched to old entities. The matching of an entity is realized through a local comparison of topological neighborhoods by a spectral graph isomorphism algorithm, as well as entities' orientation information. However, graph isomorphism is known to be NP-complete [143] and has combinatorial computational costs if a complex part is dealt with.

Comparatively, Kripac's topological ID system [144, 145] names a face based on a step ID (identifying the particular step that the face is created during the feature operations), a face index within that particular step, and the type of corresponding surface. Edges and vertices are identified by the names of adjacent faces. Each model maintains a face modeling history during the construction. To map the new entities to the old ones if the topology of the model is changed, this face modeling history is used during the comparison of the face graphs. Similarly, this approach involves time-consuming graph isomorphism procedures in each model reevaluation that is related to high cost of computation.

Wu *et al.* [146] identify faces by two names. The Original Name (ON) of a face records the feature's generating mode and the location of the face in the feature, while the Real Name (RN) of the face contains its ON and the parametric space information. New faces generated by Boolean operations will inherit the original faces' ONs. Edges and vertices are named only by

RNs, consisted of adjacent faces' RNs and parametric space information. The authors had a good observation to include parametric information of surfaces in topological IDs, but ended in the old trap of enumeration method to identify parameter values.

In the Boundary Representation scheme, geometry represents unbounded boundary information in Euclidean space, while topology is used to characterize coincidence and adjacency relations of bounded geometric elements. The latest geometric modelers have topology and geometry separately represented. In feature-based parametric modeling, topology is rather unstable and volatile. Small adjustment of some parameters may cause topology to be changed dramatically. This can be seen as the root of the naming persistency problem. The approach of identifying new entities by simply matching old topology to a new one for each re-evaluation is not a general solution from the computational efficiency point of view. A better solution is to include information that is more stable during the model construction into the identities of topological entities.

5.4.1 Parametric family

The basic technical problem of persistent naming is that a parametric solid model corresponds to a *class* of solids, but there is no formal definition or standard for what this class is [147]. While CSG models are globally parameterized, B-Rep models need extra boundary evaluation steps to apply parametric modeling, which causes the complexity of parametric family definition. In the work of Stewart [148] and Raghathama-Shapiro [149, 150], a parametric family of solids is defined based on topological mapping between cell complexes, that is, if any cell of B-Rep model K can be mapped to a cell of B-Rep model L , K belongs to the parametric family of L . This approach provides a necessary condition for boundary representation variance (BR-

variance) and parametric family classification. Nevertheless, sufficient conditions for BR-variance in parametric modeling still remain unresolved.

To generally define the parametric family of a solid, one needs to study sufficient conditions for BR-variance. A sufficient condition for BR-variance based on geometric continuity is proposed for a general definition of parametric family. Here, continuity means: throughout a valid parameter range, small changes in a solid's parameter values result in small changes in the geometry of B-rep (not "solid's representation" as in reference [149]). It is difficult, if not impossible, to organize variational / parametric families based on topology continuity. While adjacency of bounded geometric elements (topology) is volatile in the family of variational geometry, the unbounded geometric information (geometry) is more stable.

Poncelet's *continuity principle* states that if, from the nature of a particular problem, a certain number of solutions are expected, and if in any particular case this number of solutions is found, then there will be the same number of solutions in all cases, although some solutions may be imaginary [151, 152]. For instance, two circles intersect in two points, so it can be stated that every two circles intersect in two points, although the points may be imaginary or may coincide, as in Figure 40. If considered in a complex space instead of a real one, the loci of the two intersection points of the circles are continuous with respect to the distance between the centers of the circles.

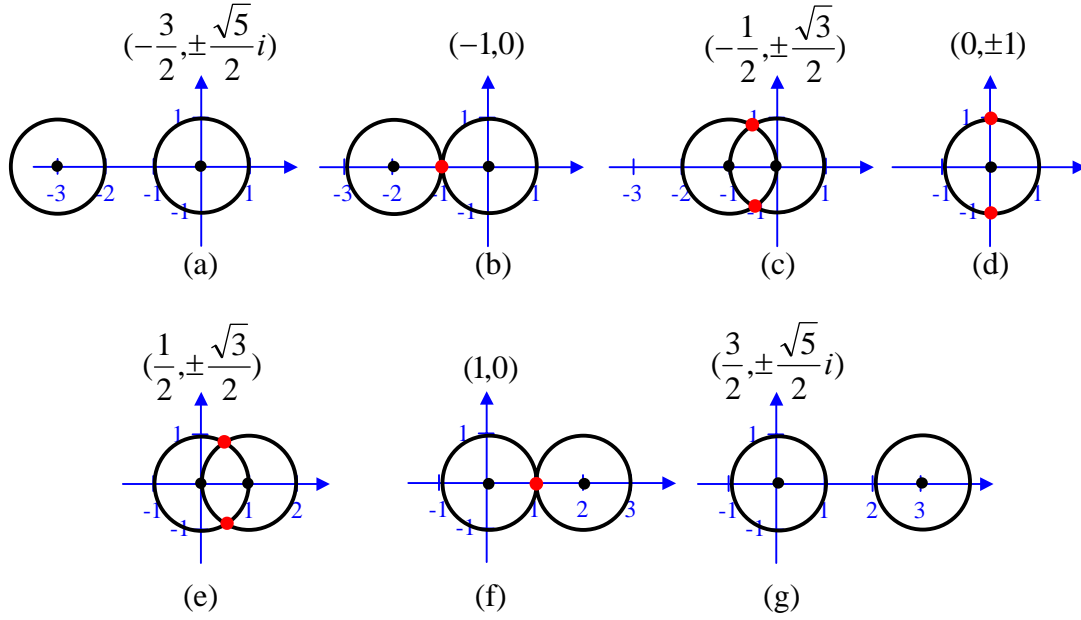


Figure 40: Example of intersect continuity

We extend Euclidean space to complex Euclidean space. In an even-dimensional Euclidean space \mathbf{R}^{2n} , points are ordered sets of $2n$ real numbers $(x_1, \dots, x_n, y_1, \dots, y_n)$, where $x_k, y_k \in \mathbf{R}$ ($k = 1, \dots, n$). If a complex structure is introduced as $z_k = x_k + iy_k$ ($k = 1, \dots, n$), we shall call the space whose points are ordered sets of n complex numbers

$$Z = (z_1, \dots, z_n) \tag{5.1}$$

the n -dimensional *complex Euclidean space*, denoted by \mathbf{C}^n .

For any point $p, p \in \mathbf{R}^n$, there is an infinite number of points q 's, $q \in \mathbf{C}^n$, such that there is a mapping function $f: \mathbf{C}^n \rightarrow \mathbf{R}^n, f(q) = p$. f is a function of orthogonal projection. The 3-dimensional Euclidean space \mathbf{E}^3 is the projected real subspace of complex Euclidean space \mathbf{C}^3 , where $f(q) = \text{Re}(q)$.

In the domain of parametric design, adding p more dimensions which represents real parameter t_j 's ($t_j \in \mathbf{R}, j=1, \dots, p$) to \mathbf{C}^3 , we have a $p \times 3$ dimensional *parametric complex Euclidean*

space denoted by $\mathbf{P}^p\mathbf{C}^3$, where $\mathbf{P}^p\mathbf{C}^3 = \mathbf{R}^p \times \mathbf{C}^3$. There are two types of parameters, *shape* parameters (s -parameters) and *relation* parameters (r -parameter), associated with each geometric object. For example, in a planar circle

$$\begin{cases} x = a + r \cos \theta \\ y = b + r \sin \theta \end{cases}, \quad (5.2)$$

θ is an s -parameter and a, b, r are r -parameters. A $\mathbf{P}^p\mathbf{C}^3$ space including m -dimensional s -parametric subspace and n -dimensional r -parametric subspace can be further defined as $\mathbf{P}^{m \times n}\mathbf{C}^3 = \mathbf{R}^m \times \mathbf{R}^n \times \mathbf{C}^3$. The BR-variance and continuity for parametric family are defined in $\mathbf{P}^p\mathbf{C}^3$.

Definition 5.1: A curve in \mathbf{C}^3 is a map $\gamma: \mathbf{R} \rightarrow \mathbf{C}^3$, denoted as $\gamma(t)$, where t ($t \in \mathbf{R}$) is an s -parameter of γ . In $\mathbf{P}^p\mathbf{C}^3$, a curve is a hyper-curve, $\gamma: \mathbf{R} \rightarrow \mathbf{R}^{(p-1)}\mathbf{C}^3$, where $p \geq 1$.

Definition 5.2: A surface in \mathbf{C}^3 is a map $\sigma: \mathbf{R}^2 \rightarrow \mathbf{C}^3$, denoted as $\sigma(u, v)$, where u and v ($u, v \in \mathbf{R}$) are s -parameters of σ . In $\mathbf{P}^p\mathbf{C}^3$, a surface is a hyper-surface, $\sigma: \mathbf{R}^2 \rightarrow \mathbf{R}^{(p-2)}\mathbf{C}^3$, where $p \geq 2$.

Definition 5.3: A curve $\gamma(t) \in \mathbf{P}^{(p-1)}\mathbf{C}^3$ ($t \in \mathbf{R}$) is called C^0 continuous with respect to t in the neighborhood of t_0 if and only if $\lim_{t \rightarrow t_0} \gamma(t) = \gamma(t_0)$.

Definition 5.4: A curve $\gamma(t) \in \mathbf{P}^{(p-1)}\mathbf{C}^3$ ($t \in \mathbf{R}$) is called C^k continuous with respect to t in the neighborhood of t_0 if and only if $\lim_{t \rightarrow t_0} \frac{\partial \gamma^k(t)}{\partial t^k} = \frac{\partial^k \gamma(t_0)}{\partial t^k}$, and $\gamma(t)$ is C^{k-1} continuous.

Definition 5.5: A surface $\sigma(u, v) \in \mathbf{P}^{(p-2)}\mathbf{C}^3$ ($u, v \in \mathbf{R}$) is called C^0 continuous with respect to u and v in the neighborhood of (u_0, v_0) if and only if $\lim_{\substack{u \rightarrow u_0 \\ v \rightarrow v_0}} \sigma(u, v) = \sigma(u_0, v_0)$.

Definition 5.6: A surface $\sigma(u, v) \in \mathbf{P}^{(p-2)}\mathbf{C}^3$ ($u, v \in \mathbf{R}$) is called C^k continuous with respect to u and v in the neighborhood of (u_0, v_0) if and only if $\lim_{u \rightarrow u_0} \frac{\partial \sigma^k(u, v)}{\partial^k u} = \frac{\partial^k \sigma(u_0, v)}{\partial^k u}$,

$$\lim_{v \rightarrow v_0} \frac{\partial \sigma^k(u, v)}{\partial^k v} = \frac{\partial^k \sigma(u, v_0)}{\partial^k v}, \quad \lim_{u \rightarrow u_0} \frac{\partial \sigma^k(u, v)}{\partial^{k-1} v \partial u} = \frac{\partial^k \sigma(u_0, v)}{\partial^{k-1} v \partial u}, \quad \lim_{v \rightarrow v_0} \frac{\partial \sigma^k(u, v)}{\partial^{k-1} u \partial v} = \frac{\partial^k \sigma(u, v_0)}{\partial^{k-1} u \partial v}, \quad \text{and}$$

$\sigma(u, v)$ is C^{k-1} continuous.

Definition 5.7: The set of *bounding surfaces* of a solid object o in space $P^p C^3$, $bs(o)$, is a set of surfaces, $\forall \sigma \in bs(o)$, $\exists p \in \sigma$, such that $\exists a, \exists b$, $a \in \varepsilon$ -neighborhood of p , $b \in \varepsilon$ -neighborhood of p , while $a \in o$, $b \notin o$.

Definition 5.8: The set of *bounding curves* of a solid object o in space $P^p C^3$, $bc(o)$, is the set of curves, each of which is the intersection of two bounding surfaces, i.e., $\forall \gamma, \gamma \in bc(o)$, such that $\exists \sigma, \exists \delta, \sigma \in bs(o), \delta \in bs(o)$, for $\forall p, p \in \gamma$, at the same time, $p \in \sigma, p \in \delta$.

In $P^p C^3$ space, two curves always intersect, either at real points, imaginary points, or infinity. If two curves have an r -parameter r , the locus of intersection of the curves is a curve with r as its s -parameter. Similarly, if two curves have r -parameters q and r , the locus of intersection of the curves is a surface with q and r as its s -parameters.

Definition 5.9: An *intersecting curve* with respect to r ($r \in \mathbf{R}$) of two curves $\gamma(s)$ and $\xi(t)$, $\chi(\gamma(s), \xi(t), r)$, is a curve of r , where $\forall p, p \in \chi(\gamma(s), \xi(t), r)$, at the same time, $p \in \gamma, p \in \xi$.

Definition 5.10: An *intersecting surface* with respect to q and r ($q, r \in \mathbf{R}$) of two curves $\gamma(s)$ and $\xi(t)$, $\chi(\gamma(s), \xi(t), q, r)$, is a surface of q and r , where $\forall p, p \in \chi(\gamma(s), \xi(t), q, r)$, at the same time, $p \in \gamma, p \in \xi$.

Definition 5.11: A solid object o is called C^0 continuous with respect to an r -parameter r within interval (a, b) in space $P^p C^3$, if $\forall \gamma(s), \forall \xi(t), \gamma(s) \in bc(o), s \in \mathbf{R}, \xi(t) \in bc(o), t \in \mathbf{R}$, such that $\chi(\gamma(s), \xi(t), r)$ ($r \in \mathbf{R}$) is C^0 continuous with respect to r on $r \in (a, b)$.

Definition 5.12: A solid object o is called C^0 continuous with respect to r -parameters q and r within interval $(a_1, b_1) \times (a_2, b_2)$ in space $\mathbf{P}^p\mathbf{C}^3$, if $\forall \gamma(s), \forall \xi(t), \gamma(s) \in \text{bc}(o), s \in \mathbf{R}, \xi(t) \in \text{bc}(o), t \in \mathbf{R}$, such that $\chi(\gamma(s), \xi(t), q, r)$ ($q, r \in \mathbf{R}$) is C^0 continuous with respect to q and r on $q \in (a_1, b_1), r \in (a_2, b_2)$.

If a solid object o can be transformed to another solid object b with C^0 continuity with respect to an r -parameter r , b belongs to the parametric family of o with respect to r . Similarly, if a solid object o can be transformed to another solid object b with C^0 continuity with respect to r -parameters q and r , b belongs to the parametric family of o with respect to q and r . High-order parametric families can be defined in a similar way.

It is noted that a parametric family should be defined with respect to r -parameters. Definition 5.11 and 5.12 give the sufficient condition of BR-variance. If a solid has the property of C^0 continuity on certain intervals of r -parameters, the variance of boundary representation can be asserted.

In brief, if solid geometry is considered in parametric complex Euclidean space, the parametric family of a solid can be defined based on the continuity of geometry. Rather than topology, unbounded geometry possesses good properties of continuity. This leads to the ideal of identifying topological entities with geometry, which is called the semantic ID method introduced in the following section.

5.4.2 Semantic ID

To resolve the issue of naming persistency, a *semantic ID* scheme is proposed. The intention is to include information of construct relation in geometric IDs and geometric meanings of the identification in the topological IDs. The problem of simple enumeration of entity IDs is

that entity identification is exposed globally for the whole product structure. The data structure of enumeration is simply a link list. If one node is inserted or removed, all nodes following it should be renumbered. Any change within the sequence will affect the identification of all following entities. Therefore, it is more protective if ID assignments are localized.

The concept of *namespace* of an entity ID is introduced here. If a group of entities have some common properties, these properties can form a boundary for their names, and a prefix based on these properties can be attached on each of these entity IDs. In this way, the namespace of entities is divided based on the prefix. Simply from the name of an entity, some characteristics of the entity can be inferred. Re-evaluating some entities in one namespace does not affect the names of entities in other namespaces. The namespace can be organized in a hierarchical structure. One namespace can be divided further into multiple subspaces with an extra layer of prefixes in the names so on and so forth, thus forming a tree structure of naming.

A feature is a natural selection for the boundary of the namespaces. The ID of a newly created geometric or topological entity will be prefixed with the ID of the feature during which this feature operation is performed. The namespace of features categorizes entities based on construct history, and is the first step to isolate entity creation and identification. For example, each entity that is created during the constructing of the first protrusion will have *Protrusion1::* at the beginning of the entity's name.

The namespace of one feature could be partitioned further to differentiate priori and posteriori features. A priori feature may have multiple steps to finish the feature definition. Each step then can be assigned an independent sub-namespace. For example, a protrusion feature operation needs two steps to finish. One is defining profile, and another is the trajectory definition. Each entity generated at each step is prefixed by the feature step ID. The entities

generated when the profile of the protrusion is defined will have *Protrusion1::Profile::* as part of the IDs. The entities created when the trajectory is defined will have a prefix *Protrusion1::Trajectory::* in their IDs. Since entities defined in priori features are independent from entities generated in posteriori features, new entities created in priori features are free of turmoil from feature re-evaluation. Thus, enumeration in priori features will not cause big problems.

Major issues of naming persistency come from topological entities created in the domain of posteriori features. For each of these entities, no feature steps are included in the entity names. Within the namespace of each feature, entities should be named in a more meaningful and stable way instead of simple enumeration. One consideration is to include stable geometric information of the entities in their identification. The question is what kind of geometric information is to be included in topological IDs. A general way is to include all geometric information of the entity. For example, an edge is named by the combination of feature ID, feature step, curve type, starting / ending directional vectors, starting / ending points, surfaces it belongs to, etc.; and a vertex is named by the combination of feature ID, feature step, curves it belongs to, coordinates, etc. However, this will be a cumbersome procedure to record each topological entity. If one value of geometry attributes is changed, the ID should be updated in time. A more feasible version is to name a topological entity by including the ID of the corresponding geometric entity, thus references of geometric entities are embedded in topological entities' IDs.

To improve geometric entities' naming stability further, information of construct relations of geometry is included in geometric IDs. Because surfaces are generally much more stable than curves and points, curves and points are named by surfaces. For 2-manifold geometry, a curve is formed by two intersecting surfaces, while a point is formed by three intersecting surfaces. Thus,

a curve is named by the IDs of the two intersecting surfaces, and a point is named by the IDs of the three intersecting surfaces. This naming method takes a general and passive approach to identify curves and points, compared to enumeration that is a direct and active approach. This general approach may require more computation for identification of curves and points. Nevertheless, the ID contains extra construct information about face-based boundary, which is a desirable property.

Besides the reference to its corresponding geometric entity, a topological entity should also include boundary relations with other geometric entities in its ID. By including boundary information in terms of geometry, the topological IDs contain the actual semantics of topology. For example, in Figure 41, if a face is generated by a protrusion feature $p1$ and is referring a surface $s1$ and bounded by planes $s2$, $s3$, $s4$, and $s5$, this face will have the name $FACE(PROTRUSION(p1)::SURFACE(s1) + SURFACE(s2)\&SURFACE(s3)\&SURFACE(s4)\&SURFACE(s5))$. And the edge that is referring the line formed at the intersection of planes $s1$ and $s2$ will have the name $EDGE(PROTRUSION(p1)::CURVE(SURFACE(s1)\&SURFACE(s2)) + SURFACE(s3) + SURFACE(s5))$. A face ID has the references of the feature namespace, the corresponding surface, and the bounding surfaces if there are any. Similarly, an edge ID has the references of the feature namespace, the corresponding curve, and the bounding surfaces if there are any. There are some special geometry curves and surfaces that do not have intrinsic boundaries in B-Rep, such as circles and spheres. In these cases, extra boundary entities shall be introduced in order to identify topological entities. Features and surfaces can be named based on enumeration because of their relative stableness.

Until now, we assume that two surfaces intersect at one curve. For polyhedrons, faces are corresponding to planes, which is the simplest case. If some faces are corresponding to quadratic

or higher order surfaces. The assumption is not always true. Figure 42 illustrates some examples of intersecting surfaces. For linear surfaces intersection (plane-plane), a line (as in Figure 42-a) is generated. For a linear surface intersecting with a quadratic surface, either one curve (as in Figure 42-b, c, d, e, f) or two curves (as in Figure 42-g, h, i) will be generated. For higher-order surface intersections, one or two curves (as in Figure 42-k, l, m, n, o) will be generated. One exception is the special case that a plane intersects a cubic cylinder or even higher order at three or more parallel lines (as in Figure 42-j).

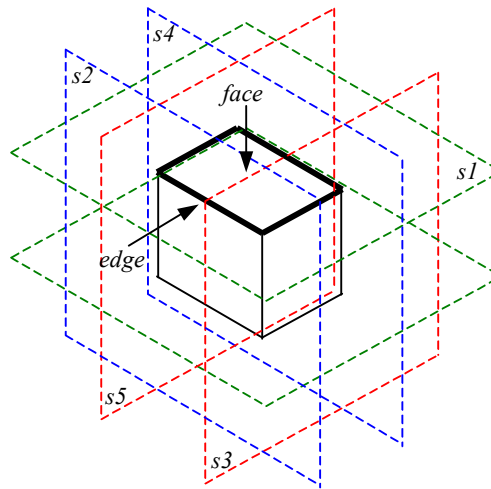


Figure 41: An Example of face bounded by surfaces and edge bounded by surfaces

The issue of how to distinguish curves and points if two surfaces intersect at two or more curves thus arises. Further, even if only one intersecting curve is generated, boundary surfaces may divide the curve into two or more edges. To identify curves and edges based on surfaces, extra information is needed if ambiguity exists. For parametric surfaces, curves can be identified based on the parameter ranges. But not all surfaces have parametric forms, whereas surfaces in parametric forms can be transformed to implicit forms. A general method is needed for surfaces in implicit forms.

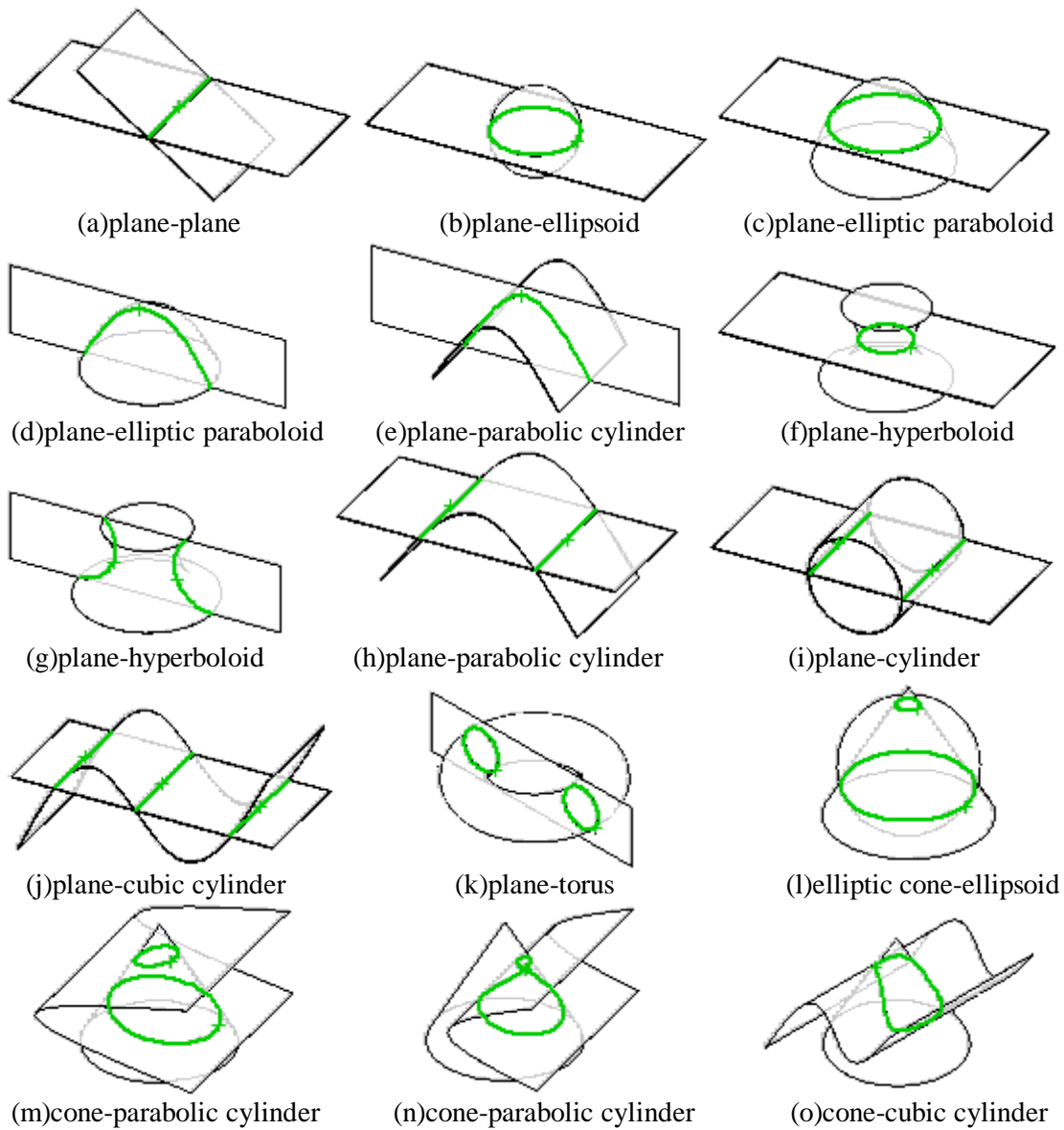


Figure 42: Examples of intersecting surfaces

One consideration is to add orientation information of curves, which is motivated by the concept of edge identification for non-parametric curves [153]. If a k^{th} gradient operator ∇^k in Cartesian coordinates is defined as

$$\nabla^k = \left[\frac{\partial^k}{\partial x^k} \quad \frac{\partial^k}{\partial y^k} \quad \frac{\partial^k}{\partial z^k} \right]^T \quad (k > 0), \quad (5.3)$$

a k^{th} gradient of the surface $f(x, y, z)=0$ at point $\mathbf{p} = (x, y, z)$ is

$$\mathbf{I}_\sigma^k(f, \mathbf{p}) = \nabla^k f(x, y, z) \quad (k > 0). \quad (5.4)$$

The *orientation* of the surface $f(x, y, z)=0$ at point $\mathbf{p} = (x, y, z)$ can be defined as the 1st gradient

$$\mathbf{I}_\sigma(f, \mathbf{p}) = \nabla f(x, y, z). \quad (5.5)$$

Let $f(x, y, z) = 0$ and $g(x, y, z) = 0$ be two surfaces intersecting at $c = \{(x, y, z) \mid f(x, y, z) = 0, g(x, y, z) = 0\}$. And the orientation of the curve c at point $\mathbf{p} = (x, y, z)$ is defined as

$$\mathbf{I}_{11}(f, g, \mathbf{p}) = \mathbf{I}_\sigma(f, \mathbf{p}) \times \mathbf{I}_\sigma(g, \mathbf{p}). \quad (5.6)$$

If the orientations of the intersecting curves at some interior points are included, edges can be identified. A simple way is to include the orientation information of bounding points of the curves. For example, in Figure 42-g, plane $y = 0$ intersects hyperboloid $x^2 + y^2 - z^2 - 1 = 0$, and two intersecting curves are bounded by planes $z + 1 = 0$ and $z - 1 = 0$. The orientations of two ending points for the left curve are $[\pm 2, 0, 2\sqrt{2}]^T$, and $[\pm 2, 0, -2\sqrt{2}]^T$ for the right curve, if the orientation is defined as the cross product of normal vectors for the plane and the hyperboloid. In Figure 42-h, plane $z = 0$ intersects with parabolic cylinder $x^2 + z - 1 = 0$, and two intersecting lines are bounded by planes $y + 1 = 0$ and $y - 1 = 0$. The orientation of two ending points for the left line is $[0, -2, 0]^T$, and $[0, 2, 0]^T$ for the right line, if the orientation is defined as the cross product of normal vectors for the plane and the hyperboloid. Here, the sequence of the vector product is important in the definition of orientation. If the positions of f and g in (5.6) are switched, the orientation will have opposite direction. If the orientations of the curves at two ending points are the same, orientations at some other corresponding points on the curves should

be derived to differentiate the two curves. If two surfaces are tangent at some points, the orientations of intersection curves at these points are zero vectors.

Extra care should be given to the exceptional cases that have three or more intersecting curves and two curves have same orientation information, such as in Figure 42-j. Plane $z = 0$ intersects with cubic cylinder $x^3 - x - z = 0$. The orientation of the left line at any point and the orientation of the right line at any point are always $[0, 2, 0]^T$, and $[0, -1, 0]^T$ for the middle line, if the orientation is defined as the cross product of normal vectors for the plane and the cubic cylinder. In this case, additional information besides orientation is needed to identify the left and the right edge. One can include second-order gradients of surfaces or curves as the supplementary information of curve orientation for edge identification.

The *adaptation* of the surface $f(x, y, z)=0$ at point $\mathbf{p} = (x, y, z)$ is the second-order gradient

$$\mathbf{I}_\sigma^2(f, \mathbf{p}) = \nabla^2 f(x, y, z). \quad (5.7)$$

The adaptation of the intersection curve by surfaces f and g can be defined as

$$\mathbf{I}_{12}(f, g, \mathbf{p}) = \mathbf{I}_\sigma(f, \mathbf{p}) \times \mathbf{I}_\sigma^2(g, \mathbf{p}) \quad (5.8a)$$

$$\mathbf{I}_{21}(f, g, \mathbf{p}) = \mathbf{I}_\sigma^2(f, \mathbf{p}) \times \mathbf{I}_\sigma(g, \mathbf{p}) \quad (5.8b)$$

$$\mathbf{I}_{22}(f, g, \mathbf{p}) = \mathbf{I}_\sigma^2(f, \mathbf{p}) \times \mathbf{I}_\sigma^2(g, \mathbf{p}) \quad (5.8c)$$

When orientation of curve cannot differentiate the intersection curves, either adaptations of surfaces or curves need to be included. In the previous example, the adaptation of the cubic cylinder is $[-6, 0, 0]^T$ at any point on the left intersecting line and is $[6, 0, 0]^T$ at any point on the right intersecting line. With the second-order gradients, these two curves can be identified even though curve orientations are equal.

If the adaptations of surfaces or curves still cannot differentiate the curves (e.g., in higher-degree surfaces), higher order gradients can be derived further to identify edges. This method can

also be extended beyond surfaces in implicit format. If some surfaces cannot be represented in closed form, they can be interpolated and approximated in polynomial forms, or in pragmatic sample data forms. The gradients and orientations can be approximated, which makes this ID format generally acceptable.

Similar to curve and edge identification, points or vertices are identified by the orientation/adaptation/gradient information of the intersecting curve of the first two surfaces at the particular positions if multiple curves or edges are generated by the same set of surfaces.

In summary, topological entities can be identified based on surfaces in evaluated solid geometry. Faces are named by the IDs of corresponding surfaces with bounding surfaces. Edges are named by the IDs of corresponding curves with bounding surfaces and extra orientation and gradient information of curves at boundary points if necessary, because it is possible that several edges are corresponding to one curve and same boundary surfaces. Curves are named by the IDs of intersecting surfaces, as well as additional orientation and gradient information about the involved surfaces at some points (e.g., the intersection points between a plane and the curves) if necessary, because it is possible that several curves are generated by intersecting surfaces. Vertices are named by the IDs of corresponding points, which in turn are named by the IDs of intersecting three or more surfaces. The syntax of topological and geometric entities' IDs is shown in Figure 43. Note that the curve orientation and gradients for a curve name are derived based on the sequence of surfaces shown in its surface list in the first segment.

```

<feature_id> ::= <feature_type> ( <feature_name> ) |
                <feature_type> ( <feature_name> ) :: <feature_step_name>
<face_id> ::= <face_type> ( <feature_id> :: <face_name> )
<edge_id> ::= <edge_type> ( <feature_id> :: <edge_name> )
<vertex_id> ::= <vertex_type> ( <feature_id> :: <vertex_name> )
<face_name> ::= <surface_id> | <surface_id> + <surface_list>
<edge_name> ::= <curve_id> | <curve_id> + <surface_list> |
                <curve_id> + <surface_list> - ( <additional_curve_info> )
<additional_curve_info> ::= <curve_orientation> & <curve_orientation> |
                <curve_orientation> & <curve_orientation> -
                <curve_adaptation> & <curve_adaptation>
<vertex_name> ::= <point_id>
<surface_list> ::= <surface_id> | <surface_id> & <surface_list>
<surface_id> ::= <surface_type> ( <surface_name> )
<curve_id> ::= <curve_type> ( <curve_name> )
<point_id> ::= <point_type> ( <point_name> )
<curve_name> ::= <surface_id> & <surface_list> |
                <surface_id> & <surface_list> - ( <additional_surface_info> )
<additional_surface_info> ::= <surface_orientation> & <surface_orientation> |
                <surface_orientation> & <surface_orientation> -
                <surface_adaptation> & <surface_adaptation>
<point_name> ::= <surface_id> & <surface_id> & <surface_list> |
                <surface_id> & <surface_id> & <surface_list> -
                ( <additional_point_info> )
<additional_point_info> ::= <curve_orientation> |
                <curve_orientation> - <curve_adaptation>
<feature_type> ::= <global_feature_type> | <local_feature_type>
<global_feature_type> ::= PROTRUSION | CUT | HOLE | LOFT | ...
<local_feature_type> ::= FILLET | CHAMFER | THREAD | ...
<face_type> ::= FACE
<edge_type> ::= EDGE
<vertex_type> ::= VERTEX
<surface_type> ::= PLANE | QUADRATIC_SURFACE | CUBIC_SURFACE |
                QUARTIC_SURFACE | FREE_FORM_SURFACE
<curve_type> ::= LINE | QUADRATIC_CURVE | CUBIC_CURVE |
                QUARTIC_CURVE | FREE_FORM_CURVE
<point_type> ::= POINT

```

Figure 43: Syntax of IDs for topological and geometric entities

5.4.3 Curve, Edge, and Point Mapping

The IDs of curves, edges, and points may consist of two segments (i.e., surface segment and orientation/adaptation/gradient segment). The first segment is rather stable because the unbounded surface geometry is independent of topological faces. Even if a face is eliminated from a solid, the geometry of a surface still exists in Euclidean space. The second segment, which contains vector values, may be changed each time when geometry is altered. That is, orientations, adaptations, and higher-order gradients of curves and surfaces at edges' boundary points and inner points may be changed if the geometry of some surfaces is modified. As a result, curves, edges, and points of newly generated solids need to be mapped to entities of old solids for each feature modification and re-evaluation.

The mapping here is based on geometric properties instead of topological correspondence in references [142, 145]. We simply call the curve, edge, or point ID in the old solid before modification *old ID*, and the ID of its counterpart in the new solid after modification *new ID*. The surface (first) segment of the new ID is the same as that of the old one, which reduces the complexity of mapping. The only difference between the old and new IDs is the orientation/adaptation/gradient segment. If only one curve is generated at an intersection, or no additional geometric information (orientation/adaptation/gradient) is included in either of the old and new IDs, there is an exact match for IDs. If two or more curves are generated at the intersection, and additional surface information is included in both old and new IDs, the mapping is based on *closeness* of curves.

Suppose c_1 is the intersection curve of surfaces f_1 and g_1 , and c_2 is the intersection curve of surfaces f_2 and g_2 . Points \mathbf{p}_1 and \mathbf{p}_2 are on curves c_1 and c_2 respectively. The *k-closeness* of curve c_1 and c_2 at \mathbf{p}_1 and \mathbf{p}_2 , $k\text{-close}(f_1, g_1, f_2, g_2, \mathbf{p}_1, \mathbf{p}_2)$, is defined as

$$k\text{-close}(f_1, g_1, f_2, g_2, \mathbf{p}_1, \mathbf{p}_2) = |\mathbf{p}_1 - \mathbf{p}_2| + \sum_{i=1}^k \sum_{j=1}^k |\mathbf{I}_{ij}(f_1, g_1, \mathbf{p}_1) - \mathbf{I}_{ij}(f_2, g_2, \mathbf{p}_2)| \quad (k \geq 0). \quad (5.9)$$

0-closeness of curve c_1 and c_2 at \mathbf{p}_1 and \mathbf{p}_2 is the distance between \mathbf{p}_1 and \mathbf{p}_2 .

Curve mapping can be done based on the values of k -closeness. If m curves (c_1, c_2, \dots, c_m) ($m > 1$) are generated by the intersection of surfaces f_1 and g_1 in the new solid, and n curves (d_1, d_2, \dots, d_n) ($n > 1$) were generated by the corresponding surfaces f_2 and g_2 in the old solid, there is a point \mathbf{p}_i selected on each of the c_i ($i = 1, 2, \dots, m$) and a point \mathbf{q}_j selected on each of the d_j ($j = 1, 2, \dots, n$), where \mathbf{p}_i and \mathbf{q}_j are the intersecting points between the curves and a plane $x = a$ (or $y = b$, or $z = c$). For each pair of c_i and d_j , $k\text{-close}(f_1, g_1, f_2, g_2, \mathbf{p}_i, \mathbf{q}_j)$ is calculated. If only orientation is included in curve IDs, $k = 1$. If adaptation information is included in curve IDs, $k = 2$. Generally, k is the highest order of surface gradient in the curve IDs. Then an $m \times n$ closeness matrix \mathbf{R} is generated by listing each of the new curves as row indices and each of the old curves as column indices. In each row r_i of \mathbf{R} , the elements r_{ij} is the rank of closeness based on $k\text{-close}(f_1, g_1, f_2, g_2, \mathbf{p}_i, \mathbf{q}_j)$ for $j=1, 2, \dots, n$. The smallest k -closeness is ranked as 1, and the largest k -closeness is ranked as n . If a tie appears in k -closeness, $(k+1)$ -closeness ($k > 0$) of the curves is calculated for the closeness matrix.

Once the closeness matrix is built, the mapping of curves can be done by selecting the lowest rank values. Each new curve will be mapped to its corresponding old curve of rank 1. In special cases, it is possible that one new curve is mapped to multiple old curves when a curve is split into multiple curves (i.e., a old curve has the lowest rank value in multiple rows). For example, plane $z = 0$ intersects with cubic cylinder $x^3 - x - z = 0$ (as in Figure 42 (j)) and three curves are generated. If the plane is changed to $z = 0.25$, three new curves need to be mapped to old curves. The 2-closeness matrix is calculated at the intersection points with plane $y = 0$. According to the matrix value

$$\begin{bmatrix} 2.1682 & 7.9364 & 15.442 \\ 8.004 & 2.2033 & 11.455 \\ 13.775 & 11.694 & 1.5924 \end{bmatrix},$$

curves can be identified.

After a curve is identified, it is possible that multiple edges are generated by bounding the curve by the same set of boundary surfaces. Edge mapping then is needed based on *k-closeness* of the curve at boundary points to identify the corresponding edges between new and old solids. Suppose m edges ($m > 1$) are generated by the same set of boundary surfaces with the same intersection curve of surfaces f_1 and g_1 after re-evaluation. Each edge a_i was bounded by starting point \mathbf{p}_{is} and ending point \mathbf{p}_{ie} ($i = 1, 2, \dots, m$). Before re-evaluation, n edges ($n > 1$) are created by corresponding surfaces f_2 and g_2 . And each edges b_j was bounded by starting point \mathbf{q}_{js} and ending point \mathbf{q}_{je} ($j = 1, 2, \dots, n$). For each pair of edges a_i and b_j , $k\text{-close}_{edge}(a_i, b_j)$, can be calculated as

$$k\text{-close}_{edge}(a_i, b_j) = k\text{-close}(f_1, g_1, f_2, g_2, \mathbf{p}_{is}, \mathbf{q}_{js}) + k\text{-close}(f_1, g_1, f_2, g_2, \mathbf{p}_{ie}, \mathbf{q}_{je}). \quad (5.10)$$

Similar to the *closeness* matrix of curves, a *closeness* matrix of edges can be derived with each element as $k\text{-close}_{edge}(a_i, b_j)$. The mapping of edges is based on the ranks of *closeness* matrix. And the mapping of points is based on the *closeness* of curves.

In this surface-based semantic ID system, prefixing IDs with feature namespaces transforms the original flat namespace to an organized logical naming hierarchy. The IDs identify themselves descriptively by the procedure of feature operations. The inclusion of geometric information and boundary association in topological IDs let a topological ID possess geometric and topological semantics. The geometric IDs possess construct relations of surfaces, curves, and points. Because of possible ambiguity if only surface IDs are included in topological entities, necessary orientation and gradient information is included in IDs when multiple curves

are formed by intersecting the same set of surfaces, or multiple edges are created by same curve and boundary surfaces. Curve and edge mapping is needed if orientation and gradient information is involved in IDs before and after geometry re-evaluation.

To summarize this chapter, feature and parametric information is indispensable part of solid modeling in the UL-PML scheme. Features are represented in dual mode such that intentional features and geometric features are combined. The redundancy ensures that design intent is represented in the model. Constraints are captured in data model to reflect design specifications, both geometric and non-geometric. While symbolic constraints are modeled descriptively, numerical constraints are represented by interval values to improve the reliability and quality of computational models. These relations are connected by virtual link. A semantic ID method is proposed such that entities are named based on persistent geometry to solve the problem of topology inconsistency in parametric modeling and broken link in the UL model. All of the above aims to improve the interoperability of CAD data modeling in a distributed design environment.

By now, several interoperability issues concerning geometry, features, and constraints for collaborative design have been addressed. Yet there is one more problem of interoperability during different stages of design, which is model interoperability in terms of time between conceptual design and detailed design. Commonly used CAD data models are only for detailed design. No applicable data model is available for the early conceptual design stage, during which geometric information is incomplete and uncertainty exists. There is no generic data model that represents geometric and non-geometric information both for conceptual design and detailed

design. Lack of geometric models for conceptual design is one of the major hurdles for the development of Computer-Aided Conceptual Design. Incorporating robustness consideration mentioned in Section 5.2.2, an interval geometric modeling scheme is proposed to enable design data to be modeled from conceptual design to detailed design and design optimization, which improves CAD data interoperability for different time frames during design. The following chapter describes this scheme.

6.0 INTERVAL GEOMETRIC MODELING

During the process of design, various parameters are specified, which include geometric parameters (e.g., dimension, coordinate, and tolerance) and non-geometric ones (e.g., material characteristics, tooling speed, and expected life). Current CAD systems only allow geometric parameters to have fixed values, such as the position of a point in 3D space, the direction of a line, the distance between two axes. Instead of simply assigning one real value to a parameter, it will be advantageous to give an interval value to each parameter in a CAD model, which means that the parameter can take any valid value between the lower and upper bounds of the interval. Fixed parameter values generate some problems.

First, fixed-value constraints bring up conflicts easily at later design stages. Specifying determined parameter values implicitly adds rigid constraints on the geometry. The rigid constraints reduce the freedom of geometric entities to the minimal levels. These predominant constraints will be carried to other stages of design and most likely are the sources of conflicts. To resolve the conflicts, some parameter values have to be changed. This trial-and-error cycle will continue until no conflicts are found. If an interval is given to a parameter instead of a fixed value such that any real value within the interval is valid, the degrees of freedom of geometric entities are increased at the early design stages. As more constraints are imposed onto the designed object during the process of design, the freedom of geometric entities will be restricted gradually. The allowable intervals of parameter values are reduced by stages. There will be fewer chances for conflicts to occur during design, and some cycles of modification will be saved.

Second, the requirement of fixed parameter values makes the development of Computer-Aided Conceptual Design (CACD) difficult. At the conceptual design stage, actual values of parameters may not be known. Usually it is not important to specify fixed values of certain parameters yet. Current CAD systems require that parameter values be fully specified and fixed, thus are not effective tools for conceptual design. It is challenging to develop a practically usable CACD tool based on the current scheme of fixed parameter values. Nevertheless, if a parameter value is specified as a range, the problem of parameter partial integrity can be solved, i.e., it is not necessary to fix all values of parameters. This increases the flexibility of the geometric shape of the designed part.

Besides the ability of tackling problems of fixed parameter values, parameter intervals also directly represent bounding information for design optimization. Current design optimization process often occurs after parameters are specified at the detailed design stage, while the intention of feasible ranges of parameters from upstream design activities is not transferable with the fixed-value scheme. Parameter constraints of feasibility have to be added separately for optimization. However, with the interval representation, the parameter information is directly applicable for parameter optimization. Parameter intervals appropriately represent design intent of feasibility, thus integrate the sketching and optimization of design. Parameter optimization can be performed based on the inherent value bounds.

In real situations, there are some uncertainty factors in CAD modeling. Aided by computer, the dimensions and shape of the designed product are calculated and stored digitally. Computational errors from rounding are inevitable, which can become serious if the magnitudes of numbers are very different. Uncertainty also comes from the measurement and tolerance of human perception. The real value of measurement is the ideal situation that cannot be realized

from statistical points of view. Further more, the precision of numbers in a computer depends on the word size and floating-point representation of the computer. Different types of computers may have different architectures and representations. Variation exists among different computers. Thus, a computer-generated value can be looked as a sample from a range of values, while the CAD data of a designed product is a sample from the population of models. Parameter intervals capture the uncertainty characteristics, and properly model the process of design.

Intervals also can provide a uniform representation for geometric data and manufacturing tolerances in CAD models. Both variational models and tolerance zone models can be represented by interval methods. Tolerance propagation or transformation can be easily performed by interval analysis.

6.1 Preliminaries of Traditional Interval Analysis

Traditional interval analysis began as a tool for bounding rounding errors in numerical computation. Early researchers include Dwyer [154], Warmus [155, 156], Sunaga [157], Moore [158], and Hansen [159], etc.

An *interval number* is defined as an ordered pair of real numbers, $[a, b]$, with $a \leq b$. That is, it consists of the set $\{x: a \leq x \leq b\}$. Degenerated intervals $[a, a]$ are equivalent to real numbers. Interval mathematics is a generalization in which interval numbers replace real numbers, interval arithmetic replaces real arithmetic, and interval analysis replaces real analysis.

Let $[a] = [\underline{a}, \bar{a}]$, $[b] = [\underline{b}, \bar{b}]$ be real intervals and \circ be one of the basic operations *addition*, *subtraction*, *multiplication* and *division* respectively for real numbers, that is, $\circ \in \{+, -, \cdot, /\}$. The corresponding operations for interval $[a]$ and $[b]$ are defined by

$$[a] \circ [b] = \{x \circ y \mid x \in [a], y \in [b]\}. \quad (6.1)$$

Assuming $0 \notin [b]$ in case of division, the arithmetic operations are defined as:

$$[a] + [b] = [\underline{a} + \underline{b}, \bar{a} + \bar{b}], \quad (6.2)$$

$$[a] - [b] = [\underline{a} - \bar{b}, \bar{a} - \underline{b}], \quad (6.3)$$

$$[a] \cdot [b] = [\min\{\underline{a}\underline{b}, \underline{a}\bar{b}, \bar{a}\underline{b}, \bar{a}\bar{b}\}, \max\{\underline{a}\underline{b}, \underline{a}\bar{b}, \bar{a}\underline{b}, \bar{a}\bar{b}\}], \quad (6.4)$$

$$\frac{[a]}{[b]} = [a] \cdot \frac{1}{[b]}, \text{ if } \frac{1}{[b]} = \left\{ \frac{1}{y} \mid y \in [b] \right\}, \text{ and } 0 \notin [b]. \quad (6.5)$$

Detailed information about interval arithmetic and analysis can be found in references [160, 161, 162, 163, 164, 165].

6.2 Concepts of Interval Geometric Modeling (IGM)

Computer graphics and surface modeling have started using methods of traditional interval analysis. Research includes rasterizing parametric surfaces [166], ray tracing of parametric surfaces [167] and implicit surfaces [168], collision detection of polyhedral objects [169] and surfaces [170, 171, 172], error bounding and approximation in polyhedral [138] and curve-surface modeling [139, 140, 173, 174]. In the above research, interval methods are employed either as assistance and approximation tools for analysis of fixed value computation, in which interval number provides a concise format for the bounding box or range commonly used in computer graphics algorithms, or as approximation representation of geometry to embody errors and improve robustness of geometric modeling. Based on trivial-width interval values, some traditional interval arithmetic and analysis methods are used for the geometry approximation.

Different from the above, Interval Geometric Modeling (IGM) presented here allows all numerical values of parameters including coordinates, dimensions, and other values to be non-trivial-width interval numbers. With this general representation scheme, issues of rigid constraints and uncertainty will be solved in CACD. Numerical constraints can be represented in a concise way such that design optimization and approximation can become an integrated part of CAD. In addition, under-constrained and over-constrained problems in current parametric modeling can be handled more elegantly.

In IGM, we define interval number X as $X = [x_L, x_N, x_U]$ which contains lower bound value x_L , nominal value x_N , and upper bound value x_U . The nominal value is usually corresponding to the specified fixed value in current CAD systems, which should be between the lower and upper bounds.

The introduction of the nominal value in an interval is necessary for CAD modeling, since the nominal value represents the specification of the parameter if the parameter is fixed, thus intervals can be easily integrated with current fixed-value system. It allows current CAD modeling systems to adopt interval parameters such that current modeling schemes and computer visualization can be used. For example, a 2D point $P([1,2,3],[4,5,6])$ has the specified nominal position $(2,5)$. The nominal values are allowed to be changed within the intervals of x and y coordinates respectively. Within a CAD system, the point can be displayed at $(2,5)$. When P is fixed, its coordinates are $([2,2,2],[5,5,5])$, where the intervals converge to the nominal values. To simplify the notation, we can use a real number for a degenerated interval. For example, 0 represents $[0,0,0]$ as well. Figure 44 shows the valid range of a point specified by intervals in 2D and 3D spaces respectively.

2D Point:

$$p(X, Y) = p([x_L, x_N, x_U], [y_L, y_N, y_U])$$

3D Point:

$$p(X, Y, Z) = p([x_L, x_N, x_U], [y_L, y_N, y_U], [z_L, z_N, z_U])$$

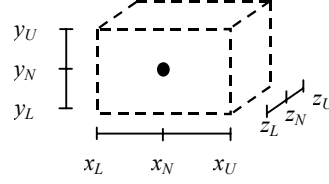
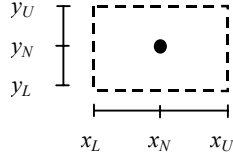


Figure 44: Range of a point specified by interval numbers

6.2.1 Interval Definitions in IGM

An interval value is a set of real numbers. An n dimensional real number space is denoted as \mathbf{R}^n . An n dimensional interval number space is denoted as \mathbf{IR}^n .

Definition 6.1: $X = [x_L, x_N, x_U] = \{x | x_L \leq x \leq x_U, x_L \leq x_N \leq x_U\}$, where $x_L \in \mathbf{R}$, $x_N \in \mathbf{R}$, $x_U \in \mathbf{R}$, and $X \in \mathbf{IR}$.

Inclusion (\subset , \subseteq , $\not\subset$) and *belong* (\in , \notin) relations of sets are valid for interval values, as well as union (\cup), intersect (\cap), and difference (\setminus). Given that $A = [a_L, a_N, a_U]$, $B = [b_L, b_N, b_U]$, we have the following relations:

Definition 6.2 (equivalence): $A = B \Leftrightarrow (a_L = b_L) \wedge (a_U = b_U)$.

The equivalence relation is reflexive, symmetric, and transitive.

Definition 6.3 (nominal equivalence): $A := B \Leftrightarrow (a_L = b_L) \wedge (a_N = b_N) \wedge (a_U = b_U)$.

Definition 6.4 (strictly greater than or equal to): $A \sim \geq B \Leftrightarrow a_L \geq b_U$.

Definition 6.5 (strictly greater than): $A \sim > B \Leftrightarrow a_L > b_U$.

Definition 6.6 (strictly less than or equal to): $A \sim \leq B \Leftrightarrow a_U \leq b_L$.

Definition 6.7 (strictly less than): $A \sim < B \Leftrightarrow a_U < b_L$.

Definition 6.8 (inclusion): $A \subseteq B \Leftrightarrow (a_U \leq b_U) \wedge (a_L \geq b_L)$,

$$A \subset B \Leftrightarrow (a_U < b_U) \wedge (a_L > b_L).$$

Figure 45 illustrates the relations of intervals. $0 = [0, 0, 0]$ is also called zero interval. Interval A is *positive*, if and only if $A \rightsquigarrow 0$. Interval A is *negative*, if and only if $A \rightsquigarrow < 0$. If the nominal value of $A = [a_L, a_N, a_U]$ is not concerned, it can simply be denoted as $[a_L, a_U]$.

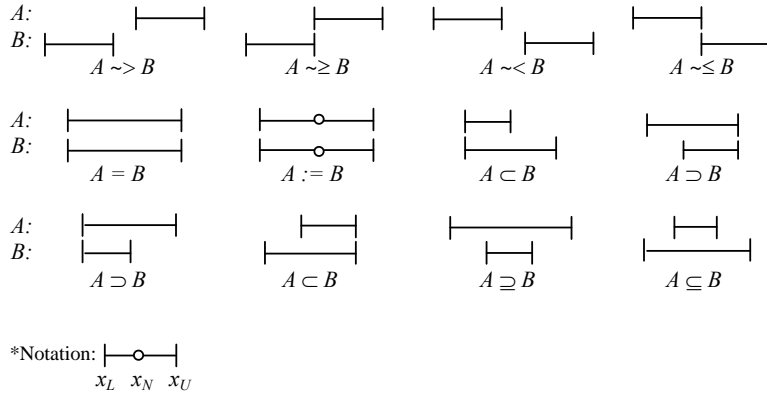


Figure 45: Relations between intervals

Definition 6.9: Interval $A = [a_L, a_N, a_U]$ is *empty*, denoted as $A = \emptyset$, if and only if $a_L > a_U$.

A is called *invalid* when $a_N > a_U$, or $a_L > a_N$, or A is empty.

Definition 6.10 (intersect): $A \cap B = \{x \mid x \in A \text{ and } x \in B, x \in \mathbf{R}\}$, if $A \cap B \neq \emptyset$, it can be derived

$$\text{by } A \cap B = [\max\{a_L, b_L\}, (\max\{a_L, b_L\} + \min\{a_U, b_U\})/2, \min\{a_U, b_U\}].$$

Definition 6.11 (union): $A \cup B = \{x \mid x \in A \text{ or } x \in B, x \in \mathbf{R}\}$, if $A \cap B \neq \emptyset$, it can be derived by

$$A \cup B = [\min\{a_L, b_L\}, (\min\{a_L, b_L\} + \max\{a_U, b_U\})/2, \max\{a_U, b_U\}].$$

Definition 6.12 (difference): $A \setminus B = \{x \mid x \in A \text{ and } x \notin B, x \in \mathbf{R}\}$.

Some basic arithmetic operations are defined.

Definition 6.13: $A + B = [a_L + b_L, a_N + b_N, a_U + b_U]$.

Definition 6.14: $A - B = [a_L - b_U, a_N - b_N, a_U - b_L]$.

Definition 6.15: $A \cdot B = [\min\{a_L b_L, a_L b_U, a_U b_L, a_U b_U\}, a_N b_N, \max\{a_L b_L, a_L b_U, a_U b_L, a_U b_U\}]$.

Definition 6.16: $\frac{1}{B} = \left\{ \frac{1}{y} \mid y \in B, 0 \notin B \right\}$.

Definition 6.17: $\frac{A}{B} = \left\{ \begin{array}{ll} A \cdot \frac{1}{B} & (0 \notin B) \\ [-\infty, 0, +\infty] & (B = 0) \\ \left[\frac{a_U}{b_L}, \frac{a_U}{b_L}, +\infty \right] & (a_U \leq 0, b_L < 0, b_U = 0) \\ \left[-\infty, \frac{a_U}{b_U}, \frac{a_U}{b_U} \right] \cup \left[\frac{a_U}{b_L}, \frac{a_U}{b_L}, +\infty \right] & (a_U \leq 0, b_L < 0, b_U > 0) \\ \left[-\infty, \frac{a_U}{b_U}, \frac{a_U}{b_U} \right] & (a_U \leq 0, b_L = 0, b_U > 0) \\ [-\infty, 0, +\infty] & (a_L < 0, a_U > 0, b_L \leq 0, b_U \geq 0) \\ \left[-\infty, \frac{a_L}{b_L}, \frac{a_L}{b_L} \right] & (a_L \geq 0, b_L < 0, b_U = 0) \\ \left[-\infty, \frac{a_L}{b_L}, \frac{a_L}{b_L} \right] \cup \left[\frac{a_L}{b_U}, \frac{a_L}{b_U}, +\infty \right] & (a_L \geq 0, b_L < 0, b_U > 0) \\ \left[\frac{a_L}{b_U}, \frac{a_L}{b_U}, +\infty \right] & (a_L \geq 0, b_L = 0, b_U > 0) \end{array} \right.$.

Note that $A - A \neq 0$. During the processes of arithmetic operations, it is possible that an *empty* interval occurs.

The width of an interval is a real number, defined as $\text{wid}(A) = a_U - a_L$. Specially, $\text{wid}(\emptyset) = 0$. Some other notations are $\text{ub}(A) = a_U$, $\text{lb}(A) = a_L$, and $\text{nom}(A) = a_N$.

6.2.2 Sampling Relation between Real Number and Interval Number

The intervals capture the uncertainty of design. The association of a real number with an interval number is considered as a *sampling* relation. The value of a parameter, which is generated by computer or selected by human designer, is a sample of the corresponding set of values within the interval. Statistically, the interval is the sampling population of real numbers.

Therefore, one CAD interval model is allowed to generate different shapes because of parameter intervals. Implicitly, a CAD interval model defines a set of geometric shapes that automatically accommodate geometry variation.

Definition 6.18: A real number x is a *sample* of interval X , if and only if $x \in X$.

Some *strict* relations exist between intervals, which are related to real number samples.

Definition 6.19: $X\mathfrak{R}Y \Leftrightarrow \forall x \in X, \forall y \in Y, x\mathfrak{R}y$. $X\mathfrak{R}Y$ denotes that X has a *strict* relation \mathfrak{R} with Y ($X \in \mathbf{IR}, Y \in \mathbf{IR}$).

That is, $X\mathfrak{R}Y$ if and only if for any sample of X , any sample of Y has a relation with it. For example, two intervals are strictly equal if and only if any two sampling real numbers from them respectively are always equal.

Definition 6.20 (*strict equivalence*): $A \sim= B \Leftrightarrow \forall x \in A, \forall y \in B, x = y$.

The definitions 6.4, 6.5, 6.6, and 6.7 implicitly define the strict unequal relations between two intervals. These four definitions are equivalent to the following definitions 6.21, 6.22, 6.23, and 6.24, respectively.

Definition 6.21 (*strictly greater than or equal to*): $A \sim\geq B \Leftrightarrow \forall x \in A, \forall y \in B, x \geq y$.

Definition 6.22 (*strictly greater than*): $A \sim> B \Leftrightarrow \forall x \in A, \forall y \in B, x > y$.

Definition 6.23 (*strictly less than or equal to*): $A \sim\leq B \Leftrightarrow \forall x \in A, \forall y \in B, x \leq y$.

Definition 6.24 (*strictly less than*): $A \sim< B \Leftrightarrow \forall x \in A, \forall y \in B, x < y$.

Besides strict relations, some *global* relations exist in interval arithmetic evaluation and problem solving.

Definition 6.25: $X\mathfrak{S}Y \Leftrightarrow \forall x \in X, \exists y \in Y, x\mathfrak{S}y$. $X\mathfrak{S}Y$ denotes that X has a *global* relation \mathfrak{S} with Y ($X \in \mathbf{IR}, Y \in \mathbf{IR}$).

That is, $X \approx Y$ if and only if for any sample of X there exists a sample of Y that has a relation with it. Global relations ensure the feasibility of interval arithmetic operations and solutions. The goal of solving interval problems is to find a region that includes all feasible solutions. The corresponding process is to eliminate certainly infeasible points from a given region so as to make it as compact as possible. The global relations make global solution and optimization of interval analysis possible. For example, the four basic arithmetic operations of intervals follow the rule of global relation and generate the global solution with a compact bound. A global equivalence can be defined as follows, which is used in systems of interval equations.

Definition 6.26 (*global equivalence*): $A = B \Leftrightarrow \forall x \in A, \exists y \in B, x = y$.

Note that global equivalence is asymmetric. The equivalence relation in definition 6.2 can be looked as a special case of symmetric global equivalence. Similarly, some inequalities can be defined as global relations that are used in systems of inequalities. For $A = [a_L, a_N, a_U]$, and $B = [b_L, b_N, b_U]$, there are

Definition 6.27 (*greater than or equal to*): $A \geq B \Leftrightarrow a_L \geq b_L$. Equivalently,

$$A \geq B \Leftrightarrow \forall x \in A, \exists y \in B, x \geq y.$$

Definition 6.28 (*greater than*): $A > B \Leftrightarrow a_L > b_L$. Equivalently, $A > B \Leftrightarrow \forall x \in A, \exists y \in B, x > y$.

Definition 6.29 (*less than or equal to*): $A \leq B \Leftrightarrow a_U \leq b_U$. Equivalently,

$$A \leq B \Leftrightarrow \forall x \in A, \exists y \in B, x \leq y.$$

Definition 6.30 (*less than*): $A < B \Leftrightarrow a_U < b_U$, where $A = [a_L, a_N, a_U]$, and $B = [b_L, b_N, b_U]$.

Equivalently, $A < B \Leftrightarrow \forall x \in A, \exists y \in B, x < y$.

Note that it is possible that $A \leq B$ and $A \geq B$ at the same time. Some properties in real analysis do not apply in interval analysis. Again, strict inequalities are special cases of global

inequalities. Function evaluation and problem solving in interval analysis are normally based on global relations.

In a multidimensional interval space, an interval vector can be defined in \mathbf{IR}^n with each component as an interval value, and an interval matrix is defined in $\mathbf{IR}^m \times \mathbf{IR}^n$ with each element as an interval value. Corresponding to a real function $f: \mathbf{R}^n \rightarrow \mathbf{R}^m$, if $f_{set}(\mathbf{X})$ denotes $\{f(\mathbf{x}) \mid \mathbf{x} = (x_1, x_2, \dots, x_n), x_i \in X_i (i = 1, \dots, n), \mathbf{X} = (X_1, X_2, \dots, X_n), \mathbf{X} \in \mathbf{IR}^n\}$, a function $F: \mathbf{IR}^n \rightarrow \mathbf{IR}^m$ is called an *inclusion function* for f at \mathbf{X} if $f_{set}(\mathbf{X}) \subseteq F(\mathbf{X})$. A *natural inclusion function* $f(\mathbf{X})$ for $f(\mathbf{x})$ is obtained by replacing each occurrence of the variable x_i by interval variable X_i . It is based on the inclusion isotonicity of the interval operations [175] and the property of pre-declared inclusions [176]. Generally, the natural inclusion function $f(\mathbf{X})$ for $f(\mathbf{x})$ is not tight enough, i.e., $f(\mathbf{X}) \subset f(\mathbf{x})$, because of dependency between variables and wrapping effect [177].

Interval vectors with same dimensions can be ranked and sorted ascendantly.

Definition 6.31: Interval vector \mathbf{A} and \mathbf{B} are ascendantly ordered, $\mathbf{A} \prec \mathbf{B}$, where $\mathbf{A} = (A_1, A_2, \dots, A_n), \mathbf{B} = (B_1, B_2, \dots, B_n) \Leftrightarrow A_n \leq B_n$, and $\neg(A_i < B_i) \rightarrow (A_{i-1} \leq B_{i-1})$ recursively apply starting from $i = n$.

Definition 6.32: Interval vector \mathbf{A} and \mathbf{B} are descendently ordered, $\mathbf{A} \succ \mathbf{B}$, where $\mathbf{A} = (A_1, A_2, \dots, A_n), \mathbf{B} = (B_1, B_2, \dots, B_n) \Leftrightarrow A_n \geq B_n$, and $\neg(A_i > B_i) \rightarrow (A_{i-1} \geq B_{i-1})$ recursively apply starting from $i = n$.

Definition 6.33: $\maxwid(\mathbf{A}) = \max_i(\text{wid}(A_i))$, where $\mathbf{A} = (A_1, A_2, \dots, A_n)$.

Definition 6.34: $\minwid(\mathbf{A}) = \min_i(\text{wid}(A_i))$, where $\mathbf{A} = (A_1, A_2, \dots, A_n)$.

6.3 Geometry Description in IGM

With the inherent capability of modeling variation, IGM has special properties that makes it different from current geometric modeling schemes.

6.3.1 Modeling Uncertainty in IGM

The characteristics of variation and uncertainty are inherent in the process of design, during which design knowledge and constraints from different aspects are applied to generate the shape and configuration of the designed product. Good CAD systems should model geometry as well as the design process, such that design can be easily modified at different design stages. Compared to traditional variational / parametric design, in which geometry is determined by parameters, IGM gives more flexibility to designers, because variation, inexactness, and uncertainty of parameters are taken into consideration.

In an IGM system, all numerical values for coordinates, dimensions, geometric constraints, and other properties are specified in the interval format. For example, a 2D model of a triangle is illustrated in Figure 46. The numerical values of geometry, including coordinates of three points, three vectors, and distances are specified with interval values. The interval format of parameters in a geometric modeling system allows variation and uncertainty to be modeled explicitly, especially at early design stages. This provides more leeway for designers to change the shape during the design. The decisions to fix values of parameters are postponed until later design stages.

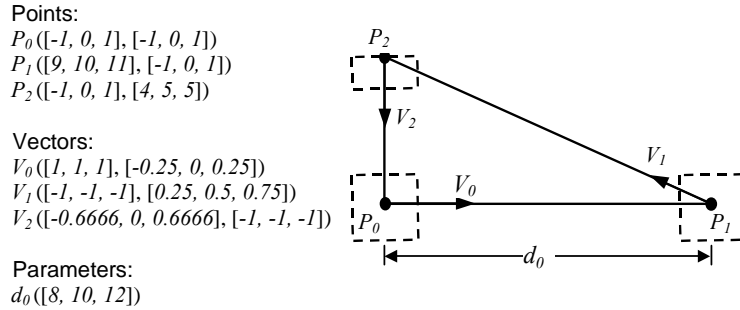


Figure 46: A 2D triangle geometry specified by intervals

While the available ranges of parameters are narrowed down gradually, uncertainty is ruled out and decisions are made throughout the design process until final design is generated. Changing current constraints or adding extra constraints would lead to different geometries. As illustrated in Figure 47, the shape of a 2D rectangular object may vary based on coordinates of four corner points within their allowable intervals. Because of the overlapping of the interval areas, the shape could be a rectangle, a triangle, or even a point. Adding or changing geometric constraints may reduce the allowable regions for these corner points, thus finalizing shapes eventually. This constraint-driven procedure reflects the nature that design is a process of constraint imposition and decision making.

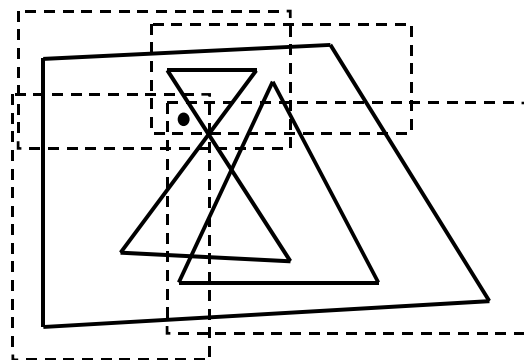


Figure 47: Constraint-driven geometry in interval modeling

6.3.2 Solving Under-constrained Problems

At early design stages, design usually deals with product concepts and system-level configuration. Values of detailed geometric parameters are not critical. Current CAD modeling scheme, which requires fixed parameter values, is not good at modeling geometry for the concept generation. At this stage, the geometric shape for each part is generated to implement functions of the new product. The general geometry and configuration are specified in terms of functionality, whereas precise values of parameters are not determined yet. In this case, geometry has the properties of incompleteness, inexactness, and approximation. It is difficult to model incomplete and inexact geometry in current fixed-value CAD systems, which require well-constrained data and information.

For example, at the initial stage of designing a mounting bracket, the geometric shape of this sheet-metal part is not decided yet, as illustrated in Figure 48a. The available constraints are the distance between corner points P_0 and P_1 , the perpendicularity between lines L_0 and L_1 , and lines L_0 and L_3 , as listed in Figure 48b. Though the 2D plate is under-constrained in traditional parametric CAD systems, the geometry still can be generated in IGM systems.

The difference of how under-constrained problems is handled in an IGM system is that each numerical value has lower bound, upper bound, and nominal value, and the interval defines the feasible region of the value implicitly. This type of *soft* constraints are applied to geometry inherently at every step of value specifications. The effect of adding more constraints is to reduce the allowable region of geometric entities systematically such that the final geometry can be fixed. In modeling under-constrained geometries, the shape of entities is constrained by the allowable value ranges, such as coordinate intervals and distance intervals. In the example of Figure 48, points P_2 and P_3 are constrained within their coordinate intervals implicitly. Even

though no other distance or angular constraints are added onto them, the geometry still can be modeled with certain flexibility. Therefore, the concept of under-constrained geometry in traditional parametric or variational design is not critical in IGM.

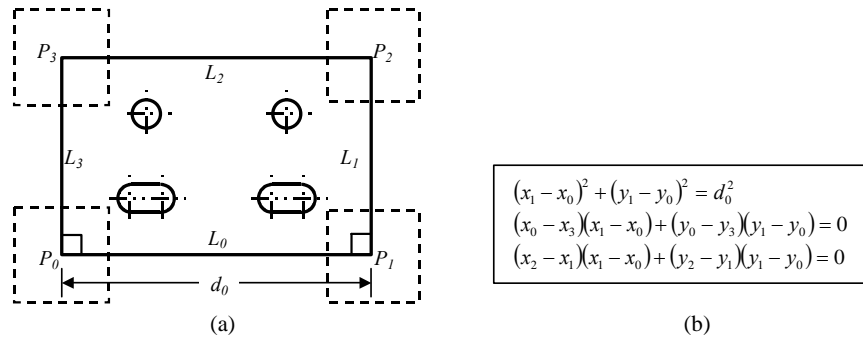


Figure 48: An example of under-constrained geometry in bracket design

6.3.3 Solving Over-constrained Problems

As design migrates from conceptual design to detailed design stages, more information is available for decision making. In most cases, the information is more than enough to determine the geometry, by which multidisciplinary specifications from different aspects are to be met. There is a high possibility that conflicts of requirements occur, thus tradeoffs of constraints should be made to resolve conflicts.

In current parametric CAD systems, only well-constrained geometry can be solved, thus proper geometric constraints should be assigned to determine geometry. Either under-constrained or over-constrained situation is not allowed. For instance, in the previous bracket design, if geometric constraints are specified as: the position of P_0 ; distances between P_0 and P_1 , P_1 and P_2 , P_2 and P_3 , and P_3 and P_0 ; L_0 is perpendicular to L_1 as well as to L_3 ; and L_0 is horizontal. Current CAD systems will complain that this geometry is over-constrained, as illustrated in Figure 49.

Traditional parametric modeling scheme has strict requirements on the number of constraints and the way constraints are applied.

In IGM, the parameter values are all interval values, which means that all distance and angle values in the previous example are interval values. Thus these interval value constraints are not as rigid as fixed-value ones. Adding more constraints reduces the feasible regions of geometric entities. Only those constraints which cause no feasible regions generate conflicts. This approach thus loosens the current requirements on applying constraints. Some of the previously over-constrained problems will no longer be over-constrained in IGM.

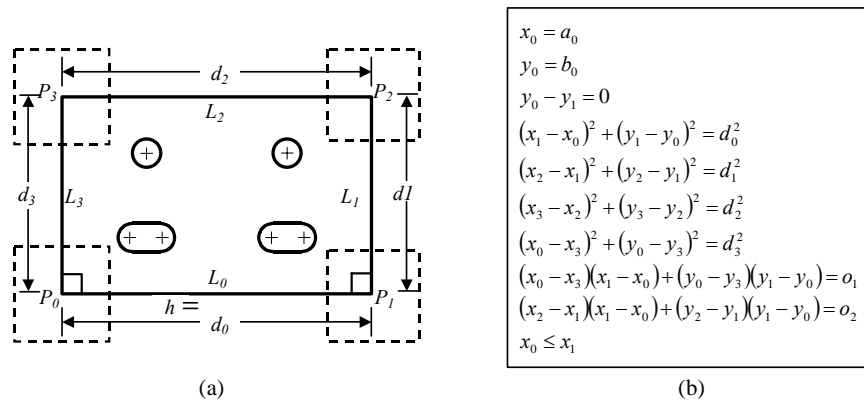


Figure 49: An example of over-constrained geometry in bracket design

6.4 Solving Equations in Interval Geometric Modeling

To incorporate interval geometric modeling methodology into current CAD systems, several fundamental issues related to geometric computation should be addressed. These include linear and nonlinear equation representations and solutions, which are essential for

transformation operation, surface intersection, and constraint solving, etc. The process of solving systems of equations or inequalities is also called *contraction*. It starts with initial values of intervals, which are rough estimates of variable values. Then subintervals which do not contain the solutions are eliminated, and intervals are “contracted”. This process normally proceeds iteratively until no further improvement. Since interval operations involve more steps and procedures than regular arithmetic operations, time and space efficient algorithms are critical to allow extensive interval computation to be accomplished with the available computational resources.

6.4.1 Interval Linear Equations

Commonly used numerical methods for solving real-value linear equations can be extended to solve interval-value linear equations, such as Gaussian elimination and triangular factorization. But matrix-based methods do not solve under-constrained or over-constrained questions. In contrast, iteration-based methods have no well-constrain requirement, such as Jacobi iteration and Gauss-Seidel iteration. An algorithm for solving interval linear equations presented here is extended from the Gauss-Seidel method, shown in Figure 50. Different from methods of Alefeld and Herzberger [161], and Hansen and Sengupta [163], this algorithm allows under-constrained and over-constrained linear systems to be solved.

To solve

$$\sum_{j=1}^n A_{ij} X_j = Y_i \quad i = 1, 2, \dots, m \quad (6.6)$$

where X_1, X_2, \dots, X_n are interval variables, A_{ij} is interval constant for each i and j , and Y_1, Y_2, \dots, Y_m are interval constants. Here, m is not necessary equal to n , which means the linear systems

could be over-constrained ($m > n$) or under-constrained ($m < n$). If an empty interval is derived during the process, there is no solution within the given initial intervals.

```

INPUT:  Interval matrix A
        Interval vector Y
OUTPUT: Interval vector X

Interval V
int i, j, k
REPEAT until stop criterion is met
  FOR each 1 <= i <= m
    FOR each 1 <= j <= n
      IF Aij=0
        continue next j iteration
      ENDIF
      V = 0
      FOR each 1<=k<j
        V = V+Aik*Xk
      ENDFOR
      FOR each j+1<=k<=n
        V = V+Aik*Xk
      ENDFOR
      V = (Yi - V)/Aij
      Xj = Xj ∩ V
    ENDFOR
  ENDFOR

```

Figure 50: Algorithm of extended Gauss-Seidel method for solving linear equations (6.6)

6.4.2 Interval Nonlinear Equations

Nonlinear equation systems can be solved by the fix-point method, forward-backward propagation, Newton's method, and Krawczyk method, etc. Given that IGM requires a constraint solving system be flexible for the number of constraints yet with fast convergence, a linear enclosure method is presented here. Let us considering the interval nonlinear equation system

$$F_i(\mathbf{X}) = C_i \quad i = 1, 2, \dots, l, \quad (6.7)$$

where \mathbf{X} is the interval variable vector $[X_1, X_2, \dots, X_n]^T$ and C_i is a constant interval. The following steps are needed to solve the system:

STEP 1: Transform each equation of (6.7) to the separable form to eliminate dependency among variables;

STEP 2: Find the linear enclosure of each of the univariate nonlinear functions and form a linear equation system;

STEP 3: Solve the linear system by the algorithm of Section 6.4.1;

STEP 4: If the stopping criterion is met, stop. Otherwise, repeat from STEP 2 to STEP4.

STEP 1:

Function $f(x_1, x_2, \dots, x_n)$ is said to be *separable* if and only if $f(x_1, x_2, \dots, x_n) = f_1(x_1) + f_2(x_2) + \dots + f_n(x_n)$. According to Yamamura's algorithm [178], functions that are composed of four basic arithmetic operations (+, -, ×, /), unary operations (sin, exp, log, sqrt, etc.), and the power operation (^) can be transformed into the separable form by introducing necessary functions. For example, $f = f_1 \times f_2$ can be transformed to $f = (y^2 - f_1^2 - f_2^2)/2$ and $y = f_1 + f_2$; $f = f_1 / f_2$ can be transformed to $f = (y^2 - f_1^2 - 1/f_2^2)/2$ and $y = f_1 + 1/f_2$; and $f = (f_1)^2$ can be transformed to $f = \exp(y_1)$, $y_1 = (y_2^2 - (\log(f_1))^2 - f_2^2)/2$, and $y_2 = \log(f_1) + f_2$. In geometric modeling, most of the constraints/functions can be transformed into the separable form.

Thus equations (6.7) can be transformed into

$$\sum_{j=0}^n f_{ij}(X_j) = D_i \quad i = 1, 2, \dots, m, \tag{6.8}$$

where X_1, X_2, \dots, X_n are interval variables and D_1, D_2, \dots, D_m are interval constants.

STEP 2:

The algorithm based on linear interval enclosure here is more general than Kolev's method [179, 180]. Kolev's method only considers the degenerated case when $D_i = 0$ for all i . The evaluation based on linear enclosure has sharper bounds than the one based on the interval Newton's method if the widths of intervals are nontrivial or thick. Methods using coefficient matrix inverse operation, such as Hansen and Greenberg's [181], are not applicable here since situations of under-constrained and over-constrained are considered.

Linear enclosure of $f_{ij}(x_j)$ is found within the initial interval of $X_j^{(0)}$ for each i and j as follows. Let $X_j^{(0)} = [x_L^j, x_N^j, x_U^j]$, we can have

$$f_{ij}^S = f_{ij}(x_L^j), \text{ and} \quad (6.9)$$

$$f_{ij}^T = f_{ij}(x_U^j). \quad (6.10)$$

Let

$$a_{ij} = \frac{f_{ij}^T - f_{ij}^S}{x_U^j - x_L^j}. \quad (6.11)$$

The *linear enclosure* of $f_{ij}(x_j)$ can be defined as

$$E_{ij}(x) = B_{ij} + a_{ij}x \quad \text{for } x \in X_j^{(0)}, \quad (6.12)$$

such that

$$f_{ij}(x) \in E_{ij}(x) \quad \text{for } \forall x \in X_j^{(0)}, \quad (6.13)$$

as illustrated in Figure 51.

To find out a B_{ij} with the minimum width with given a_{ij} , derivation of $f_{ij}(x)$ is used if $f_{ij}(x)$ is continuous and differentiable within interval $X_j^{(0)}$. The question then is reduced to solving real value nonlinear equation

$$f_{ij}'(x) = a_{ij} \quad \text{for } x \in X_j^{(0)}, \quad (6.14)$$

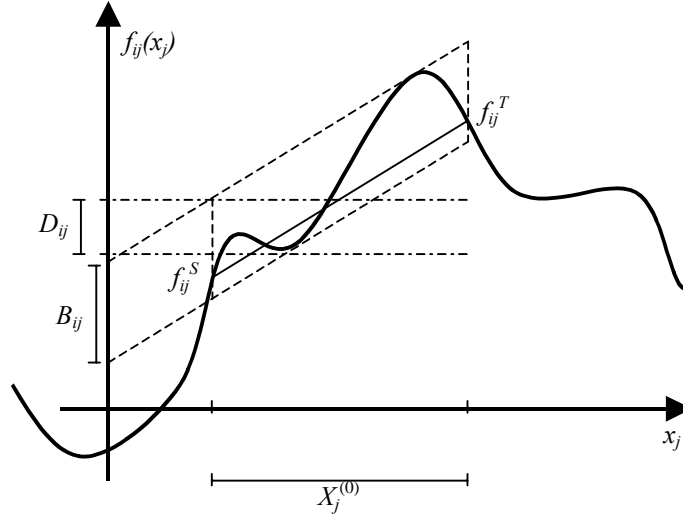


Figure 51: Linear enclosure of nonlinear interval function

Given that $f_{ij}(x)$ is continuous and differentiable for most of geometric relations, equation (6.14) has at least one solution. The Secant method can be used to solve the equation efficiently. Having been transformed to the separable form, $f'_{ij}(x)$ is a univariate polynomial function or a function with unary operations (*sin*, *cos*, etc.) for most geometric constraints. For polynomial functions, roots can be isolated within disjointed intervals individually based on Descartes' rule of signs before equations are solved numerically. Descartes' bound gives the upper bound of the number of positive roots of a polynomial. Once polynomial functions are solved, solutions to unary functions such as *sin* and *cos* can be easily found.

Let $P(x)$ be a polynomial with real coefficients, the following transformations are defined.

Definition 6.35 (*Reverse transformation*): $R[P(x)] = x^n P(1/x)$ where n is the degree of P .

Definition 6.36 (*Translation transformation*): $T_t[P(x)] = P(x + t)$ for $t \in \mathbf{R}$.

Definition 6.37 (*Homothetic transformation*): $H_c[P(x)] = P(cx)$ for $c \in \mathbf{R}$.

Based on the algorithm of Collins *et al.* [182, 183], $P_{ij}(x)$ for $x \in X_j^{(0)}$ is transformed to $P_{ij}^0(x)$ for $x \in [0, 1]$ by $P_{ij}^0(x) = H_{b-a}[T_a[P_{ij}(x)]]$ where a is the lower bound of $X_j^{(0)}$ while b is the upper bound

of X_j^0 . The roots of $P_{ij}(x)$ for $x \in X_j^0$ have one-to-one correspondence with the roots of $P_{ij}^0(x)$ for $x \in [0, 1]$. A list of root intervals or exact roots can be obtained by calling $RootIsolation(P_{ij}^0, 0, 0)$, wherein the algorithm listed in Figure 52. For each root interval or exact root with information of $(depth, index)$ in the list, there is an corresponding $x \in [\frac{(b-a)index}{2^{depth}} + a, \frac{(b-a)(index+1)}{2^{depth}} + a]$ for root intervals or $x = \frac{(b-a)index}{2^{depth}} + a$ for exact roots such that $P_{ij}(x)=0$.

```

INPUT: Polynomial P with n degree
       int depth
       int index
OUTPUT: RootIntervalList

IF P(0) = 0
    RootIntervalList.addExactRoot(depth, index)
ENDIF
IF P(1) = 0
    RootIntervalList.addExactRoot(depth, index+1)
ENDIF
Polynomial Q = T1[R(P)]
IF DecartesBound(Q) = 1
    RootIntervalList.addRootInterval(depth, index)
ELSEIF DecartesBound(Q) >= 2
    Polynomial P1 = 2nH1/2[P]
    RootIsolation(P1, depth+1, 2*index)
    Polynomial P2 = T1[P1]
    RootIsolation(P2, depth+1, 2*index+1)
ENDIF

```

Figure 52: RootIsolation procedure based on Descartes' rule of signs

Thus, interval $X_j^{(0)}$ can be subdivided into small intervals containing an individual root. Let $g(x) = f'_{ij}(x) - a_{ij}$. Solutions to (6.14) within interval $X_j^{(0)}$ can be found by (6.15), which lists the computation for each iteration n.

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{g(x_n) - g(x_{n-1})} g(x_n) \quad n = 1, 2, 3, \dots \quad (6.15)$$

Suppose x_{jp} ($p=1, 2, \dots, P$) is the p^{th} solution of equation (6.14), and $x_{j0} = x_L^j$. Let $B_{ij} = [b_L^{ij}, b_N^{ij}, b_U^{ij}]$, where

$$b_U^{ij} = \max_p \{f_{ij}(x_{jp}) - a_{ij}x_{jp}, p = 0,1,2,\dots,P\}, \quad (6.16a)$$

$$b_N^{ij} = f_{ij}(x_{j0}) - a_{ij}x_{j0}, \quad (6.16b)$$

$$b_L^{ij} = \min_p \{f_{ij}(x_{jp}) - a_{ij}x_{jp}, p = 0,1,2,\dots,P\}. \quad (6.16c)$$

From (6.13), we have

$$f_{ij}(X_j) \subseteq E_{ij}(X_j) \quad \text{for } i = 1,2,\dots,m, \quad (6.17)$$

thus,

$$\sum_{j=1}^n f_{ij}(X_j) \subseteq \sum_{j=1}^n E_{ij}(X_j) = \sum_{j=1}^n (B_{ij} + a_{ij}X_j) \quad \text{for } i = 1,2,\dots,m. \quad (6.18)$$

STEP 3:

Solving (6.8) thus is reduced to solving linear equations (6.19) iteratively.

$$\sum_{j=1}^n (B_{ij} + a_{ij}X_j) = D_i \quad \text{for } i = 1,2,\dots,m. \quad (6.19)$$

This linear system can be solved using the algorithm described in Section 6.4.1. Because the coefficient a_{ij} 's are degenerated intervals, only one iteration is needed to solve the linear equations. Suppose Y_j is the j^{th} variable solution of (6.19) in the k^{th} iteration. By (6.20), the initial value of X_j in the $(k+1)^{\text{th}}$ iteration is calculated. If an empty interval is derived, the original system has no solution within the given initial intervals $(X_1^{(0)}, X_2^{(0)}, \dots, X_n^{(0)})$.

$$X_j^{(k+1)} = X_j^{(k)} \cap Y_j \quad \text{for } j = 1,2,\dots,n. \quad (6.20)$$

STEP 4:

When the stopping criterion, such as the width of intervals has no further improvement (6.21a) or the intervals are sharp enough (6.21b), is met, the iteration is stopped. Otherwise, go back to (6.8) to find out the new linear enclosures within the updated intervals and repeat the procedure starting from STEP 2.

$$\left| \sum_{j=1}^n \text{wid}(X_j^{(k+1)}) - \sum_{j=1}^n \text{wid}(X_j^{(k)}) \right| < \varepsilon_1 \quad \text{for iteration } k. \quad (6.21a)$$

$$\left| \sum_{j=1}^n \text{wid}(X_j^{(k)}) \right| < \varepsilon_2 \quad \text{for iteration } k. \quad (6.21b)$$

6.4.3 Interval Inequalities

Consider a set of linear or nonlinear inequalities

$$F_i(\mathbf{X}) \leq C_i \quad i = 1, 2, \dots, l, \quad (6.22)$$

where \mathbf{X} is the interval variable vector $[X_1, X_2, \dots, X_n]^T$ and C_i is a constant interval, inequalities are transformed into equations

$$F_i(\mathbf{X}) + S_i = C_i \quad i = 1, 2, \dots, l, \quad (6.23)$$

where S_i is a slack variable with initial value of $[0, 0, +\infty]$. Similarly,

$$F_i(\mathbf{X}) \geq C_i \quad i = 1, 2, \dots, l, \quad (6.24)$$

where \mathbf{X} is the interval variable vector $[X_1, X_2, \dots, X_n]^T$ and C_i is a constant interval, can be transformed into

$$F_i(\mathbf{X}) + S_i = C_i \quad i = 1, 2, \dots, l, \quad (6.25)$$

where S_i is a slack variable with initial value of $[-\infty, 0, 0]$. Systems of inequalities can be changed to systems of equalities, thus can be solved by methods in Sections 6.4.1 and 6.4.2.

6.4.4 A Numerical Example

As an illustration of solving nonlinear equations using the algorithms described in the above sections, constraints the bracket design in Figure 49 are used. The designer specifies the nominal value, lower bound, and upper bound of each coordinate and parameter. For example, the coordinates of P_0 are $([0, 0.25, 0.5], [0, 0.25, 0.5])$. The coordinates of P_I are $([0.5, 0.75, 1], [0, 0.25, 0.5])$. The distance between P_0 and P_I is $[0.49, 0.50, 0.51]$. Geometric constraints are assigned to generate the outline of the bracket, which is over-constrained in the sense of the traditional parametric modeling. The interval geometric modeler then calculates the ranges of geometric points based on the algorithms of solving interval linear and nonlinear equations.

Figure 53 lists the constraint equations in Figure 49 (b) which are transformed to separable form. Based on the algorithm in Section 6.4.2, this over-constrained nonlinear equation system is solved. The numerical results are listed in Table 6.

$x_0 = a_0$	$y_2 - y_3 + v_3 = 0$
$y_0 = b_0$	$u_4^2 + v_4^2 = d_3^2$
$y_0 - y_1 = 0$	$-x_0 + x_3 + u_4 = 0$
$u_1^2 + v_1^2 = d_0^2$	$-y_0 + y_3 + v_4 = 0$
$x_0 - x_1 + u_1 = 0$	$u_1^2/2 + v_1^2/2 + u_4^2/2 + v_4^2/2 - w_1^2/2 - w_2^2/2 = o_1$
$y_0 - y_1 + v_1 = 0$	$-u_1 - u_4 + w_1 = 0$
$u_2^2 + v_2^2 = d_1^2$	$-v_1 - v_4 + w_2 = 0$
$x_1 - x_2 + u_2 = 0$	$u_1^2/2 + v_1^2/2 + u_2^2/2 + v_2^2/2 - w_3^2/2 - w_4^2/2 = o_2$
$y_1 - y_2 + v_2 = 0$	$-u_1 - u_2 + w_3 = 0$
$u_3^2 + v_3^2 = d_2^2$	$-v_1 - v_2 + w_4 = 0$
$x_2 - x_3 + u_3 = 0$	$-x_0 + x_1 = c$

Figure 53: Constraint equations of Figure 49 (b) in separable form

Table 6: Numerical results of the bracket example

Variables	Initial values	Final values (after 20 iterations)	Descriptions
	$X_0 = [0, 0.25, 0.5]$	$X_0 = [0, 0, 0]$	x coordinate of P_0
	$Y_0 = [0, 0.25, 0.5]$	$Y_0 = [0, 0, 0]$	y coordinate of P_0
	$X_1 = [0.5, 0.75, 1]$	$X_1 = [0.5, 0.505012, 0.510024]$	x coordinate of P_1
	$Y_1 = [0, 0.25, 0.5]$	$Y_1 = [0, 0, 0]$	y coordinate of P_1
	$X_2 = [0.5, 0.75, 1]$	$X_2 = [0.5, 0.516686, 0.533372]$	x coordinate of P_2
	$Y_2 = [0, 0.25, 0.5]$	$Y_2 = [0.23886, 0.249714, 0.260569]$	y coordinate of P_2
	$X_3 = [0, 0.25, 0.5]$	$X_3 = [0, 0.0116355, 0.0232709]$	x coordinate of P_3
	$Y_3 = [0, 0.25, 0.5]$	$Y_3 = [0.238869, 0.249677, 0.260485]$	y coordinate of P_3
Parameters			
	$A_0 = [0, 0, 0]$		fixed position of P_0
	$B_0 = [0, 0, 0]$		fixed position of P_0
	$D_0 = [0.49, 0.50, 0.51]$		distance d_0
	$D_1 = [0.24, 0.25, 0.26]$		distance d_1
	$D_2 = [0.49, 0.50, 0.51]$		distance d_2
	$D_3 = [0.24, 0.25, 0.26]$		distance d_3
	$O_1 = [-0.001, 0, 0.001]$		perpendicularity
	$O_2 = [-0.001, 0, 0.001]$		perpendicularity

Figure 54 shows the convergence of interval calculation in solving nonlinear equations is reasonably fast. After about 15 iterations, the widths of intervals are not changed any more.

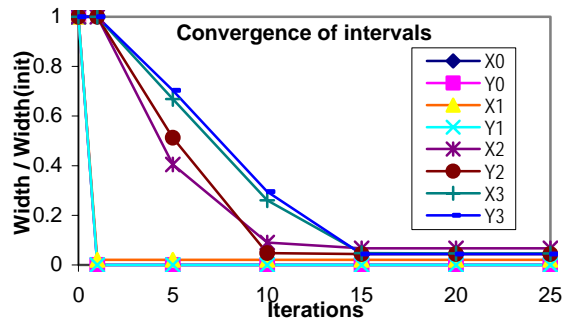


Figure 54: Convergence of Interval calculation in the bracket example

Figure 55 illustrates the variation allowance of the bracket profile. As more constraints are added, the feasible range for each corner should be narrowed down further until the position is fixed.

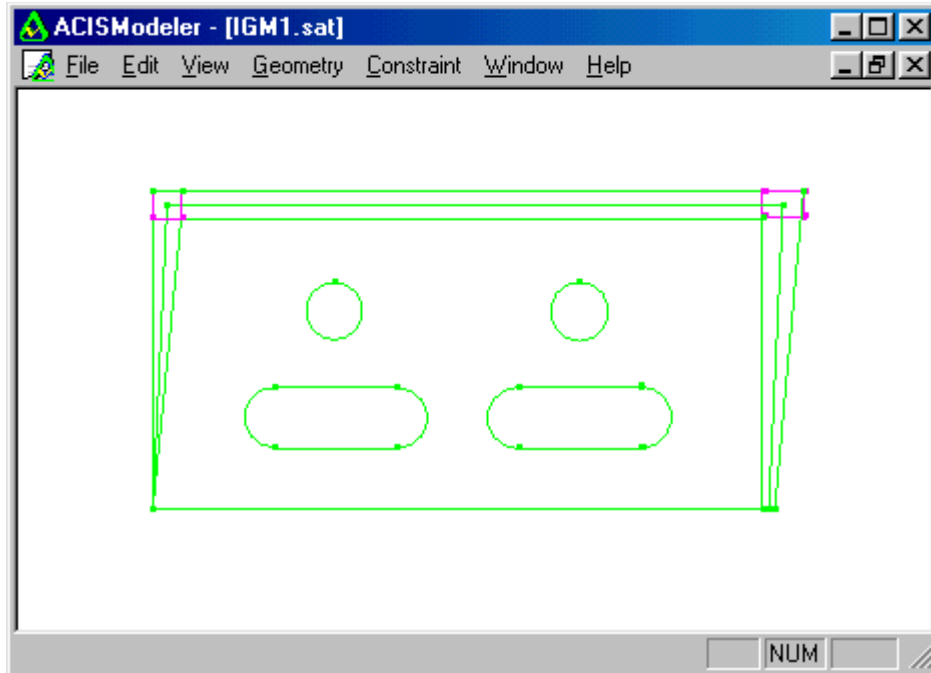


Figure 55: Variation allowance of the bracket

6.5 Design Refinement

One important aspect related to interval representation of variance allowance is the over estimation of allowances. An interval vector simply encloses the allowable region by a hyper cube, which often includes infeasible region. During the function evaluation, inclusion functions are likely to give a set that is larger than the actual solution set due to dependency or wrapping effect. Thus, interval computation tends to over estimate parameter ranges. Design refinement is needed to generate more delicate design if desirable details are not reached yet. There are two ways to refine design: *interval subdivision* and *constraint re-specification*. Interval subdivision is to divide existing interval regions into unions of subintervals to achieve the refined view of

current design. Constraint re-specification is to modify some of constraints or to add extra valid constraints to contract feasible regions.

6.5.1 Interval Subdivision

Interval subdivision (also called *subpaving*) substitutes an interval vector with multiple interval vectors such that the corresponding real space region is subdivided into multiple smaller regions to cover the actual solution set more compactly. For example, in equations

$$\begin{cases} [0,1.5,1]x_1 + [2,2.5,3]x_2 = [0,3,6] \\ [4,5,6]x_1 + [1,1.5,2]x_2 = [-6,0,8] \end{cases}$$

the solution set can be derived in four quadrants of x_1 - x_2 space respectively. Considering lower and upper bounds only, we have the actual solution set that is illustrated by the region in Figure 56.

It is clear that even the best solvers that derive the most compact solution $\mathbf{X} = ([-4, 8/3], [-4/3, 5])$ will not represent the actual solution set in terms of interval vectors. Thus, in order to approximate actual solution set well, interval vectors can be subdivided further to represent the solution. As shown in Figure 57, the interval vector $\mathbf{X} = ([-4, 8/3], [-4/3, 5])$ can be bisected recursively and tested if the subintervals belong to the actual solution set. The actual solution set thus is approximated by the union of subinterval regions.

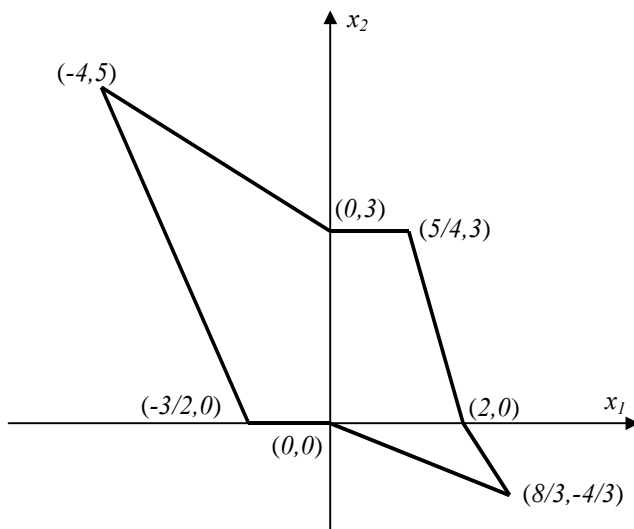


Figure 56: The solution set represented as a 2D region

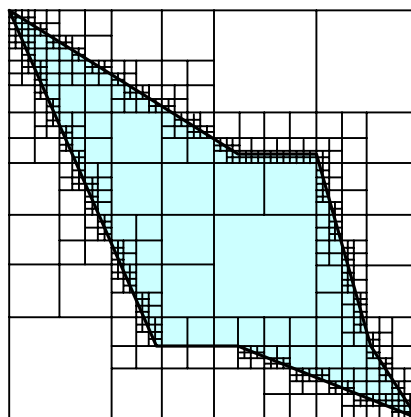


Figure 57: Two-dimensional interval vector subdivision

To represent subdivision of intervals concisely, a *power interval* can be used.

Definition 6.38: An n -dimensional *power interval* with degrees of m , denoted as $\mathbf{P}^{(m, n)}$, is an ordered list of m non-overlapped interval vectors of n -dimensional, i.e., $\mathbf{P}^{(m, n)} = [\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_m]$, where $\mathbf{X}_i \in \mathbf{IR}^n$ ($i = 1, \dots, m$), $\text{minwid}(\mathbf{X}_i \cap \mathbf{X}_j) = 0$ ($i \neq j$), and $\mathbf{X}_i \angle \mathbf{X}_{i+1}$ ($i = 1, \dots, m-1$).

Consider a design problem $f(\mathbf{X}) = \mathbf{Y}$. The target is to find out the actual solution set $\mathbf{S} \subseteq \mathbf{X}$ with minimal size such that $f(\mathbf{S}) = \mathbf{Y}$. Interval arithmetic only gives a valid solution \mathbf{D} with $f(\mathbf{D}) \supseteq \mathbf{Y}$. If the valid solution is represented by power intervals, refinement can be looked as degree elevation of power intervals. If the original solution to a problem is found as an n -dimensional vector $\mathbf{X} = [X_1, X_2, \dots, X_n]$, the corresponding power interval is $\mathbf{P}_{(0)}^{(1,n)} = [\mathbf{X}]$. One elevation operation will bisect \mathbf{X} , with each interval vector being deleted and inserted new subintervals. Feasibility of each new subinterval then can be tested. The algorithm of subdivision is shown in Figure 58.

```

INPUT:  Power Interval  $\mathbf{P}^{(m,n)}$ 
        Interval vector  $\mathbf{Y}$ 
        Mapping function  $\mathbf{f}$ 
OUTPUT: Power Interval  $\mathbf{P}^{(k,n)}$ 

IF stop criterion is met
    Return  $\mathbf{P}^{(m,n)}$ 
ELSE
     $j = m*n$ 
     $\mathbf{Q}^{(j,n)} = \text{Bisect}(\mathbf{P}^{(m,n)})$ 
    FOR  $1 \leq i \leq m*n$ 
        IF  $\mathbf{f}(\mathbf{Q}^{(j,n)}(i)) \not\subseteq \mathbf{Y}$ 
            Delete( $\mathbf{Q}^{(j,n)}(i)$ )
        ENDIF
    ENDFOR
    Subdivide( $\mathbf{Q}^{(j,n)}, \mathbf{Y}, \mathbf{f}$ )
ENDIF

```

Figure 58: Subdivide procedure for power interval elevation

Power intervals can be implemented as linked lists. Deleting and adding interval vectors during elevation operation can be completed easily. In the numerical example of Section 6.4.4, the result is represented by a power interval with a degree of 8. The degree elevation operation is done by subdividing elements 5 and 6 recursively. A refined design can be derived as shown from Table 7 to Table 10, and compared in Figure 59.

Table 7: Initial result of Section 6.4.4

Set	P ₀	P ₁	P ₂	P ₃
<i>init</i>	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.5, 0.516685, 0.533371] [0.23886, 0.249714, 0.260569]	[0, 0.0116354, 0.0232708] [0.238869, 0.249677, 0.260485]

Table 8: Subdivision level 1

Subset	P ₀	P ₁	P ₂	P ₃
<i>I</i>	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.5, 0.508343, 0.516685] [0.23886, 0.244287, 0.249714]	[0, 0.0116354, 0.0232708] [0.238869, 0.249677, 0.260485]
<i>2</i>	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.516685, 0.525028, 0.533371] [0.23886, 0.244287, 0.249714]	[0.00658538, 0.0149281, 0.0232708] [0.238869, 0.249677, 0.260485]
<i>3</i>	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.516685, 0.525028, 0.533371] [0.249714, 0.255142, 0.260569]	[0.00658538, 0.0149281, 0.0232708] [0.238869, 0.249677, 0.260485]
<i>4</i>	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.5, 0.508343, 0.516685] [0.249714, 0.255142, 0.260569]	[0, 0.0116354, 0.0232708] [0.238869, 0.249677, 0.260485]

Table 9: Subdivision level 2

Subset	P ₀	P ₁	P ₂	P ₃
<i>11</i>	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.5, 0.504171, 0.508343] [0.239791, 0.242039, 0.244287]	[0, 0.009388, 0.018776] [0.239264, 0.249824, 0.260384]
<i>12</i>	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.508343, 0.512514, 0.516685] [0.239563, 0.241925, 0.244287]	[0, 0.0116354, 0.0232708] [0.238869, 0.249677, 0.260485]
<i>13</i>	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.508343, 0.512514, 0.516685] [0.244287, 0.247001, 0.249714]	[0, 0.0116354, 0.0232708] [0.238869, 0.249677, 0.260485]
<i>14</i>	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.5, 0.504171, 0.508343] [0.244287, 0.247001, 0.249714]	[0, 0.00941007, 0.0188201] [0.238869, 0.249677, 0.260485]
<i>21</i>				
<i>22</i>				
<i>23</i>				
<i>24</i>	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.516685, 0.520857, 0.525028] [0.244287, 0.247001, 0.249714]	[0.00658538, 0.0148512, 0.023117] [0.238883, 0.249573, 0.260263]
<i>31</i>	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.516685, 0.520857, 0.525028] [0.249714, 0.252428, 0.255142]	[0.00658538, 0.0148512, 0.023117] [0.238883, 0.249573, 0.260263]
<i>32</i>				
<i>33</i>				
<i>34</i>				
<i>41</i>	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.5, 0.504171, 0.508343] [0.249714, 0.252428, 0.255142]	[0, 0.0094117, 0.0188234] [0.238869, 0.249677, 0.260485]
<i>42</i>	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.508343, 0.512514, 0.516685] [0.249714, 0.252428, 0.255142]	[0, 0.0116354, 0.0232708] [0.238869, 0.249677, 0.260485]
<i>43</i>	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.508343, 0.512514, 0.516685] [0.255142, 0.257639, 0.260137]	[0, 0.0116354, 0.0232708] [0.238869, 0.249677, 0.260485]
<i>44</i>	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.5, 0.504171, 0.508343] [0.255142, 0.25759, 0.260039]	[0, 0.00940018, 0.0188004] [0.239261, 0.249823, 0.260385]

Table 10: Subdivision level 3

Subset	P ₀	P ₁	P ₂	P ₃
111	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.5, 0.502086, 0.504171] [0.239791, 0.240915, 0.242039]	[0, 0.00730215, 0.0146043] [0.239264, 0.249824, 0.260384]
112	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.504171, 0.506257, 0.508343] [0.239791, 0.240915, 0.242039]	[0, 0.00938782, 0.0187756] [0.239264, 0.249824, 0.260384]
113	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.504171, 0.506257, 0.508343] [0.242039, 0.243163, 0.244287]	[0, 0.00938782, 0.0187756] [0.239264, 0.249824, 0.260384]
114	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.5, 0.502086, 0.504171] [0.242039, 0.243163, 0.244287]	[0, 0.00730215, 0.0146043] [0.239264, 0.249824, 0.260384]
121	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.508343, 0.510428, 0.512514] [0.239563, 0.240744, 0.241925]	[0, 0.0114805, 0.0229609] [0.238869, 0.249677, 0.260485]
122	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.512514, 0.5146, 0.516685] [0.239563, 0.240744, 0.241925]	[0.00241403, 0.0128424, 0.0232708] [0.238869, 0.249677, 0.260485]
123	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.512514, 0.5146, 0.516685] [0.241925, 0.243106, 0.244287]	[0.00241403, 0.0128424, 0.0232708] [0.238869, 0.249677, 0.260485]
124	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.508343, 0.510428, 0.512514] [0.241925, 0.243106, 0.244287]	[0, 0.0114805, 0.0229609] [0.238869, 0.249677, 0.260485]
131	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.508343, 0.510428, 0.512514] [0.244287, 0.245644, 0.247001]	[0, 0.011393, 0.0227861] [0.238869, 0.249677, 0.260485]
132	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.512514, 0.5146, 0.516685] [0.244287, 0.245644, 0.247001]	[0.00241403, 0.0128424, 0.0232708] [0.238869, 0.249677, 0.260485]
133	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.512514, 0.5146, 0.516685] [0.247001, 0.248358, 0.249714]	[0.00241403, 0.0128424, 0.0232708] [0.238869, 0.249677, 0.260485]
134	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.508343, 0.510428, 0.512514] [0.247001, 0.248358, 0.249714]	[0, 0.011393, 0.0227861] [0.238869, 0.249677, 0.260485]
141	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.5, 0.502086, 0.504171] [0.244287, 0.245644, 0.247001]	[0, 0.00722152, 0.014443] [0.239244, 0.249818, 0.260391]
142	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.504171, 0.506257, 0.508343] [0.244287, 0.245644, 0.247001]	[0, 0.00930719, 0.0186144] [0.239244, 0.249818, 0.260391]
143	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.504171, 0.506257, 0.508343] [0.247001, 0.248358, 0.249714]	[0, 0.00930719, 0.0186144] [0.239244, 0.249818, 0.260391]
144	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.5, 0.502086, 0.504171] [0.247001, 0.248358, 0.249714]	[0, 0.00722152, 0.014443] [0.239244, 0.249818, 0.260391]
211				
212				
213				
214				
221				
222				
223				
224				
231				
232				
233				
234				
241	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.516685, 0.518771, 0.520857] [0.244287, 0.245644, 0.247001]	[0.00658538, 0.0148512, 0.023117] [0.238884, 0.249574, 0.260263]
242				
243	[0, 0, 0] [0, 0, 0]	[0.503355, 0.506689, 0.510024] [0, 0, 0]	[0.520857, 0.522942, 0.525028] [0.247001, 0.248358, 0.249714]	[0.0107567, 0.0169369, 0.023117] [0.238884, 0.249574, 0.260263]
244	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.516685, 0.518771, 0.520857] [0.247001, 0.248358, 0.249714]	[0.00658538, 0.0148512, 0.023117] [0.238884, 0.249574, 0.260263]
311				
312				
313				
314	[0, 0, 0] [0, 0, 0]	[0.502726, 0.505651, 0.508576] [0, 0, 0]	[0.516685, 0.518771, 0.520857] [0.252428, 0.253785, 0.255142]	[0.00658538, 0.0148512, 0.023117] [0.238884, 0.249574, 0.260263]
321				
322				
323				
324				

Table 10: Subdivision level 3 (continued)

Subset	P ₀	P ₁	P ₂	P ₃
331				
332				
333				
334				
341				
342				
343				
344				
411	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.5, 0.502086, 0.504171] [0.249714, 0.251071, 0.252428]	[0, 0.00722277, 0.0144455] [0.239244, 0.249817, 0.260391]
412	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.504171, 0.506257, 0.508343] [0.249714, 0.251071, 0.252428]	[0, 0.00930844, 0.0186169] [0.239244, 0.249817, 0.260391]
413	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.504171, 0.506257, 0.508343] [0.252428, 0.253785, 0.255142]	[0, 0.00930844, 0.0186169] [0.239244, 0.249817, 0.260391]
414	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.5, 0.502086, 0.504171] [0.252428, 0.253785, 0.255142]	[0, 0.00722277, 0.0144455] [0.239244, 0.249817, 0.260391]
421	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.508343, 0.510428, 0.512514] [0.249714, 0.251071, 0.252428]	[0, 0.0113943, 0.0227886] [0.238869, 0.249677, 0.260485]
422	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.512514, 0.5146, 0.516685] [0.249714, 0.251071, 0.252428]	[0.00241403, 0.0128424, 0.0232708] [0.238869, 0.249677, 0.260485]
423	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.512514, 0.5146, 0.516685] [0.252428, 0.253785, 0.255142]	[0.00241403, 0.0128424, 0.0232708] [0.238869, 0.249677, 0.260485]
424	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.508343, 0.510428, 0.512514] [0.252428, 0.253785, 0.255142]	[0, 0.0113943, 0.0227886] [0.238869, 0.249677, 0.260485]
431	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.508343, 0.510428, 0.512514] [0.255142, 0.25639, 0.257639]	[0, 0.0114879, 0.0229758] [0.238869, 0.249677, 0.260485]
432	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.512514, 0.5146, 0.516685] [0.255142, 0.25639, 0.257639]	[0.00241403, 0.0128424, 0.0232708] [0.238869, 0.249677, 0.260485]
433	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.512514, 0.5146, 0.516685] [0.257639, 0.258888, 0.260137]	[0.00241403, 0.0128424, 0.0232708] [0.238869, 0.249677, 0.260485]
434	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.508343, 0.510428, 0.512514] [0.257639, 0.258888, 0.260137]	[0, 0.0114879, 0.0229758] [0.238869, 0.249677, 0.260485]
441	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.5, 0.502086, 0.504171] [0.255142, 0.256366, 0.25759]	[0, 0.00730655, 0.0146131] [0.239262, 0.249823, 0.260384]
442	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.504171, 0.506257, 0.508343] [0.255142, 0.256366, 0.25759]	[0, 0.00939223, 0.0187845] [0.239262, 0.249823, 0.260384]
443	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.504171, 0.506257, 0.508343] [0.25759, 0.258815, 0.260039]	[0, 0.00939223, 0.0187845] [0.239262, 0.249823, 0.260384]
444	[0, 0, 0] [0, 0, 0]	[0.5, 0.505012, 0.510024] [0, 0, 0]	[0.5, 0.502086, 0.504171] [0.25759, 0.258815, 0.260039]	[0, 0.00730655, 0.0146131] [0.239262, 0.249823, 0.260384]

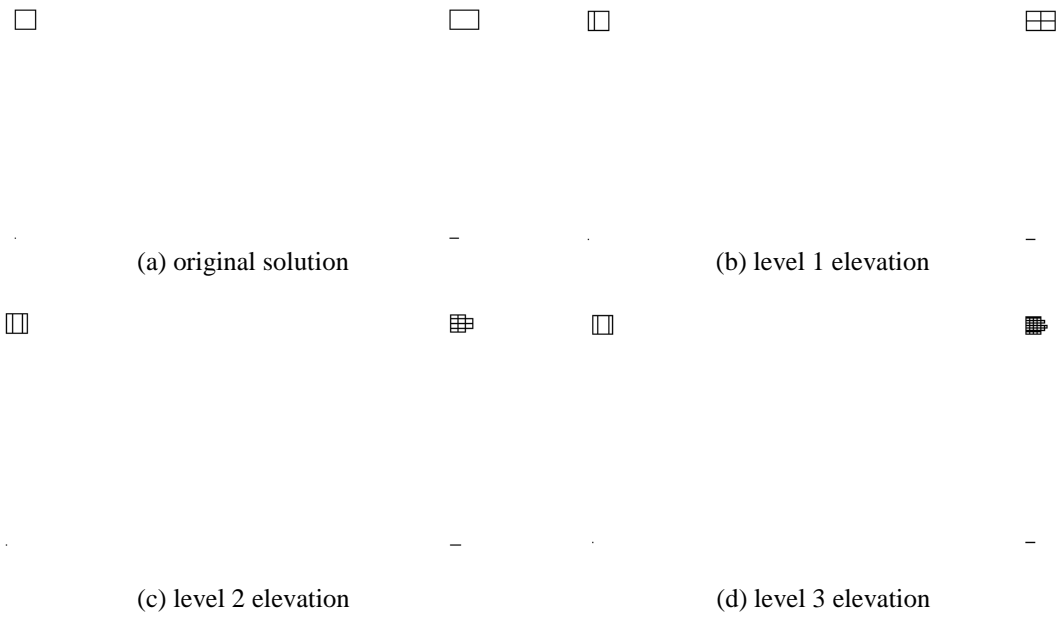


Figure 59: Comparison of different levels of subdivisions

It can be seen that the subdivision until level 3 leaves out some infeasible sub-regions that are included in the initial result. Subdivision provides a more accurate design based on existing constraints. Figure 60 shows the refined bracket design by interval subdivision.

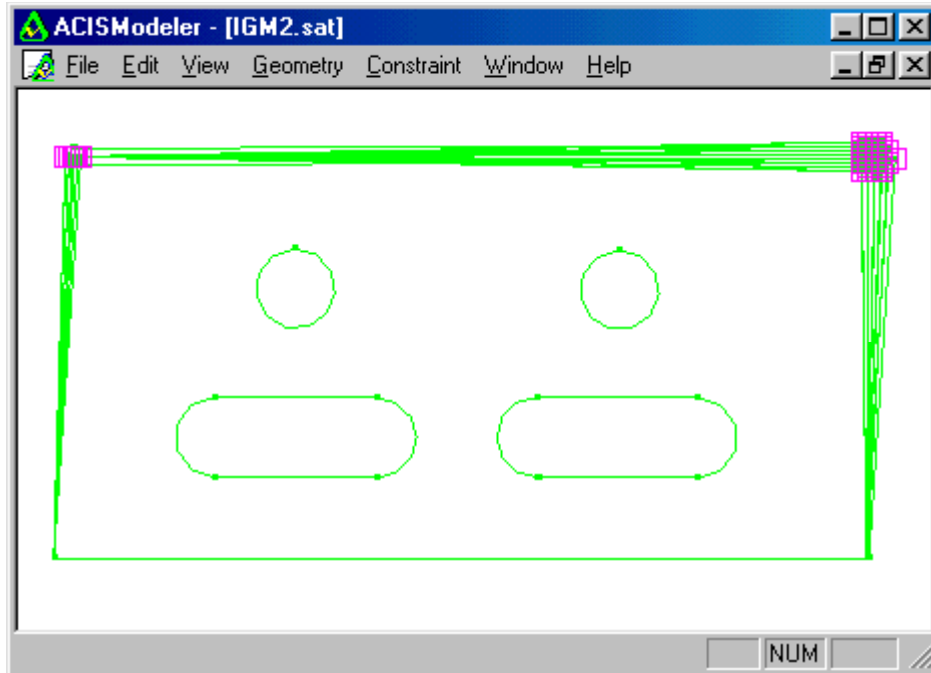


Figure 60: Refined bracket design by subdivision

6.5.2 Constraint Re-specification

Another way to contract a solution is to change or add valid constraints to narrow down feasible regions. Feasibility and effectiveness should be considered simultaneously. Constraint modification depends on sensitivity analysis, while adding constraints is largely dependent on human users' preferences. One basic question is to differentiate *active* and *inactive* constraints. Active constraints scope the actual range of solution while inactive constraints have certain levels of slackness. At the beginning of interval computation, all constraints are active if a sufficiently large initial region is given. As the iteration proceeds, some constraints turn to be inactive. The decision of which constraints to be modified is based on the selection of active constraints.

Lemma: For a constraint set $p = \{f(\mathbf{X}) = \mathbf{Y} \text{ and } g(\mathbf{X}) = \mathbf{Z}\}$, the subset $f(\mathbf{X}) = \mathbf{Y}$ with respect to a solution $\mathbf{D} \subset \mathbf{X}$ is inactive if $f(\mathbf{D}) \subset \mathbf{Y}$ and $g(\mathbf{D}) \supseteq \mathbf{Z}$.

Proof:

Suppose S_1 and S_2 are actual solution sets of f and g respectively, and S is the actual solution set of p . Given that $f(S_1) = Y$ and $f(D) \subset Y$, because of the property of inclusion monotonic, $S_1 \supset D$. Similarly, $D \supseteq S_2$. Thus, $S_1 \supset S_2$. \square

As illustrated by Figure 61, subset f is inactive and g is active in case (a); both f and g are active in case (b); and f is active and g is inactive in case (c).

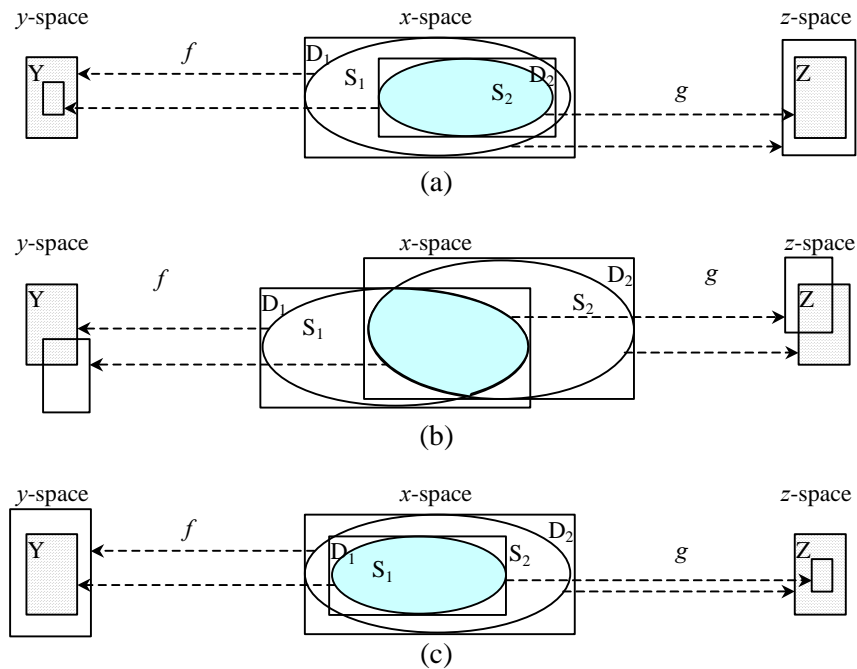


Figure 61: Relations of two constraint subsets

In the numerical example of Section 6.4.4, it can be proven that the last constraint is inactive based on the above lemma.

This chapter presents a new geometric modeling scheme, IGM, based on the interval representation and analysis, in which all parameters of geometric modeling (coordinates as well as parametric constraints) are non-trivial-width interval values instead of fixed values. It can be used as a generic representation of numerical constraints during the process of conceptual design, detailed design, and design optimization. It avoids rigid constraints and thus reduces the chance of conflicts between constraints. It relaxes the restriction of under-constrained and over-constrained situations for variational geometry. Constraint-driven interval geometric modeling captures more information about constraints for optimization and decision making during the design process. IGM provides a possible interoperable way of design data representation and integration for different design stages.

7.0 IMPLEMENTATIONS AND TESTS

The implementation and test of various concepts are conducted within the framework of a distributed design environment – *Pegasus* [184]. Pegasus is a service-oriented concurrent engineering system which aims to aid customers, designers, manufacturers, suppliers, and other stakeholders to participate in the early stage of product design so as to reduce the new product development cycle time. Since design is an interdisciplinary and complicated process, it requires various contributions from all of the participants. This system integrates the services required during the product development period, such as conceptual and detailed design, various analysis and tests for assemblability, manufacturability, material, ergonomics, and logistics. Pegasus is an open system that possesses good extensibility, portability, interoperability, scalability, and transparency. If certain new services are required, the system can incorporate the new functional units with no or little changes. Furthermore, heterogeneous service providers can work collaboratively and harmoniously within the system over the Internet without compatibility problems.

7.1 Service Architecture of Pegasus

Concurrent engineering requires the collaboration of various engineering and non-engineering disciplines, such as aesthetics, drafting, materials, manufacturing processes, quality, marketing, maintenance, and government regulations. There have been many computational

tools in those different areas. These CAD/CAM/CAE tools can be plugged in Pegasus system to form a distributed product development environment, which provides engineering services over the Internet. An important approach to achieve transparent transaction is to define engineering service protocols explicitly. Thus, the service information can be represented and interpreted correspondingly by each individual tool according to these service protocols at the application level.

Service is defined as a process that provides a functional use for a person, an application program, or another service in the system. Services should be specified from the functional aspect of service providers. To make an existing tool available online or to build a brand new tool for such a system, services associated with this tool should be defined. The service transaction among service providers, service consumers, and the service manager within Pegasus system is illustrated in Figure 62. Once a service is registered at a central administrative manager, called the Service Manager, it is then available within the whole system. This process is *service publication*. When a service consumer within the system needs a service, it will request a lookup service from the Service Manager. This process is *service lookup*. If the service is available, the service consumer can request the service from the service provider by the aid of Service Manager. Most importantly, this service triangular relationship should be built at run-time. The service consumer (client) does not know the name, the location, or even the way to invoke the service from the service provider (server) during the system and tools development period.

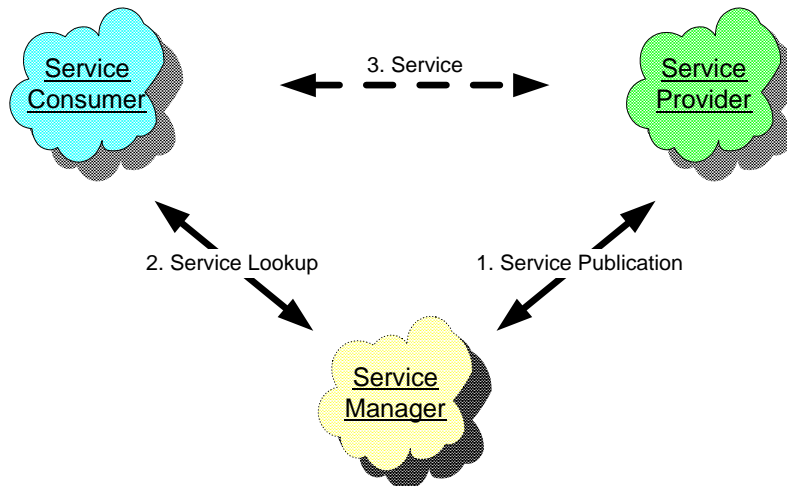


Figure 62: Service triangular relationship

The collaboration between engineering tools is established and executed based on the characteristics of services that can be provided. For example, the relationship between a CAD tool and a Finite Element Analysis (FEA) tool is based on the FEA service that the FEA tool can provide for the CAD tool. The Service Manager, on the other hand, offers service publication and lookup for service providers and consumers.

Service providers that provide different services such as drafting, assembly, manufacturing, analysis, optimization, procurement, and ergonomics can be developed independently. As showed in Figure 63, servers that provide different engineering services (which are represented by nodes) are linked by the Internet. Each node in this network may require or provide certain engineering services. Thus, it could be a client or a server for different services depending on whether it is the recipient or the provider of such a service. The client/server relationship is determined at run-time. The system is open for the future expansion and extension, in case that more services are available. The notion of service-oriented collaboration lets the Pegasus system have appropriate flexibility. Plug-and- Play (PnP) is an important consideration of this structure.

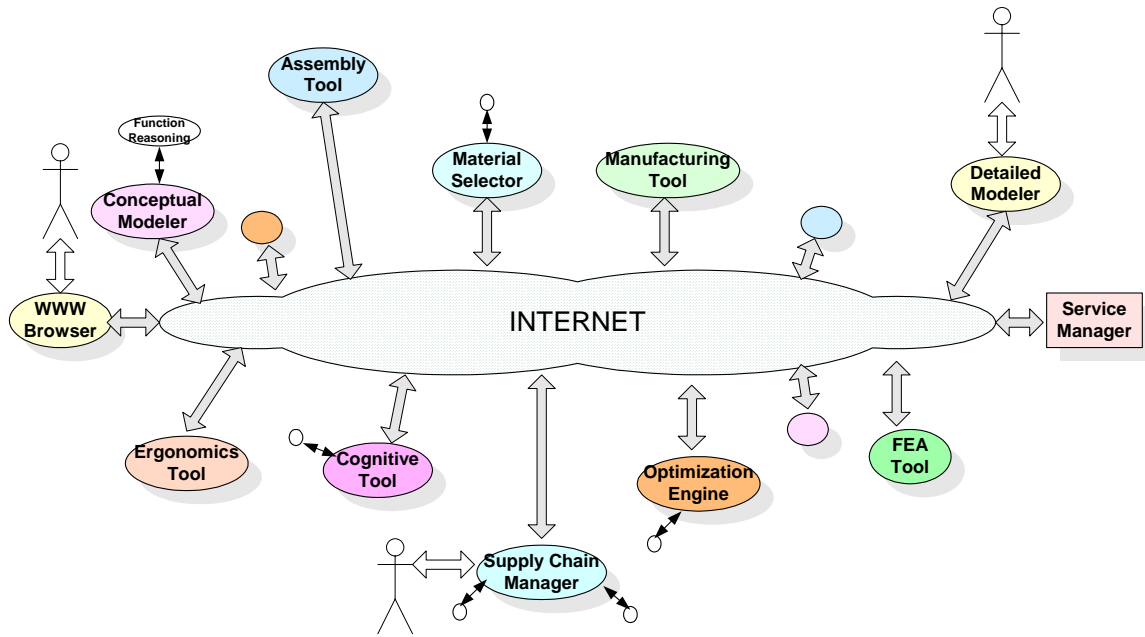


Figure 63: Pegasus system architecture

Service publication and lookup are the primary services provided by the Service Manager. As depicted in Figure 64, service publication for service providers includes *name publication*, *catalog publication*, and *implementation publication*. Name publication service is similar to the *white-page* service provided by telephone companies, by which the name of the service provider is published. Catalog publication service is similar to the *yellow-page* service: the name and the functional description of the service are published. Implementation publication service is the procedure by which the service provider makes its implementation and invocation of services public so that clients can invoke the service at run-time. Service lookup for service consumers includes *name lookup*, *catalog lookup*, and *interface lookup*. Name lookup service is provided so that consumers can locate the service providers based on the service names. Catalog lookup service is for those consumers who need certain services according to their needs and specifications but do not know the names of the services. Interface lookup service is to provide a way such that consumers can check the protocols of how to invoke the service. For example, if a

consumer wants to do welding analysis but does not know the name of the service (e.g., thermo-structural finite element analysis), it can use the catalog lookup service. To query the analysis procedure of thermo-structural analysis from an interface repository, it may use interface lookup service to find out input parameters, return type, etc., of this service.

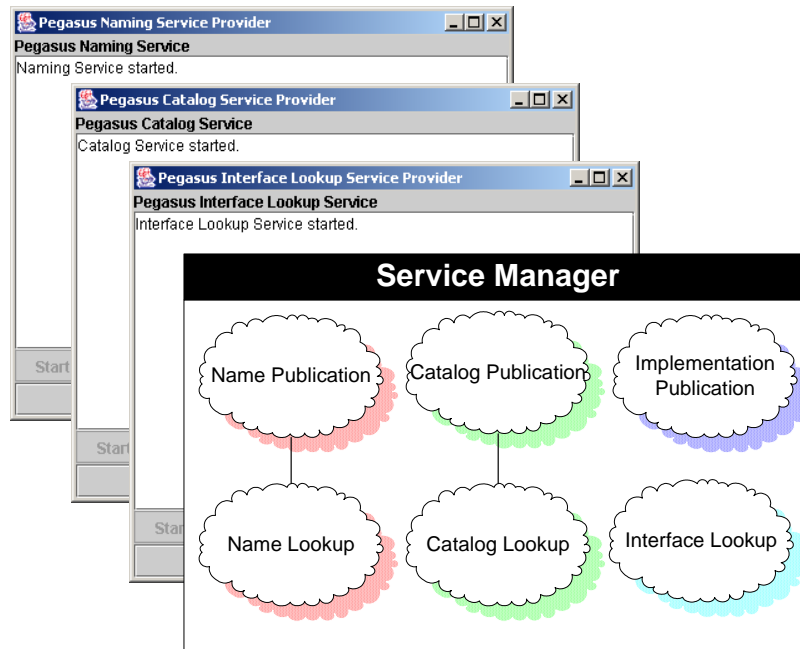


Figure 64: Services provided by Service Manager

Design data transfer and transaction among servers can be completed based upon various distributed computing protocols, such as HTTP, CORBA, Distributed Component Object Model (DCOM), and Simple Object Access Protocol (SOAP).

In today's software engineering arena where heterogeneity is inevitable, *openness* is an essential characteristic for a distributed computational architecture. It allows complex software systems to be efficiently developed, deployed, and maintained. An open collaborative product engineering system, such as CAD, CAM, and CAE, should incorporate the following features:

- (1) Compliance with industry standards of programming, communication, networking, system management, and interfaces between applications and system services;
- (2) Portability of applications across different computer operating systems so that the system can be easily adopted by various service providers and end users;
- (3) Scalability of application performance and throughput such that the system is applicable for either large enterprises with large-scale projects or individuals with simple artifacts;
- (4) Interoperability which is independent of hardware platforms, operating systems, network protocols, and application formats such that service providers can be developed independently with different programming languages;
- (5) Extensibility which allows new functionalities for existing service providers or new service providers to be added into the system such that the system is expandable in the future.

To ensure that Pegasus is an open system, the implementation should support the above five features. In this work, CORBA is employed as basic computational protocol to achieve openness. Data transfer and transaction are implemented by CORBA as illustrated in Figure 65. The components in this distributed system have peer-to-peer relationships with each other. CORBA serves as glue to integrate the whole system. It provides good features of transparency for collaborative computation. Computationally intensive applications can be distributed across the network. From the end user's point of view, distributing application components between clients and servers does not change the look and feel of one single application.

The time sequence diagram of service transaction in Pegasus is listed in Table 11 and illustrated in Figure 66. It includes the processes of service binding (the service provider publishes a service at the service manager), service resolution (the service consumer looks up a

service from the service manager), and service execution (the service provider provides the requested service).

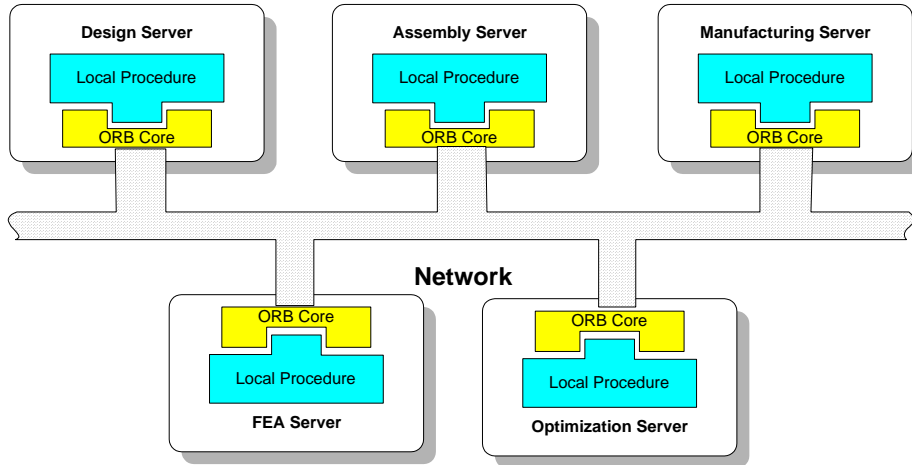


Figure 65: Peer-to-peer relationship among service providers

Table 11: Service sequence in a client/server transaction

No.	Service Transactions
1	Service provider requests service binding from service manager
2	Service manager provides service binding to service provider
3	Service consumer requests service resolution from service manager
4	Service manager provides service resolution to service consumer
5	Service consumer requests service from service provider
6	Service provider provides service to service consumer

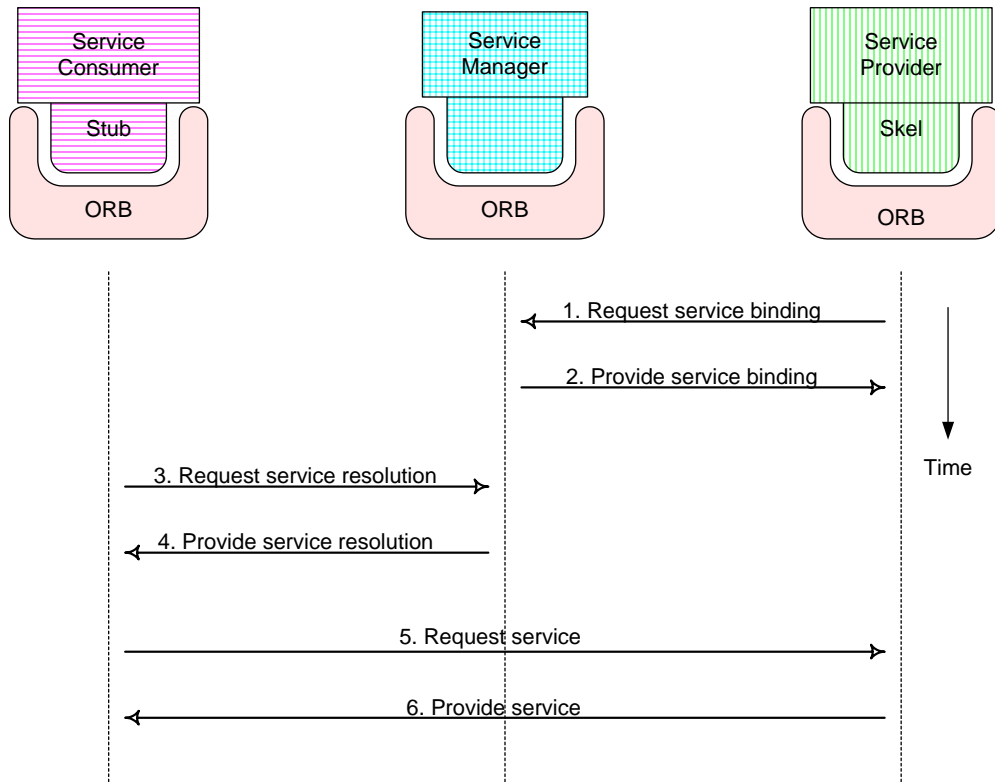


Figure 66: Sequence diagram of a service

7.2 UL-PML Scheme in Collaborative Design

UL-PML is a distributed CAD data scheme, which allows geometric and non-geometric entities, structures, and constraints to be created, stored, and queried in a distributed fashion. This allows information transfers at the basic entity level rather than the component level. It provides a flexible way for information exchange intelligently and accumulatively without losing logical integrity. In a top-down approach, a PML tree can provide different levels of detail. In a bottom-up approach, loosely coupled linkage allows lean information transfer.

PML can be applied in two approaches. One is to use PML as a part of the native data structure in geometric modelers. The other is to translate design data from various formats of existing CAD systems into PML models for information exchange. In this research, both of the two approaches are implemented and tested. A prototype of geometric modeler using PML as the native data structure is built. Mechanisms of lean information transfers based on protocols of HTTP and CORBA are developed. Distributed geometry and constraint information can be linked based on the UL-PML scheme. The translation mechanism between ACIS data structure to PML model is developed and tested in an ACIS modeler prototype.

7.2.1 PML Modeler

A native PML modeler is developed completely based on PML data format. Figure 67 illustrates the architecture of the modeler and Figure 68 shows its user interface. Within the modeler, geometry can be generated and processed in the form of a PML tree. Data is stored and transferred in PML file format.

Users interact with the system in a regular design mode, while the Data Manager is responsible for local PML tree processing and transparent remote data query. Compatibility to computer and Internet standards is necessary to make an open system. The PML modeler uses industry standards for file transfer and remote data access. Design information transfer in PML is independent of network data transmission.

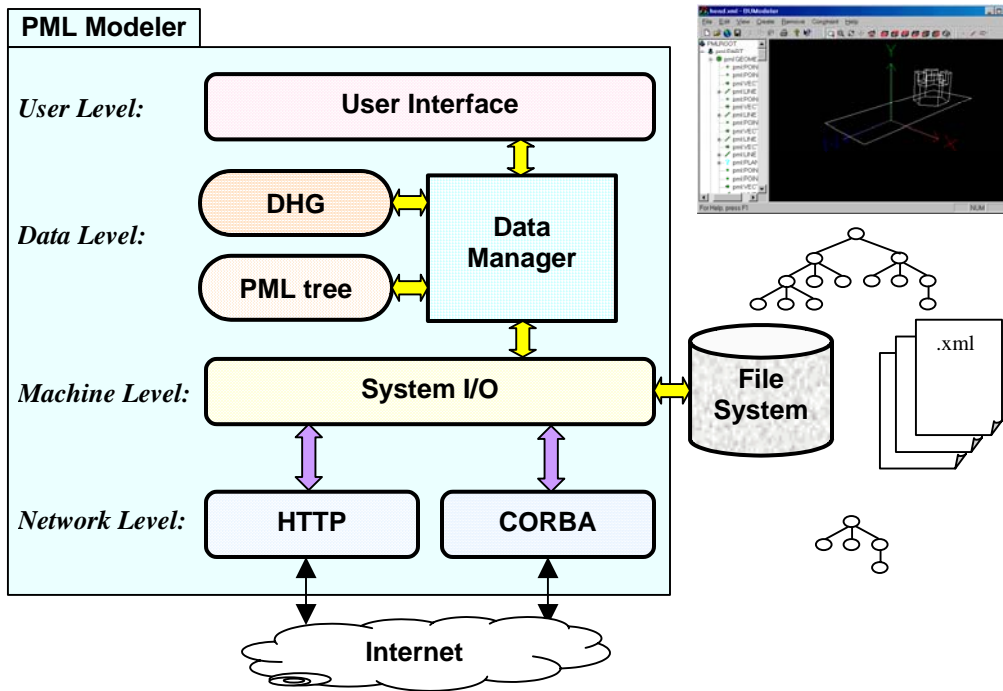


Figure 67: Architecture of PML modeler

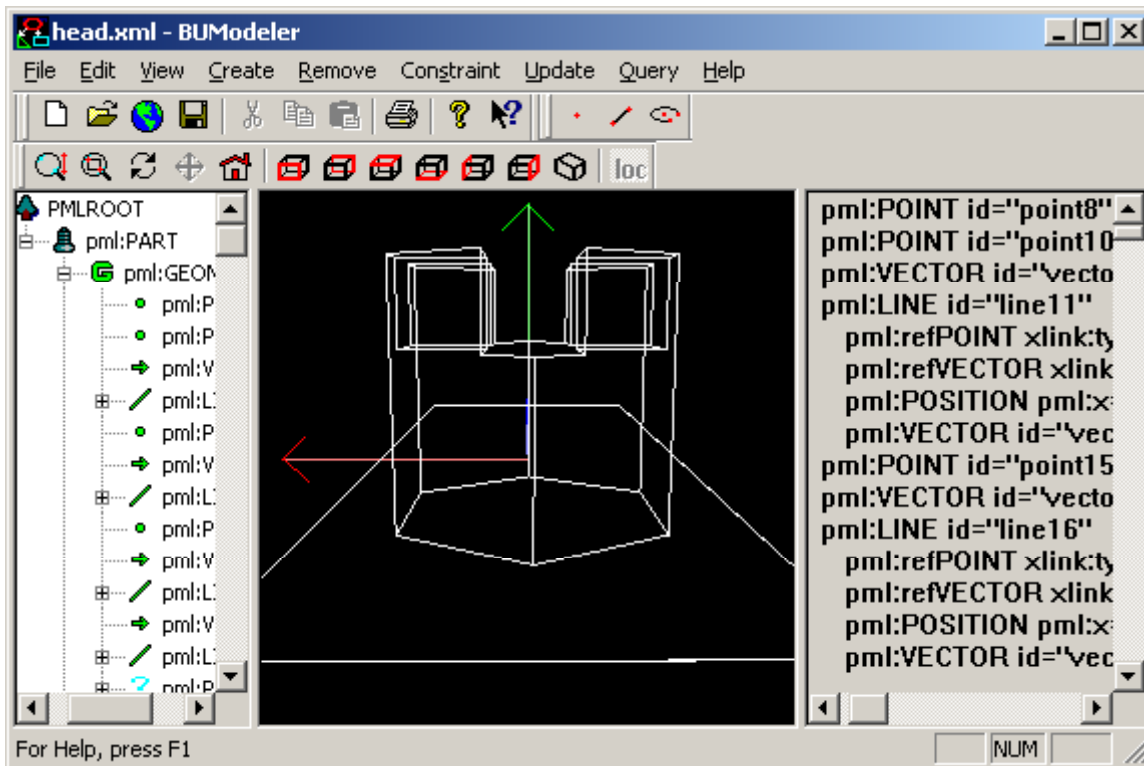


Figure 68: Interface of PML modeler

7.2.2 Lean Information Transfer Based on HTTP

PML information transfer can use a variety of network protocols such as HTTP. HTTP is a widely used application protocol for web service. PML remote data access and selective information transfer based on HTTP are developed in the PML modeler. In the example of Figure 22, the process of face information transfer between the two groups is illustrated in Figure 69.

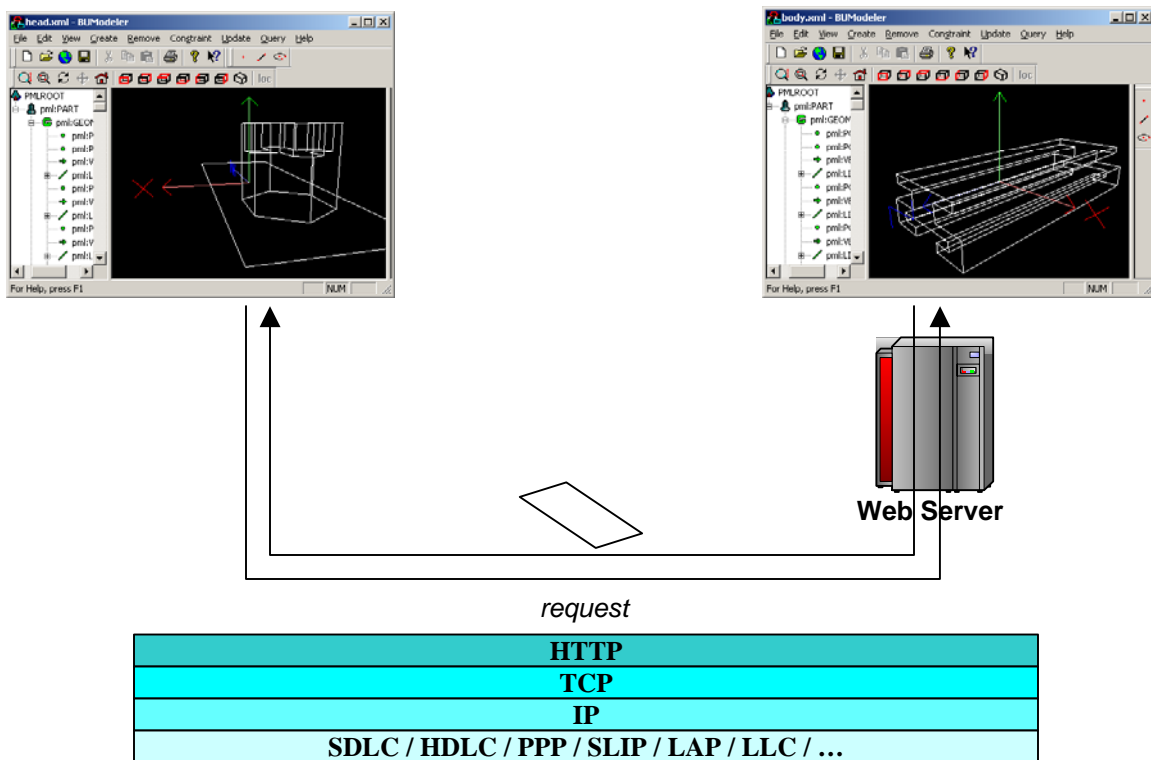


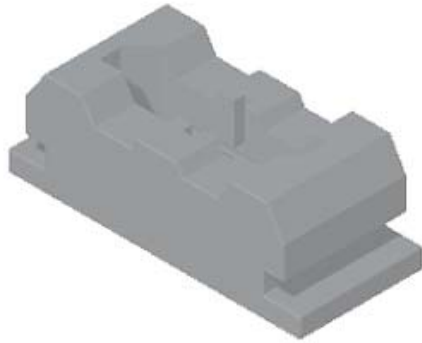
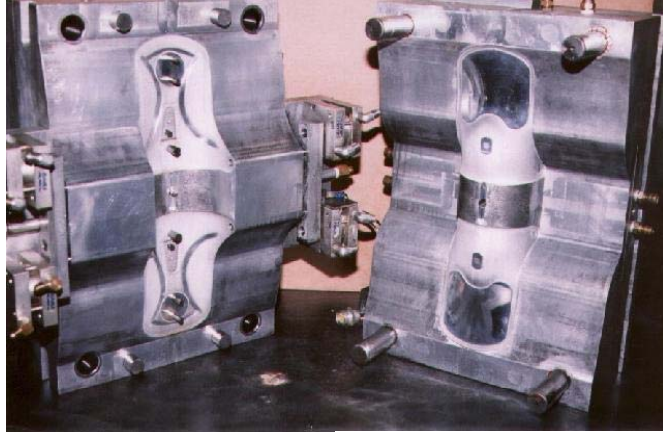
Figure 69: Lean information transfer base on HTTP

7.2.3 Lean Information Transfer Based on CORBA

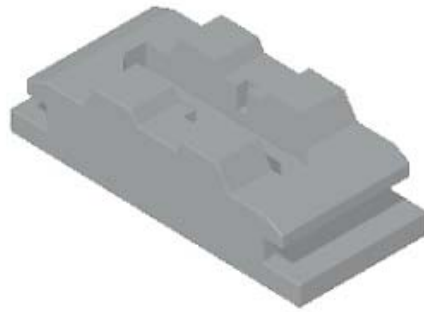
The PML modeler also supports lean information transfer based on protocols of CORBA. Different from HTTP requests, ORB requests have fat-client architecture. The client/server data transfer can be transparently completed through ORB brokers. Clients do not have to specify the IP addresses of the target PML references.

For example, a pair of moulds (Figure 70) are designed separately by two groups. Some contacting faces of the two parts must geometrically match each other. In the UL-PML scheme, links between the faces in *Mould2* and the corresponding ones in *Mould1* can be built. Thus the geometry and topology information about these faces in the *Mould2* can be fetched from *Mould1* to maintain the consistency. In this linkage relation, *Mould1* (Figure 71) is at the server site. Once it is published in the library (Figure 72) for data sharing, it is available for references.

In order to meet the surface match requirement, *face504*, *face978*, and *face1004* in *Mould2* are specified to refer to *face3*, *face239*, and *face286* in *Mould1* (Figure 73). Three faces and six bounding edges in Table 12 as well as the corresponding geometry are transferred to the client site through data sharing agents.



(a) mould1



(b) mould2

Figure 70: A pair of moulds in collaborative design

Table 12: Selective topology transferred to *Mould2*

Location	Name	Entity Type	Reference	Link Type
<i>mould2.xml</i>	<i>face504</i>	Face	<i>mould1.xml#face3</i>	simple
	<i>face978</i>	Face	<i>mould1.xml#face229</i>	simple
	<i>face1004</i>	Face	<i>mould1.xml#face286</i>	simple
	<i>edge508</i>	Edge	<i>mould1.xml#edge13</i>	simple
	<i>edge518</i>	Edge	<i>mould1.xml#edge23</i>	simple
	<i>edge593</i>	Edge	<i>mould1.xml#edge55</i>	simple
	<i>edge635</i>	Edge	<i>mould1.xml#edge168</i>	simple
	<i>edge588</i>	Edge	<i>mould1.xml#edge60</i>	simple
	<i>edge640</i>	Edge	<i>mould1.xml#edge163</i>	simple

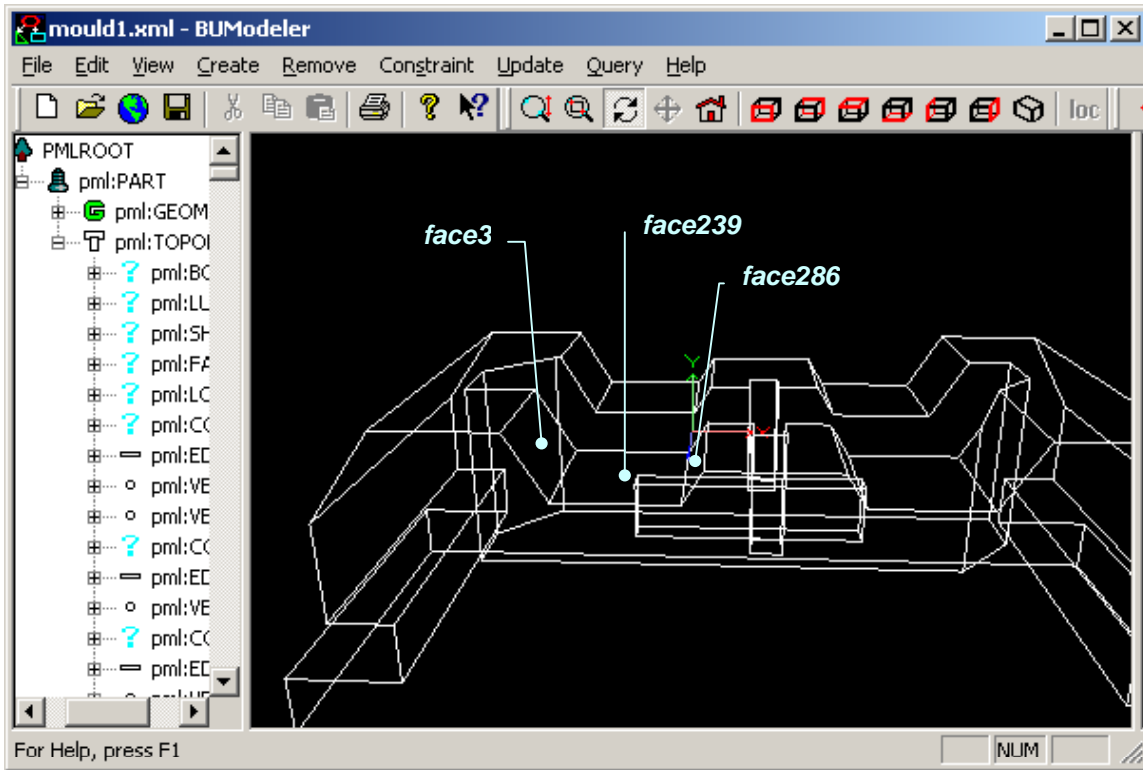


Figure 71: The first mould designed at the server site

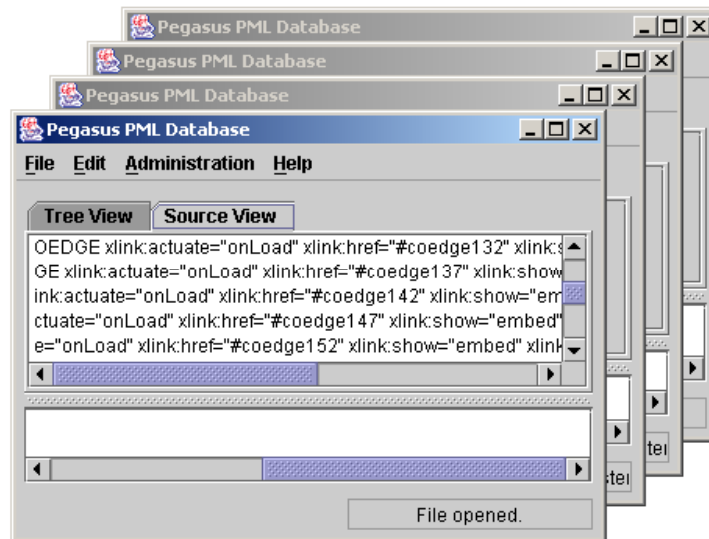


Figure 72: Design library for data sharing

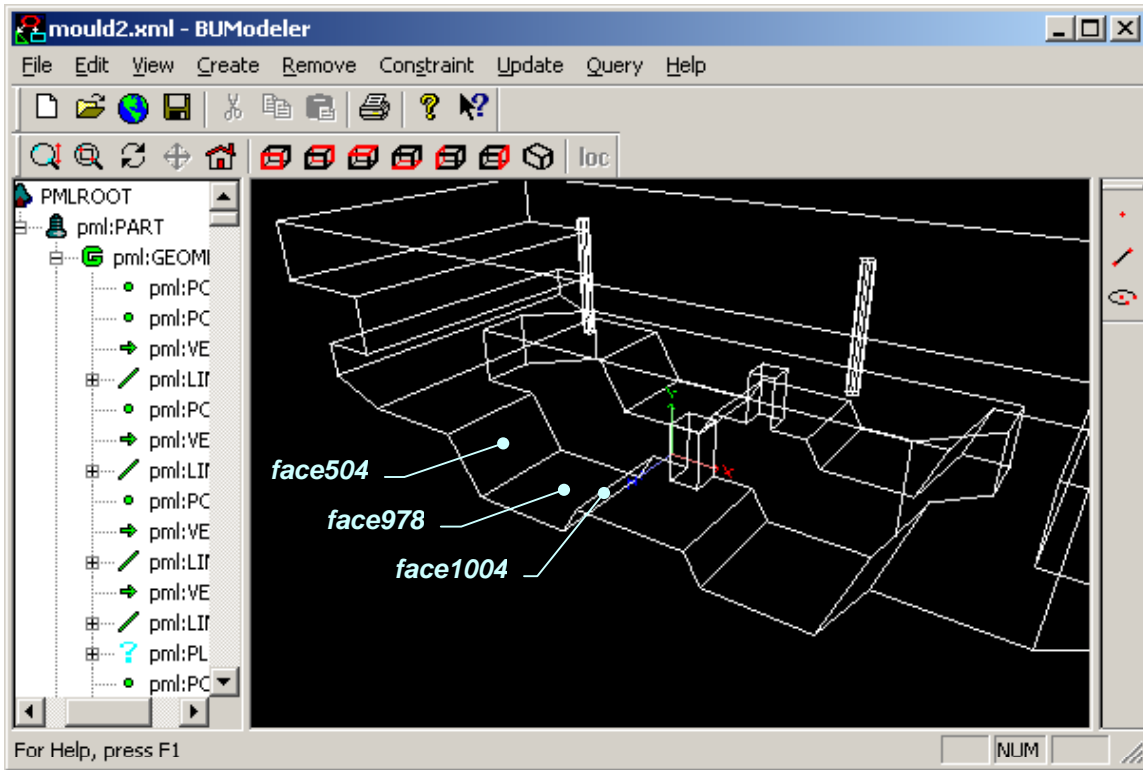


Figure 73: The second mould designed at the client site

If there is any change about the three faces of the first mold (Figure 74), the update of the second mold will be done automatically because of the linkage (Figure 75). Note that for each update, only PML nodes of three faces and six edges are transferred across networks.

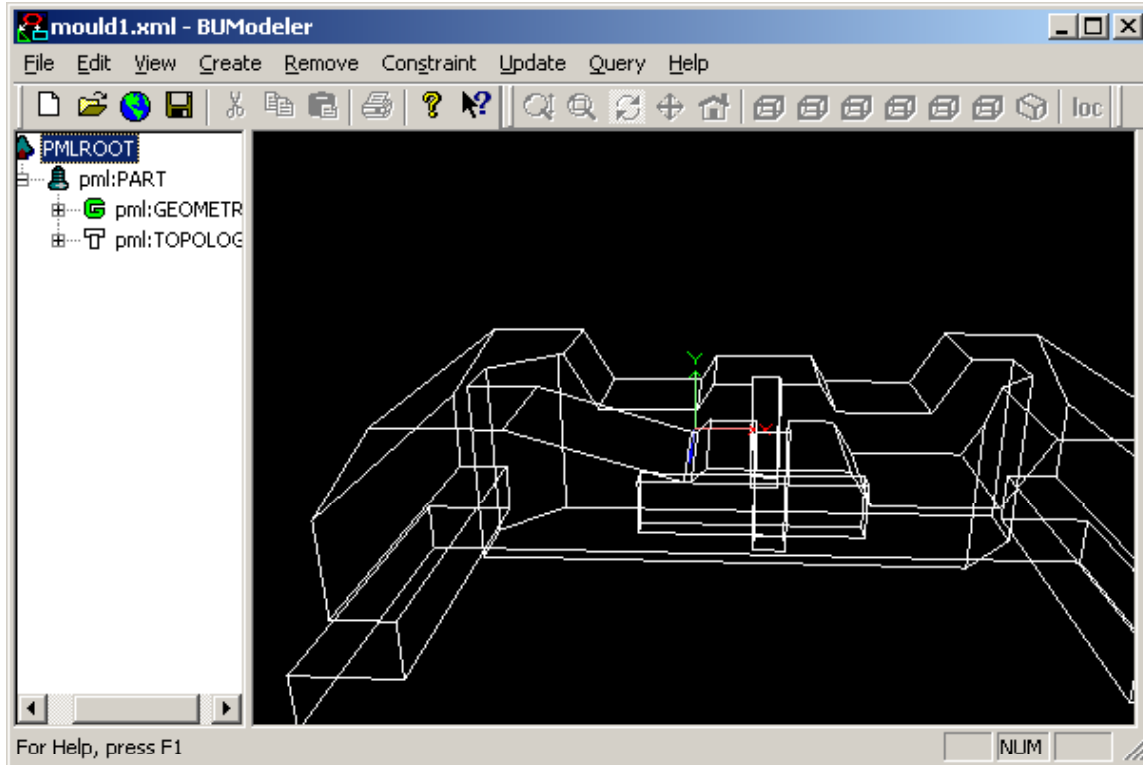


Figure 74: Updated design of the first mould

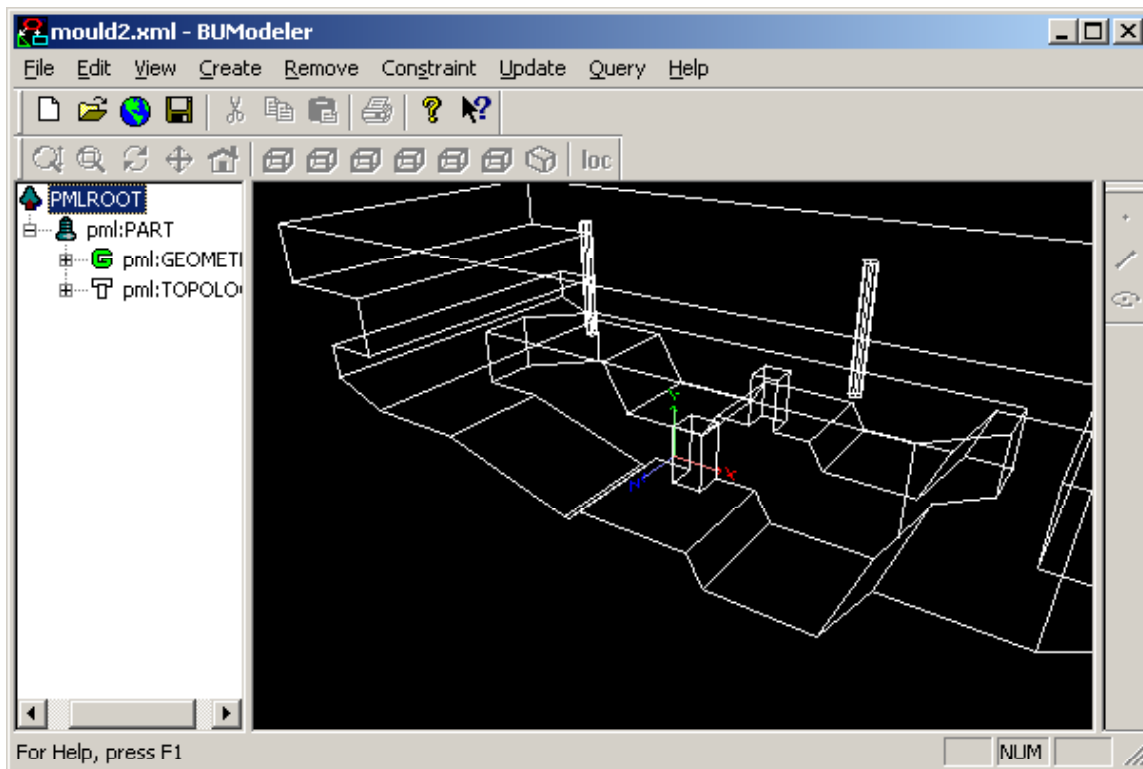


Figure 75: Updated second mould by translating corresponding references

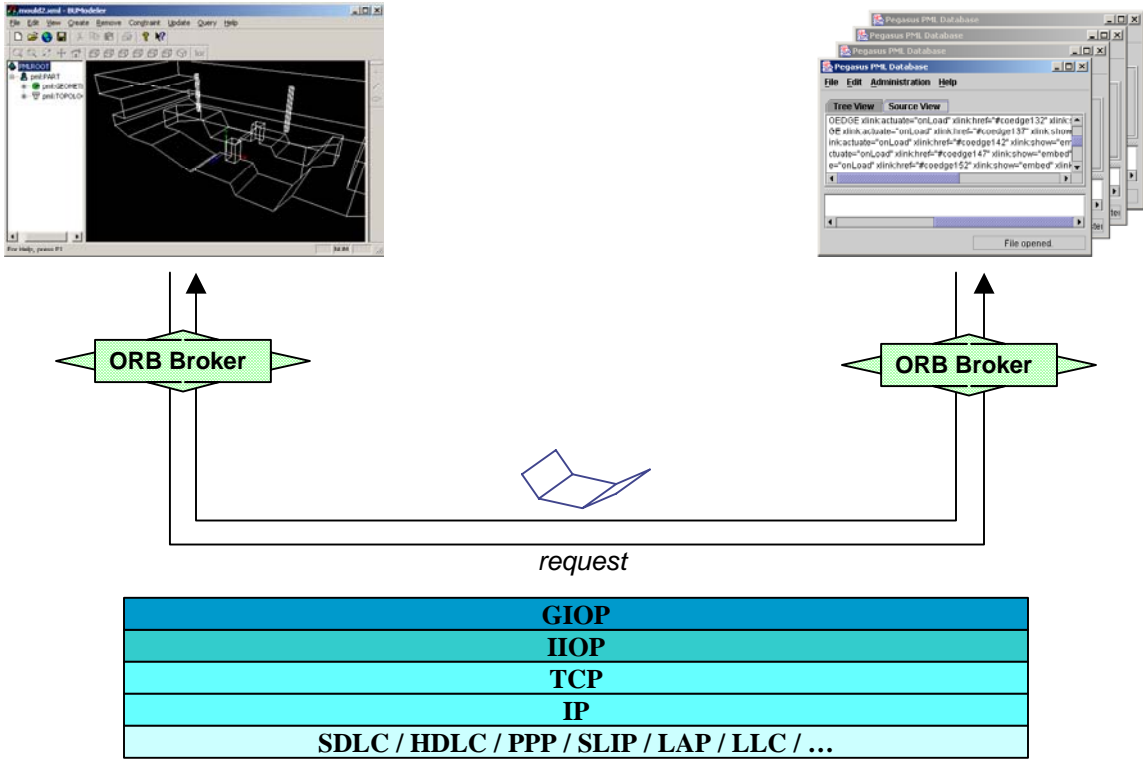


Figure 76: Lean information transfer based on CORBA

7.2.4 Distributed Design Information Integration

The PML model provides a mechanism to link distributed design information. Elements of entities, relations, and constraints can be located locally as well as remotely. As illustrated in Figure 23 and Figure 24, geometry and topology can be distributed in different files. Similarly, constraints can be linked either locally or remotely. As shown in Figure 77, Figure 78, and Figure 79, constraints can be defined either in the same file or in different files. Some example constraints are listed in Table 13.

As a result, design can be created or modified without processing a large amount of data. Design information is modeled in an extensible and uniform format. The efficiency of collaborative design then can be improved by the UL-PML scheme.

Table 13: Constraint examples in *mold1.xml*, *mold2.xml*, and *constr.xml*

Location	Name	Type	Source	Target	Direction
<i>mold1.xml</i>	<i>a1</i>	angle (geometric)	# <i>face3</i>	# <i>face1</i>	bi-directional
	<i>a2</i>	angle (geometric)	# <i>face4</i>	# <i>face2</i>	bi-directional
	<i>d1</i>	distance (geometric)	# <i>face5</i>	# <i>face3</i>	bi-directional
	<i>d2</i>	distance (geometric)	# <i>face5</i>	# <i>face3</i>	bi-directional
	<i>e1</i>	Equation (non-geometric)	# <i>a2</i>	# <i>a1</i>	unidirectional
	<i>e2</i>	Equation (non-geometric)	# <i>d2</i>	# <i>d1</i>	unidirectional
<i>mold2.xml</i>	<i>a1</i>	angle (geometric)	# <i>face3</i>	# <i>face1</i>	bi-directional
	<i>a2</i>	angle (geometric)	# <i>face4</i>	# <i>face2</i>	bi-directional
	<i>d1</i>	distance (geometric)	# <i>face5</i>	# <i>face3</i>	bi-directional
	<i>d2</i>	distance (geometric)	# <i>face5</i>	# <i>face3</i>	bi-directional
	<i>d3</i>	distance (geometric)	# <i>face5</i>	<i>mold1.xml</i> # <i>face5</i>	bi-directional
<i>constr.xml</i>	<i>e1</i>	Equation (non-geometric)	<i>mold2.xml</i> # <i>a1</i>	<i>mold1.xml</i> # <i>a1</i>	unidirectional
	<i>e2</i>	Equation (non-geometric)	<i>mold2.xml</i> # <i>a2</i>	<i>mold1.xml</i> # <i>a2</i>	unidirectional
	<i>e3</i>	Equation (non-geometric)	<i>mold2.xml</i> # <i>d1</i>	<i>mold1.xml</i> # <i>d1</i>	unidirectional
	<i>e4</i>	Equation (non-geometric)	<i>mold2.xml</i> # <i>d2</i>	<i>mold1.xml</i> # <i>d2</i>	unidirectional

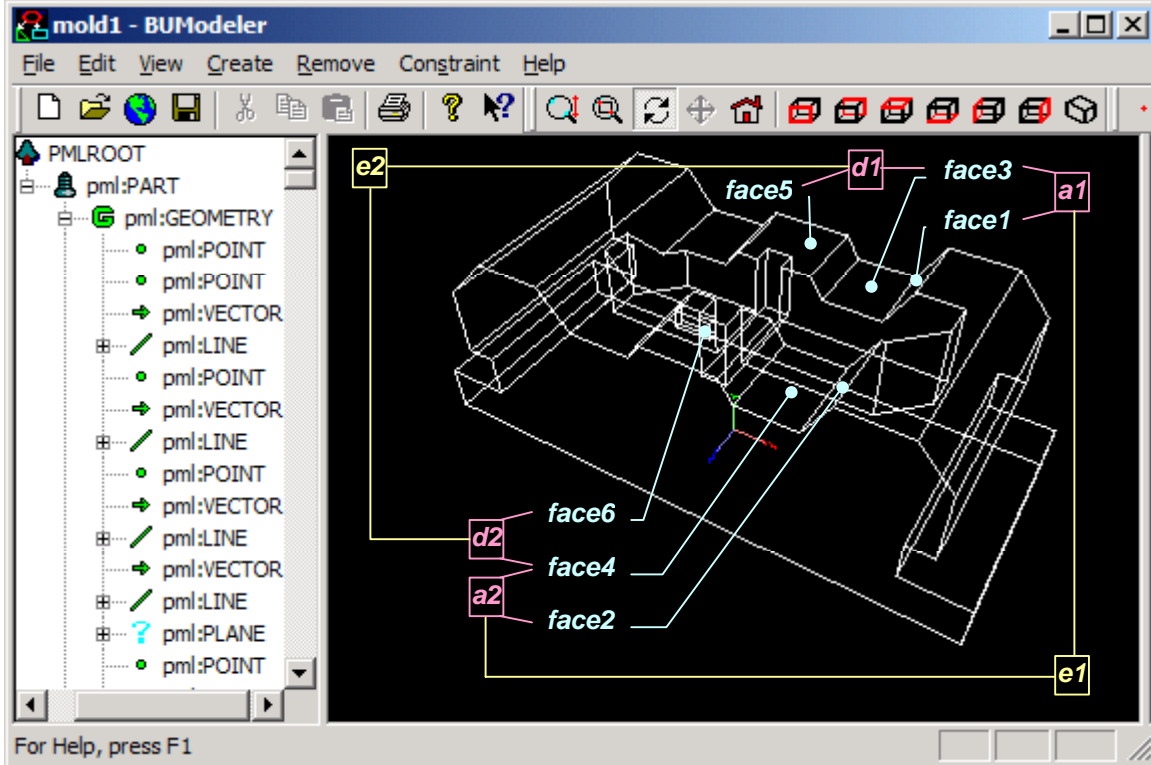


Figure 77: Design constraints in *mold1.xml*

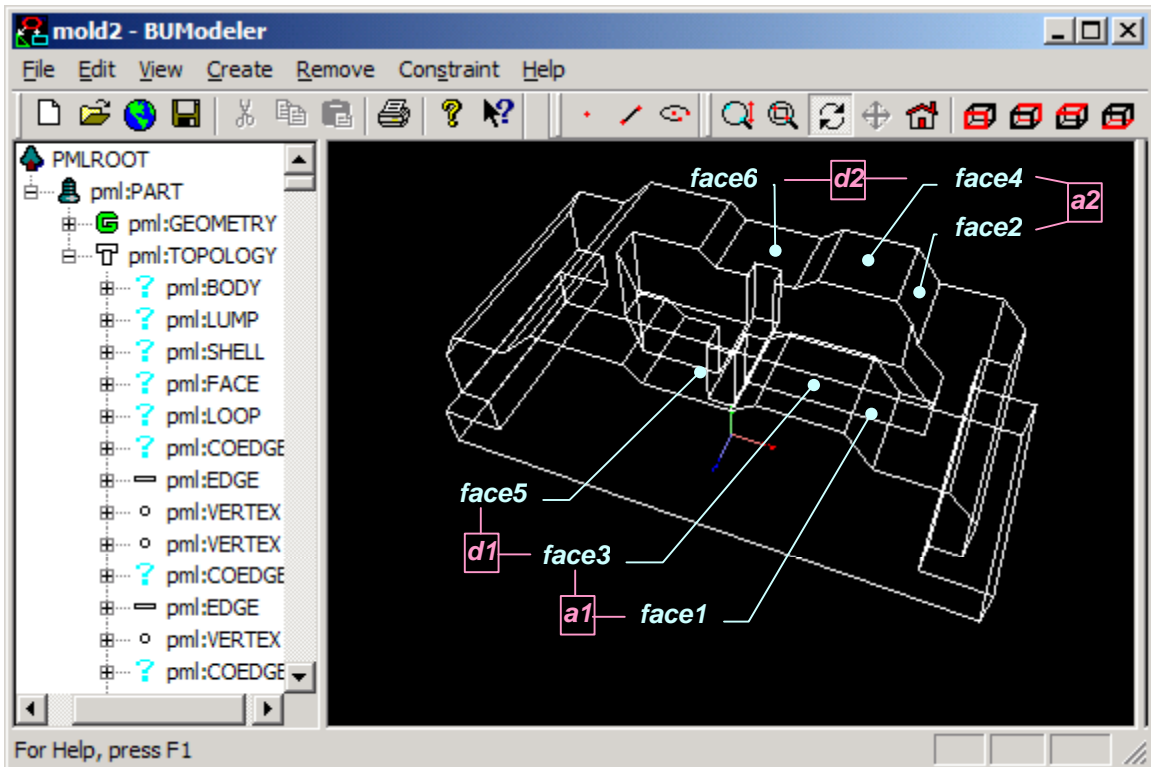


Figure 78: Design constraints in *mold2.xml*

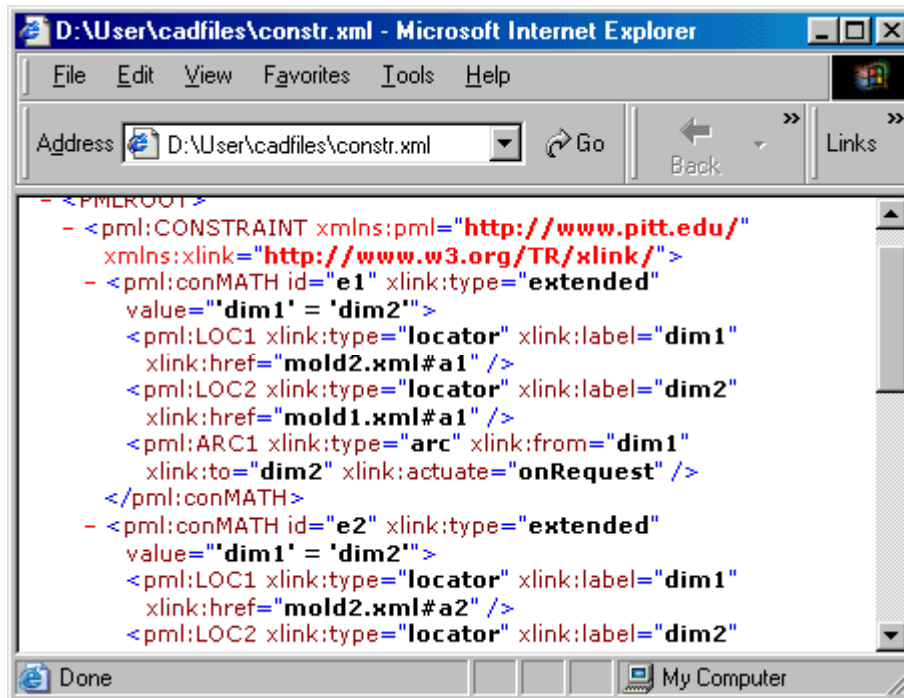


Figure 79: Design constraints in *constr.xml*

7.2.5 Mapping Between Native CAD Data Models and PML Model

PML model can be a medium of data transfer for distributed design. Different CAD systems can exchange lean design information based on PML model. As illustrated in Figure 80, CAD systems can map both geometric and non-geometric information to PML tree structure. Selectively, a PML sub tree is transferred within a collaborative design environment. To integrate PML model and existing CAD systems, translation is needed to map native data structures of different CAD systems to the PML structure.

To demonstrate the possibility of integration between PML and current CAD systems, a geometric modeler prototype based on ACIS[®] kernel is developed, and translation between PML and ACIS model is implemented. Figure 81 shows the architecture of the ACIS modeler.

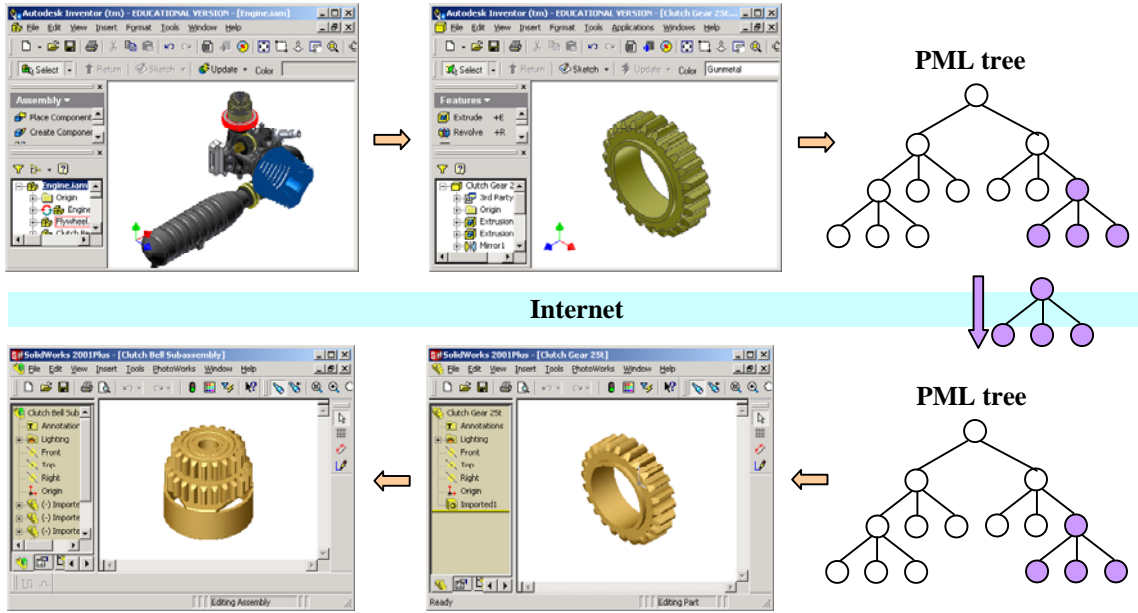


Figure 80: PML model as a medium for selective information transfer between CAD systems

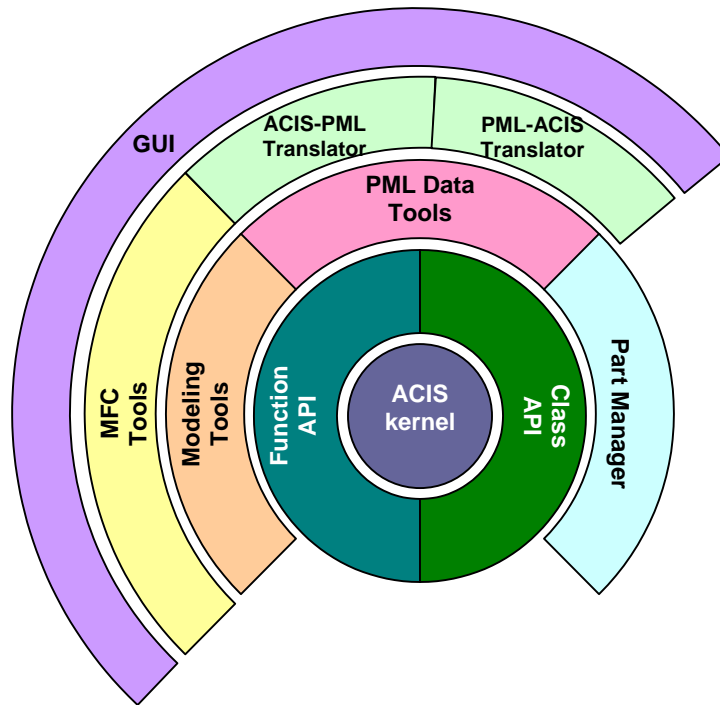


Figure 81: The architecture of the ACIS modeler

In the example shown in Figure 82, a jig model is in the ACIS modeler. It can be translated into PML model by the ACIS-PML translator and read by the PML modeler, as in Figure 83. Within a collaborative design system, geometric and non-geometric information in PML format then can be transferred among groups. If any PML data is received from other parties, it can be read and processed either by a PML modeler (as in Figure 84), or by the PML-ACIS translator and an ACIS model can be built (as in Figure 85). Note that it is more efficient and secure to transmit only partial data in PML across networks, while complete PML models reside in local systems.

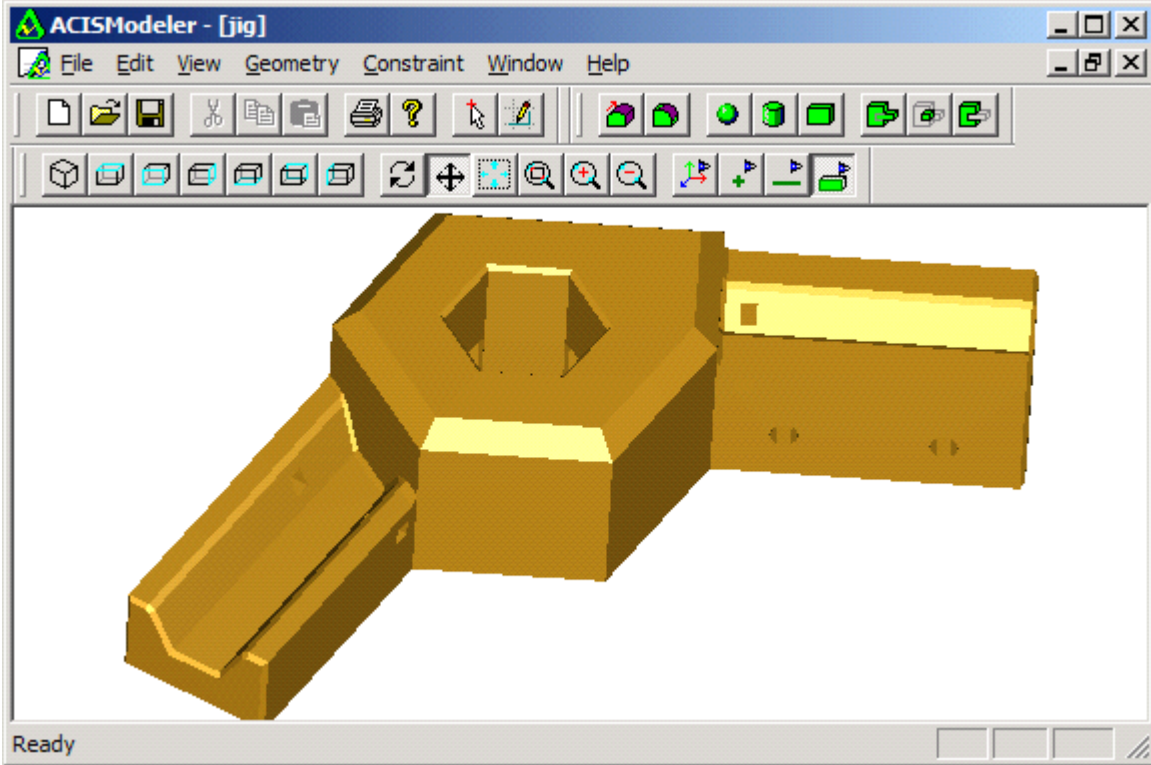


Figure 82: A jig model in SAT format in the ACIS modeler

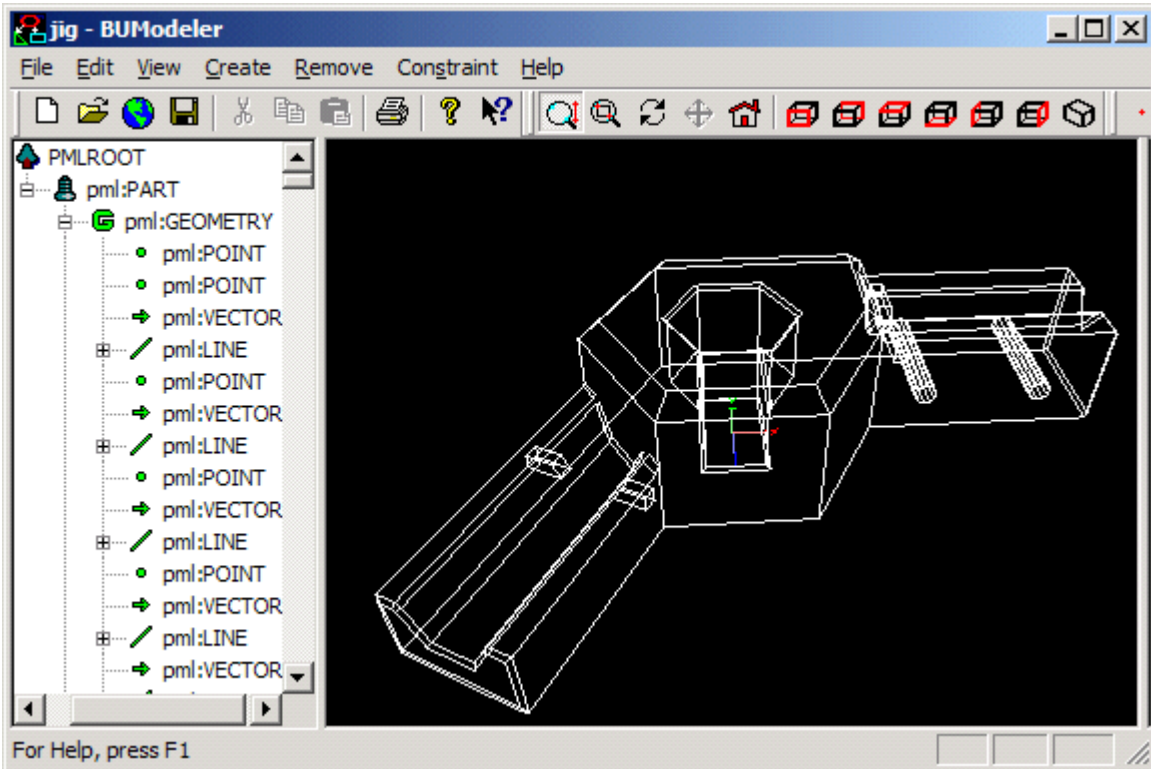


Figure 83: The translated jig model in PML format in PML native modeler at server site

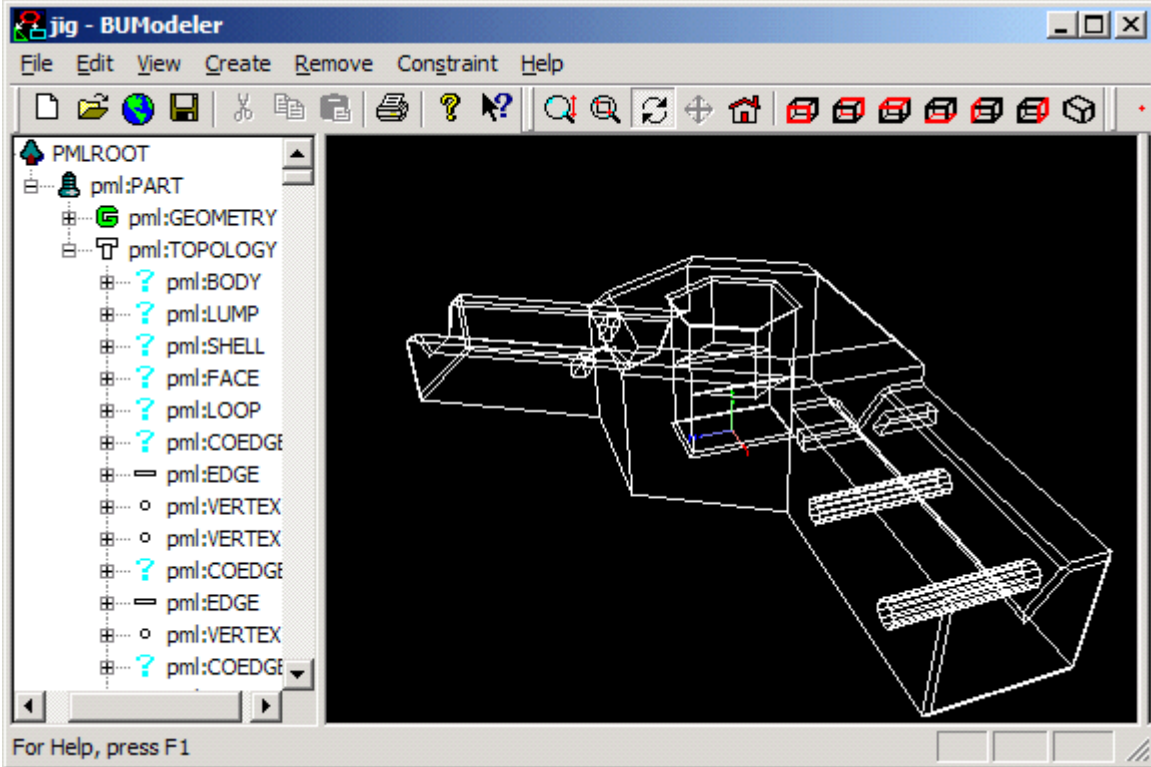


Figure 84: The jig model in PML format received at client site

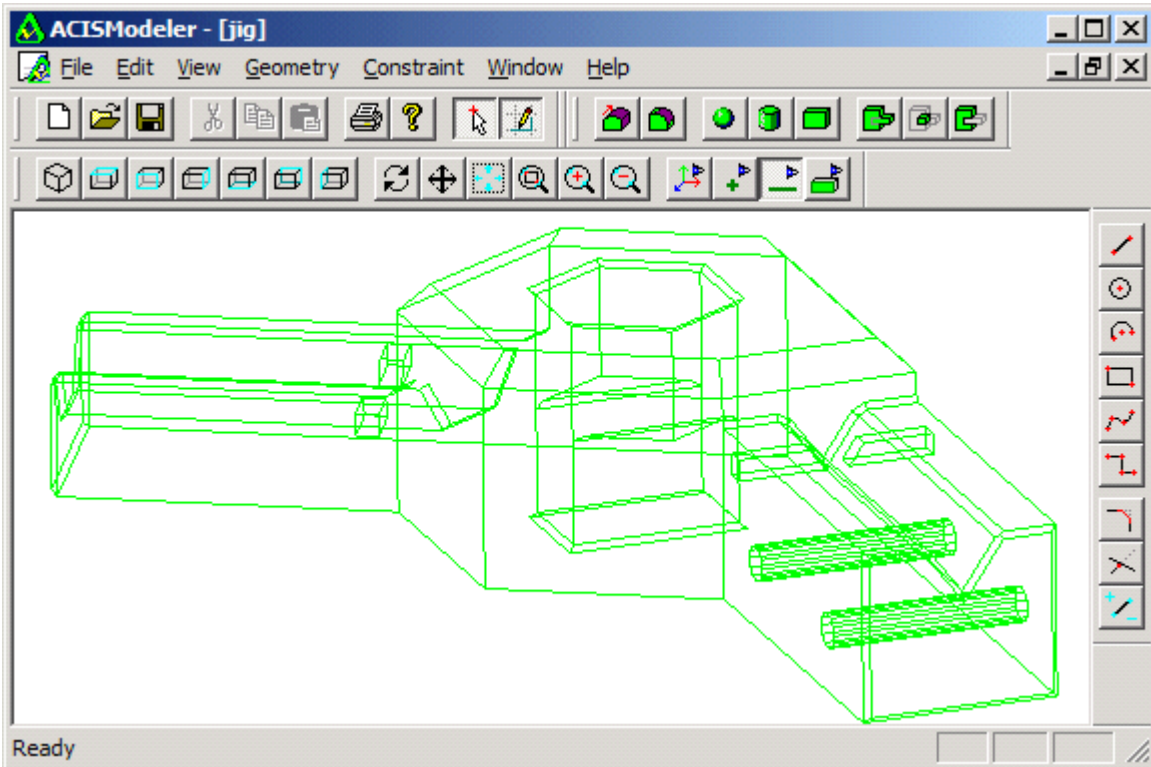


Figure 85: The translated jig model in the ACIS modeler

7.2.6 Constraint Propagation and Management

The UL-PML scheme has a determined format for constraint representation to support design knowledge interoperability. Geometry constraints are attached to low-level topological entities symbolically to eliminate ambiguity. Non-geometric constraints are attached to high-level entities such as constraints, features, and parts. The generic and uniform constraint representation allows constraints to be propagated effectively to support collaborative design.

To allow various constraints to be propagated within a collaborative design environment, a central constraint management unit is needed to maintain a library of constraint standards. Collaborators need to publish the format of new constraints in the library, thus other parties can check the library and understand the usage. UL-PML scheme provides a uniform and extensible constraint format so that interoperable constraint representation is possible. A Constraint Library service provider is developed to preserve the constraint format standard in the Pegasus system. It stores the syntax of multidisciplinary constraints such that format information of constraints is available for lookup. The functionalities include: 1) provide constraint format lookup service; 2) maintain the standards of constraints; 3) add, remove, and update constraint format.

The Constraint Library provides the constraint format query service. As demonstrated in Figure 86, once the Constraint Library service provider is registered in the system (step 1), constraint formats can be looked up. A client, which can be a geometric modeling system, needs to find out how to represent material constraints in a standard way such that other parties can understand the syntax and meaning. It requests a constraint format query lookup service through a Service Manager (step 2). After getting the server information from the Service Manager (step 3), it sends query of constraint *conMATERIAL* to the Constraint Library (step 4). The Constraint

Library looks in its database, finds out the format of constraint *conMATERIAL*, and sends the format back to the client (step 5).

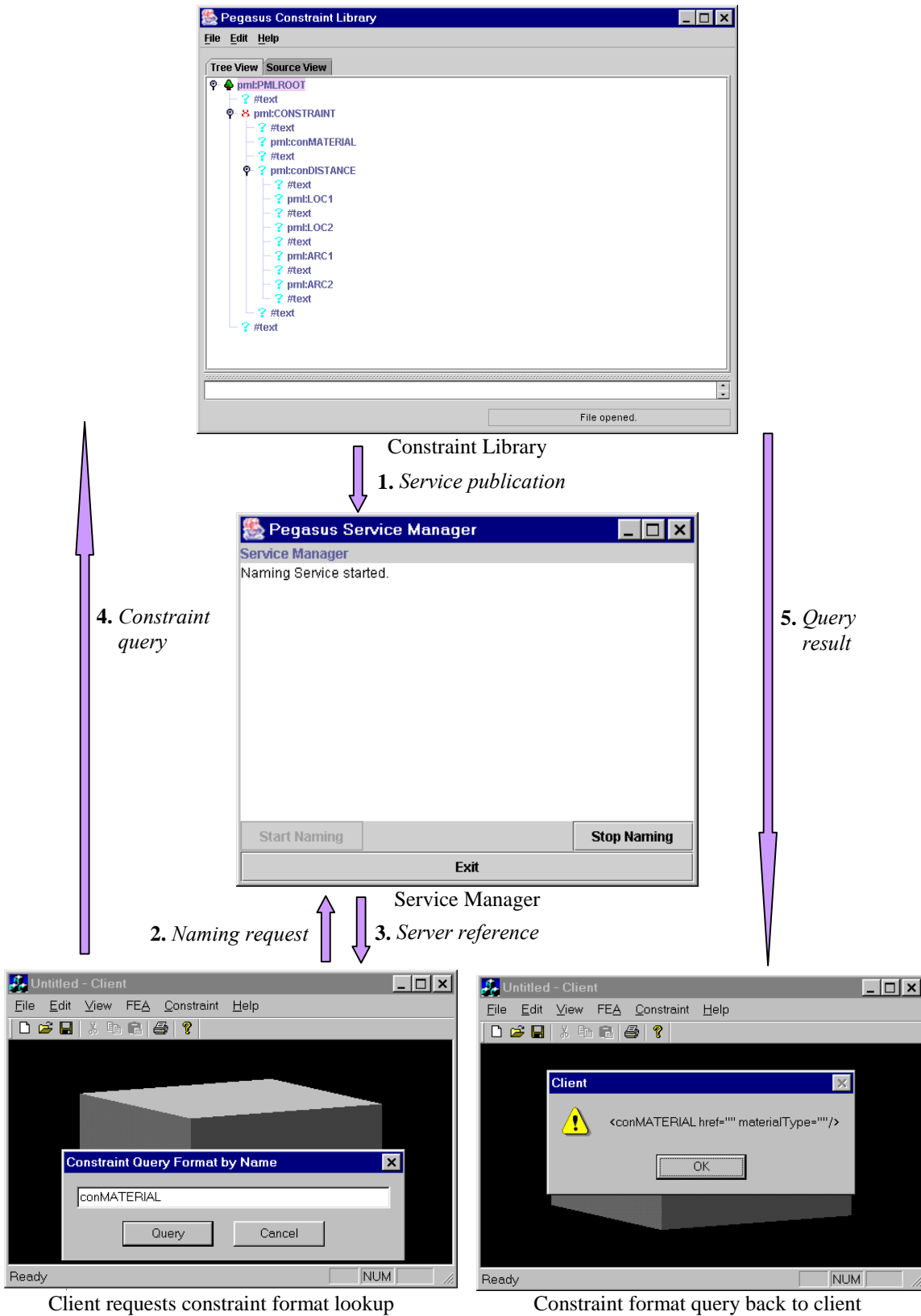


Figure 86: Process of constraint format query from constraint library

In summary, the UL-PML scheme provides a generic design information model for collaborative design. It models geometric and non-geometric entities, relations, and constraints in a uniform and extensible format. Based on the PML model, design information can be transferred without transmitting large amounts of data, which increases the efficiency of information sharing in a distributed environment.

Nevertheless, the flexibility of the PML model is achieved at the cost of computation in applications. The major cost of lean product information transfers is the overhead related to information interpretation and consistency maintenance. Partial data information, which is based on linkage in PML, needs to be interpreted by referring to the source. In order to maintain data consistency at the client site and the server site, a signal mechanism should be developed so that server can notify clients when an information update is needed.

8.0 SUMMARY AND FUTURE WORK

In this dissertation, incompleteness, improcessability, and inconsistency issues related to design information interoperability for collaborative design are researched. Research topics include network-conscious geometric information modeling, design knowledge and specification capturing, and multidisciplinary constraint representation integration within geometric data. The objective is to create new interoperability mechanisms and methodologies to enable the evolution of CAD data modeling from current standalone CAD to Internet-based collaborative design.

To tackle the design information incompleteness and improcessability problems, a UL-PML scheme is developed to capture geometric and non-geometric relations among entities by explicit links in PML. These links allow references between entities to be built across the boundary of files and physical locations. This model enables heterogeneous design information to be distributed at different physical locations in a collaborative environment, and virtually integrated through networks. This distributed format makes selective design information transfers possible among design collaborators, which includes fundamental topology and geometry elements, structural relations, as well as high-level design information such as design features, geometric and non-geometric constraints, components, and assembly data. PML utilizes standard XML syntax. Schemas of PML are defined for entities and relations. Graph decomposition method is developed to map graph-structured entities and relations to tree-structured PML. PML trees allow distributed design information elements to be processed, stored, and queried easily. Geometry-based entity naming method is developed to maintain

universal linkage among design entities such that relations within a file and among files are stable and persistent for different design sessions at multiple locations.

The UL-PML scheme can be employed as a part of collaborative design information infrastructure because of its simplicity, extensibility, system independence, and openness. Commonly used CAD models can be mapped to PML models so that lean information exchange and partial query can be performed through PML. Intelligent information sharing and design reuse thus are supported. A prototype of the PML modeler is developed, which uses PML as the native data structure. The PML data structure is independent of network protocols, which can make it an open data protocol in a collaborative design environment. Remote data access and query based on HTTP and lean information transfers based on CORBA are tested. PML is intended to be a medium for heterogeneous CAD systems in design data exchange. Translation between different native data structures and PML is needed to apply UL-PML scheme in current CAD systems. A translation mechanism between ACIS data structure and PML for explicit geometry is developed. And it is tested based on an ACIS modeler prototype.

To address the design information inconsistency and improve design reliability and quality, a method of interval value numerical constraint representation is developed so that computational errors and ambiguity can be reduced and robustness of geometric modeling can be improved. Additionally, to model design uncertainty and inexactness, and to build the model interoperability for different design stages, an IGM scheme is developed based on the non-trivial-width interval representation and analysis. Soft constraints and preferences are integrated in constraint-driven systems. Algorithms for solving IGM constraints are developed.

The IGM scheme aims to provide a generic numerical constraint representation for conceptual design, detailed design, and design optimization. Preferences and constraints are

embedded in data models as driving forces and decision-making aids. The IGM kernel for interval representation, operation, and constraint solving is developed and constraint solving algorithms are tested.

In summary, the major contributions of this dissertation are as follows:

- A new network-conscious UL model for geometric and non-geometric entities and relations based on XML is proposed and developed. Design information interoperability is accomplished based on general data interoperability. At the syntax level, the openness of this model is guaranteed. Semantics interoperability is independent from syntax interoperability. This independence provides an open scheme to solve CAD interoperability issues.
- A new concept of distributed CAD information modeling is developed to integrate multidisciplinary design information elements at multiple locations for seamless synthesis and integration.
- A mechanism of lean design information modeling and intelligent information sharing at the entity level is developed so that information with partial integrity can be transmitted within the limited network bandwidth. This mechanism can have physically distributed entities linked across the boundary of data files, thereby introduces a new way of distributed design data modeling, storage, and query.
- Dual-Rep feature representation incorporates intentional and geometric features independently for the improvement of design intent capturing and exchange among collaborators.

- Geometric and non-geometric constraints are represented in an extensible form. Integrated with other design data, constraints eliminate inconsistency and ambiguity, and improve CAD model's completeness.
- A new concept of interval numerical constraint representation is presented, which reduces model inconsistency due to numerical errors, and improves CAD models' reliability and robustness.
- A geometry-based semantic ID method is developed. This method adds semantics of geometry and topology into IDs, therefore increasing the stability of entity reference. It builds the identification framework for the distributed UL model, and improves the naming persistency of current CAD systems.
- A new IGM scheme based on interval representation and analysis is developed to improve model robustness and capture design uncertainty and inexactness. Constraint-driven interval geometric modeling supports more design interaction for optimization and decision-making. IGM establishes a generic approach for interoperable numerical constraint representation for the entire design cycle.

As extensions of this work, several research issues can be studied further. Application of PML in heterogeneous systems is to be researched further, including feature-based and explicit modeling systems. Research on meta-information about PML distributed data network is necessary for information search and management. Compression and encryption of PML needs to be studied. Feature and constraint schemas need to be defined and standardized. The method of surface mapping for semantic ID is important to generalize the persistent ID system. In IGM, methods for large-scale problem solving and algorithms improvement are essential for

commercial application. Interval width selection and optimization for numerical constraint imposition is necessary for interactive IGM.

The research of this dissertation is summarized in Table 14.

Table 14: Research Summary

Problems	Research Solutions & Methodologies	Contributions	Test / Validation	Broader Impact	Future Extensions
There are no open and general solutions for interoperability of different proprietary CAD file formats and version control.	XML format is employed to build information interoperability based on data interoperability, where XML syntax is standardized.	Advancement of current CAD translation methods by separating semantics interoperability from syntax interoperability	As a data protocol, PML is tested based on HTTP and CORBA protocols. Data structure mapping between ACIS and PML for polyhedrons is tested.	CAD data and information infrastructure for collaborative design will be established based on computation and Internet standards to maximize the openness and interoperability.	Application of PML in heterogeneous systems
Design collaboration needs to integrate distributed design information.	UL-PML scheme has a distributed file format that links elements of design at multiple locations.	New concept of distributed CAD information modeling for collaborative design	Geometry and constraint files are linked in PML.	Distributed design data model will provide an innovative data network for seamless synthesis and integration.	Meta-information and management
Standalone CAD file formats do not support collaborative design because of inefficient information transfer.	UL-PML scheme models reference of entities and relations by explicit linkage at the lowest level.	New concept to support design data creation, storage, and query with partial integrity and fine granularity	PML modeler transmits entity-level information.	Flexible and loosely coupled design data elements at different levels of detail will allow lean information sharing and real-time collaboration.	Encryption and compression of PML
Design feature is not interoperable by current translation mechanism.	Dual-Rep feature representation captures intentional and geometric features independently.	Advancement of current feature representation schemes in a general approach	UL-PML models incorporate global and local features.	A generic feature representation is essential to capture design intent with good interoperability.	Feature schema definition and standardization
Multidisciplinary design constraints cannot be modeled uniformly and design information is incomplete.	Explicit geometric and non-geometric constraints are represented in an extensible form.	Advancement of current constraint representation schemes in a general approach	UL-PML models incorporate geometric and non-geometric constraints at different entity levels.	Multidisciplinary design constraint representation allows design knowledge sharing, reuse, and propagation.	Constraint schema definition and standardization
Numerical errors generate inconsistent and unreliable geometry.	Interval value numerical constraint representation gives allowance for geometry interpretation and increases the model robustness.	Advancement of improving geometry robustness by new concept of constraint representation	UL-PML models have interval numerical constraint values embedded. IGM kernel is tested.	Reliable geometry is fundamental for design intent capturing.	Optimal interval width selection
Topological ID is not persistent that causes unstableness of geometry.	Geometry-based Semantic ID method is developed to improve naming persistency within a session and between sessions.	Advancement of existing heuristic naming methods for stable modeling and reference	Linear and quadratic surfaces are tested.	Persistent reference and model consistency will maintain a reliable distributed design data infrastructure.	Surface mapping
There is no generic data modeling to support constraint direct imposition at different design stages.	Interval Geometric Modeling scheme is developed to model uncertainty and preferences.	New concepts and methodologies for solving information interoperability issues for the entire design cycle	IGM kernel is built and constraint solving is tested for ~10-constraint problem.	Interoperable conceptual design, detailed design, and design optimization tools will use unified data form to support decision-making.	Large-scale problem solving and efficiency improvement

APPENDICES

APPENDIX I – XML SYNTAX

- [1] document ::= prolog element Misc* /*Document */
- [2] Char ::= #x9 | #xA | #xD | [#x20-#xD7FF] |
[#xE000-#xFFFF] | [#x10000-#x10FFFF] /*Character range */
- [3] S ::= (#x20 | #x9 | #xD | #xA)+ /*White space */
- [4] NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar | Extender
- [5] Name ::= (Letter | '_' | ':') (NameChar)* /*Names and tokens */
- [6] Names ::= Name (S Name)*
- [7] Nmtoken ::= (NameChar)+
- [8] Nmtokens ::= Nmtoken (S Nmtoken)*
- [9] EntityValue ::= ' "' ([^%&"] | PEReference | Reference)* ' "' | /*Literals */
' "' ([^%&'] | PEReference | Refeence)* ' "'
- [10] AttValue ::= ' "' ([^<&"] | Reference)* ' "' | ' "' ([^<&'] | Reference)* ' "'
- [11] SystemLiteral ::= (' "' [^"]* ' "') | (' "' [^']* ' "')
- [12] PubidLiteral ::= ' "' PubidChar* ' "' | ' "' (PubidChar – ' "')* ' "'
- [13] PubidChar ::= #x20 | #xD | #xA | [a-zA-Z0-9] | [-'()+,./:=?;!*@\$_%]
- [14] CharData ::= [^<&]* - ([^<&]* '])>' [^<&]* /*Character data */
- [15] Comments ::= <!-- ((Char – ' - ') | (' - ' (Char – ' - '))) * '--> /*Comments */
- [16] PI ::= '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?>' /*Processing Instructions*/
- [17] PITarget ::= Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
- [18] CDsect ::= CDStart CData CDEnd /*CDATA sections */
- [19] CDStart ::= '<![CDATA['
- [20] CData ::= (Char* - (Char* '])>' Char*)
- [21] CDEnd ::= ']]>'
- [22] prolog ::= XMLDecl? Misc* (doctypedekl Misc*)? /*Prolog */
- [23] XMLDecl ::= '<?xml' VersionInfo EncodingDecl? SDDDecl? S? '?>'
- [24] VersionInfo ::= S 'version' Eq (' "' VersionNum " "' | ' "' VersionNum ' "')
- [25] Eq ::= S? '=' S?
- [26] VersionNum ::= ([a-zA-Z0-9_.:] | ' - ')+
- [27] Misc ::= Comment | PI | S
- [28] doctypedekl ::= '<!DOCTYPE' S Name (S ExternalID)? S? /*Document Type Definition*/
(' [' (markupdecl | DeclSep)* '] S?)? '>'
- [28a] DeclSep ::= PEReference | S
- [29] markupdecl ::= elementdecl | AttlistDecl | EntityDecl | NotatoinDecl | PI | Comment
- [30] extSubset ::= TextDecl? ExtSubsetDecl /*External subset */
- [31] extSutsetDecl ::= (markupdecl | conditionalSect | DeclSep)*
- [32] SDDDecl ::= S 'standalone' Eq ((' "' ('yes' | 'no') " "') /*Standalone document declaration*/
(' "' ('yes' | 'no') ' "')
- [33] LanguageID ::= Langcode (' - ' Subcode)* /*Language Identification*/
- [34] Langcode ::= ISO639Code | lanaCode | Usercode

```

[35] ISO639Code ::= ([a-z] | [A-Z]) ([a-z] | [A-Z])
[36] IanaCode ::= ('i' | 'l') '-' ([a-z] | [A-Z])+
[37] UserCode ::= ('x' | 'X') '-' ([a-z] | [A-Z])+
[38] Subcode ::= ([a-z] | [A-Z])+
[39] element ::= EmptyElemTag | STag content Etag           /*Element           */
[40] STag ::= '<' Name (S Attribute)* S? '>'              /*Start tag         */
[41] Attribute ::= Name Eq AttValue
[42] ETag ::= '</' Name S? '>'                          /*End tag           */
[43] content ::= CharData? ((element | Reference | CDsect | PI | /*Content of elements */
    Comment) CharData?)*
[44] EmptyElemTag ::= '<' Name (S Attribute)* S? '/>'     /*Empty element     */
[45] elementdecl ::= '<!Element' S Name S contentspec S? '>' /*Element type declaration*/
[46] contentspec ::= 'EMPTY' | 'ANY' | Mixed | children
[47] children ::= (choice | seq)('? ' | '*' | '+')?        /*Element content models*/
[48] cp ::= (Name | choice | seq)('? ' | '*' | '+')?
[49] choice ::= '(' S? cp (S? ' | ' S? cp)+ S? ')'
[50] seq ::= '(' S? cp (S? ' | ' S? cp)* S? ')'
[51] Mixed ::= '(' S? '#PCDATA' (S? ' | ' S? Name)* S? ')*' /*Mixed content declaration*/
    | '(' S? '#PCDATA' S? ')'
[52] AttlistDecl ::= '<!ATTLIST' S Name AttDef* S? '>'     /*Attribute list declaration*/
[53] AttDef ::= S Name S AttType S DefaultDecl
[54] AttType ::= StringType | TokenizedType | EnumeratedType /*Attribute types   */
[55] StringType ::= 'CDATA'
[56] TokenizedType ::= 'ID' | 'IDREF' | 'IDREFS' | 'ENTITY' |
    'ENTITIES' | 'NMTOKEN' | 'NMTOKENS'
[57] EnumeratedType ::= NotationType | Enumeration         /*Enumerated attribute types*/
[58] NotationType ::= 'NOTATION' S '(' S? Name (S? ' | ' S? Name)* S? ')'
[59] Enumeration ::= '(' S? Nmtoken (S? ' | ' S? Nmtoken)* S? ')'
[60] DefaultDecl ::= '#REQUIRED' | '#IMPLIED'             /*Attribute defaults */
    | (( '#FIXED' S)? AttValue)
[61] conditionalSect ::= includeSect | ignoreSect          /*Conditional section */
[62] includeSect ::= '<![ ' S? 'INCLUDE' S? '[' extSubsetDecl ']]>'
[63] ignoreSect ::= '<![ ' S? 'IGNORE' S? '[' ignoreSectContents* ']]>'
[64] ignoreSectContents ::= Ignore ('<![ ' ignoreSectContents ']]>' Ignore)*
[65] Ignore ::= Char* - (Char* ('<![ ' | ']]>') Char*)
[66] CharRef ::= '&#' [0-9]+ ';' | '&#x' [0-9a-fA-F]+ ';' /*Character reference */
[67] Reference ::= EntityRef | CharRef                   /*Entity reference    */
[68] EntityRef ::= '&' Name ';'
[69] PEReference ::= '%' Name ';'
[70] EntityDecl ::= GEDecl | PEDecl                       /*Entity declaration  */
[71] GEDecl ::= '<!ENTITY' S Name S EntityDef S? '>'
[72] PEDecl ::= '<!ENTITY' S '%' S Name S PEDef S? '>'
[73] EntityDef ::= EntityValue | (ExternalID NdataDecl?)
[74] PEDef ::= EntityValue | ExternalID
[75] ExternalID ::= 'SYSTEM' S SystemLiteral              /*External entity declaration*/
    | 'PUBLIC' S PuidLiteral S SystemLiteral
[76] NdataDecl ::= S 'NDATA' S Name

```

```

[77] TextDecl ::= '<?xml' VersionInfo? EncodingDecl S? '?>' /*Text declaration */
[78] extParsedEnt ::= TextDecl? content /*Well-formed external parsed entity*/
[79] extPE ::= TextDecl? extSubsetDecl
[80] EncodingDecl ::= S 'encoding' Eq ( ' ' ' ' EncName ' ' ' ' | ' ' ' ' /*Encoding declaration */
      EncName ' ' ' ')
[81] EncName ::= [A-Za-z] ([A-Za-z0-9._] | ' - ')*
[82] NotationDecl ::= '<!NOTATION' S Name S /*Notation declarations */
      (ExternalID | PublicID) S? '>'
[83] PublicID ::= 'PUBLIC' S PubidLiteral
[84] Letter ::= BaseChar | Ideographic /*Characters */
[85] BaseChar ::= [#x0041-#x005A] | [#x0061-#x007A] | [#x00C0-#x00D6] | [#x00D8-#x00F6] | [#x00F8-#x00FF] | [#x0100-#x0131] | [#x0134-#x013E] | [#x0141-#x0148] | [#x014A-#x017E] | [#x0180-#x01C3] | [#x01CD-#x01F0] | [#x01F4-#x01F5] | [#x01FA-#x0217] | [#x0250-#x02A8] | [#x02BB-#x02C1] | #x0386 | [#x0388-#x038A] | #x038C | [#x038E-#x03A1] | [#x03A3-#x03CE] | [#x03D0-#x03D6] | #x03DA | #x03DC | #x03DE | #x03E0 | [#x03E2-#x03F3] | [#x0401-#x040C] | [#x040E-#x044F] | [#x0451-#x045C] | [#x045E-#x0481] | [#x0490-#x04C4] | [#x04C7-#x04C8] | [#x04CB-#x04CC] | [#x04D0-#x04EB] | [#x04EE-#x04F5] | [#x04F8-#x04F9] | [#x0531-#x0556] | #x0559 | [#x0561-#x0586] | [#x05D0-#x05EA] | [#x05F0-#x05F2] | [#x0621-#x063A] | [#x0641-#x064A] | [#x0671-#x06B7] | [#x06BA-#x06BE] | [#x06C0-#x06CE] | [#x06D0-#x06D3] | #x06D5 | [#x06E5-#x06E6] | [#x0905-#x0939] | #x093D | [#x0958-#x0961] | [#x0985-#x098C] | [#x098F-#x0990] | [#x0993-#x09A8] | [#x09AA-#x09B0] | #x09B2 | [#x09B6-#x09B9] | [#x09DC-#x09DD] | [#x09DF-#x09E1] | [#x09F0-#x09F1] | [#x0A05-#x0A0A] | [#x0A0F-#x0A10] | [#x0A13-#x0A28] | [#x0A2A-#x0A30] | [#x0A32-#x0A33] | [#x0A35-#x0A36] | [#x0A38-#x0A39] | [#x0A59-#x0A5C] | #x0A5E | [#x0A72-#x0A74] | [#x0A85-#x0A8B] | #x0A8D | [#x0A8F-#x0A91] | [#x0A93-#x0AA8] | [#x0AAA-#x0AB0] | [#x0AB2-#x0AB3] | [#x0AB5-#x0AB9] | #x0ABD | #x0AE0 | [#x0B05-#x0B0C] | [#x0B0F-#x0B10] | [#x0B13-#x0B28] | [#x0B2A-#x0B30] | [#x0B32-#x0B33] | [#x0B36-#x0B39] | #x0B3D | [#x0B5C-#x0B5D] | [#x0B5F-#x0B61] | [#x0B85-#x0B8A] | [#x0B8E-#x0B90] | [#x0B92-#x0B95] | [#x0B99-#x0B9A] | #x0B9C | [#x0B9E-#x0B9F] | [#x0BA3-#x0BA4] | [#x0BA8-#x0BAA] | [#x0BAE-#x0BB5] | [#x0BB7-#x0BB9] | [#x0C05-#x0C0C] | [#x0C0E-#x0C10] | [#x0C12-#x0C28] | [#x0C2A-#x0C33] | [#x0C35-#x0C39] | [#x0C60-#x0C61] | [#x0C85-#x0C8C] | [#x0C8E-#x0C90] | [#x0C92-#x0CA8] | [#x0CAA-#x0CB3] | [#x0CB5-#x0CB9] | #x0CDE | [#x0CE0-#x0CE1] | [#x0D05-#x0D0C] | [#x0D0E-#x0D10] | [#x0D12-#x0D28] | [#x0D2A-#x0D39] | [#x0D60-#x0D61] | [#x0E01-#x0E2E] | #x0E30 | [#x0E32-#x0E33] | [#x0E40-#x0E45] | [#x0E81-#x0E82] | #x0E84 | [#x0E87-#x0E88] | #x0E8A | #x0E8D | [#x0E94-#x0E97] | [#x0E99-#x0E9F] | [#x0EA1-#x0EA3] | #x0EA5 | #x0EA7 | [#x0EAA-#x0EAB] | [#x0EAD-#x0EAE] | #x0EB0 | [#x0EB2-#x0EB3] | #x0EBD | [#x0EC0-#x0EC4] | [#x0F40-#x0F47] | [#x0F49-#x0F69] | [#x10A0-#x10C5] | [#x10D0-#x10F6] | #x1100 | [#x1102-#x1103] | [#x1105-#x1107] | #x1109 | [#x110B-#x110C] | [#x110E-#x1112] | #x113C | #x113E | #x1140 | #x114C | #x114E | #x1150 | [#x1154-#x1155] | #x1159 | [#x115F-#x1161] | #x1163 | #x1165 | #x1167 | #x1169 | [#x116D-#x116E] | [#x1172-#x1173] | #x1175 | #x119E | #x11A8 | #x11AB | [#x11AE-#x11AF] | [#x11B7-#x11B8] | #x11BA | [#x11BC-#x11C2] | #x11EB | #x11F0 | #x11F9 | [#x1E00-#x1E9B] | [#x1EA0-#x1EF9] | [#x1F00-#x1F15] | [#x1F18-#x1F1D] | [#x1F20-#x1F45] | [#x1F48-#x1F4D] | [#x1F50-#x1F57] | #x1F59 | #x1F5B | #x1F5D | [#x1F5F-#x1F7D] | [#x1F80-#x1FB4] | [#x1FB6-#x1FBC] | #x1FBE | [#x1FC2-#x1FC4] | [#x1FC6-#x1FCC] | [#x1FD0-#x1FD3] | [#x1FD6-#x1FDB] | [#x1FE0-#x1FEC] | [#x1FF2-#x1FF4] | [#x1FF6-#x1FFC] | #x2126 | [#x212A-#x212B] | #x212E | [#x2180-#x2182] | [#x3041-#x3094] | [#x30A1-#x30FA] | [#x3105-#x312C] | [#xAC00-#xD7A3]
[86] Ideographic ::= [#x4E00-#x9FA5] | #x3007 | [#x3021-#x3029]
[87] CombiningChar ::= [#x0300-#x0345] | [#x0360-#x0361] | [#x0483-#x0486] | [#x0591-#x05A1] | [#x05A3-#x05B9] | [#x05BB-#x05BD] | #x05BF | [#x05C1-#x05C2] | #x05C4 | [#x064B-#x0652] | #x0670 | [#x06D6-#x06DC] | [#x06DD-#x06DF] | [#x06E0-#x06E4] | [#x06E7-#x06E8] | [#x06EA-#x06ED]

```

[#x0901-#x0903] | #x093C | [#x093E-#x094C] | #x094D | [#x0951-#x0954] | [#x0962-#x0963] | [#x0981-#x0983] | #x09BC | #x09BE | #x09BF | [#x09C0-#x09C4] | [#x09C7-#x09C8] | [#x09CB-#x09CD] | #x09D7 | [#x09E2-#x09E3] | #x0A02 | #x0A3C | #x0A3E | #x0A3F | [#x0A40-#x0A42] | [#x0A47-#x0A48] | [#x0A4B-#x0A4D] | [#x0A70-#x0A71] | [#x0A81-#x0A83] | #x0ABC | [#x0ABE-#x0AC5] | [#x0AC7-#x0AC9] | [#x0ACB-#x0ACD] | [#x0B01-#x0B03] | #x0B3C | [#x0B3E-#x0B43] | [#x0B47-#x0B48] | [#x0B4B-#x0B4D] | [#x0B56-#x0B57] | [#x0B82-#x0B83] | [#x0BBE-#x0BC2] | [#x0BC6-#x0BC8] | [#x0BCA-#x0BCD] | #x0BD7 | [#x0C01-#x0C03] | [#x0C3E-#x0C44] | [#x0C46-#x0C48] | [#x0C4A-#x0C4D] | [#x0C55-#x0C56] | [#x0C82-#x0C83] | [#x0CBE-#x0CC4] | [#x0CC6-#x0CC8] | [#x0CCA-#x0CCD] | [#x0CD5-#x0CD6] | [#x0D02-#x0D03] | [#x0D3E-#x0D43] | [#x0D46-#x0D48] | [#x0D4A-#x0D4D] | #x0D57 | #x0E31 | [#x0E34-#x0E3A] | [#x0E47-#x0E4E] | #x0EB1 | [#x0EB4-#x0EB9] | [#x0EBB-#x0EBC] | [#x0EC8-#x0ECD] | [#x0F18-#x0F19] | #x0F35 | #x0F37 | #x0F39 | #x0F3E | #x0F3F | [#x0F71-#x0F84] | [#x0F86-#x0F8B] | [#x0F90-#x0F95] | #x0F97 | [#x0F99-#x0FAD] | [#x0FB1-#x0FB7] | #x0FB9 | [#x20D0-#x20DC] | #x20E1 | [#x302A-#x302F] | #x3099 | #x309A
[88] Digit ::= [#x0030-#x0039] | [#x0660-#x0669] | [#x06F0-#x06F9] | [#x0966-#x096F] | [#x09E6-#x09EF] | [#x0A66-#x0A6F] | [#x0AE6-#x0AEF] | [#x0B66-#x0B6F] | [#x0BE7-#x0BEF] | [#x0C66-#x0C6F] | [#x0CE6-#x0CEF] | [#x0D66-#x0D6F] | [#x0E50-#x0E59] | [#x0ED0-#x0ED9] | [#x0F20-#x0F29]
[89] Extender ::= #x00B7 | #x02D0 | #x02D1 | #x0387 | #x0640 | #x0E46 | #x0EC6 | #x3005 | [#x3031-#x3035] | [#x309D-#x309E] | [#x30FC-#x30FE]

APPENDIX II – XML NAMESPACE SYNTAX

- [1] NSAttName ::= PrefixedAttName | DefaultAttName
- [2] PrefixedAttName ::= 'xmlns:' NCName
- [3] DefaultAttName ::= 'xmlns'
- [4] NCName ::= (Letter | '_') (NCNameChar)*
- [5] NCNameChar ::= Letter | Digit | '.' | '-' | CombiningChar | Extender
- [6] QName ::= (Prefix ':')? LocalPart
- [7] Prefix ::= NCName
- [8] LocalPart ::= NCName
- [9] Stag ::= '<' QName (S Attribute)* S? '>' [NSC:Prefix Declared]
- [10] Etag ::= '</' QName S? '>' [NSC:Prefix Declared]
- [11] EmptyElemTag ::= '<' QName (S Attribute)* S? '/>' [NSC:Prefix Declared]
- [12] Attribute ::= NSAttName Eq AttValue | QName Eq AttValue [NSC:Prefix Declared]
- [13] doctypeddecl ::= '<!DOCTYPE' S QName (S ExternalID)? S? ('['
 (markupdecl | PEReference | S)* ']' S?)? '>'
- [14] elementdecl ::= '<!Element' S QName S contentspec S> '>'
- [15] cp ::= (QName | choice | seq) ('?' | '*' | '+')?
- [16] Mixed ::= '(' S? '#PCDATA' (S? '|' S? QName)* S? ')' * | '(' S? '#PCDATA' S? ')' *
- [17] AttlistDecl ::= '<!ATTLIST' S QName AttDef* S? '>'
- [18] AttDef ::= S (QName | NSAttName) S AttType S DefaultDecl

APPENDIX III – XLINK SYNTAX

- [1] Locator ::= URI | Connector (XPointer | Name) | URI Connector (XPointer | Name)
- [2] Connector ::= '#' | '|'
- [3] URI ::= URIchar*
- [4] Query ::= 'XML-XPTR=' (XPointer | Name)

APPENDIX IV – XPATH SYNTAX

- [1] LocationPath ::= RelativeLocationPath | AbsoluteLocationPath
- [2] AbsoluteLocationPath ::= '/' RelativeLocationPath? | AbbreviatedAbsoluteLocationPath
- [3] RelativeLocationPath ::= Step | RelativeLocationPath '/' Step | AbbreviatedRelativeLocationPath
- [4] Step ::= AxisSpecifier NodeTest Predicate* | AbreviatedStep
- [5] AxisSpecifier ::= AxisName '::' | AbbreviatedAxisSpecifier
- [6] AxisName ::= 'ancestor' | ancestor-or-self' | 'attribute' | 'child' | 'descendant'
| 'descendant-or-self' | 'following' | 'following-sibling' | 'namespace'
| 'parent' | 'preceding' | 'preceding-sibling' | 'self'
- [7] NodeTest ::= NameTest | NodeType '(' ')' | 'processing-instruction' '(' Literal ')'
- [8] Predicate ::= '[' PredicateExpr ']'
- [9] PredicateExpr ::= Expr
- [10] AbbreviatedAbsoluteLocationPath ::= '// RelativeLocationPath
- [11] AbbreviatedRelativeLocationPath ::= RelativeLocationPath '// Step
- [12] AbbreviatedStep ::= '.' | '..'
- [13] AbbreviatedAxisSpecifier ::= '@'?
- [14] Expr ::= OrExpr
- [15] PrimaryExpr ::= VariableReference | '(' Expr ') | Literal | Number | FunctionCall
- [16] FunctionCall ::= FunctionName '(' (Argument (',' Argument)*)? ')'
- [17] Argument ::= Expr
- [18] UnionExpr ::= PathExpr | UnionExpr '|' PathExpr
- [19] PathExpr ::= LocationPath | FilterExpr | FilterExpr '/'
- [20] FilterExpr ::= PrimaryExpr | FilterExpr Predicate
- [21] OrExpr ::= AndExpr | OrExpr 'or' AndExpr
- [22] AndExpr ::= EqualityExpr | AndExpr 'and' EqualityExpr
- [23] EqualityExpr ::= RelationalExpr | EqualityExpr '=' RelationalExpr |
EqualityExpr '!=' RelationalExpr
- [24] RelationalExpr ::= AdditiveExpr | RelationalExpr '<' AdditiveExpr |
RelationalExpr '>' AdditiveExpr | RelationalExpr '<=' AdditiveExpr |
RelationalExpr '>=' AdditiveExpr
- [25] AdditiveExpr ::= MultiplicativeExpr | AdditiveExpr '+' MultiplicativeExpr |
AdditiveExpr '-' MultiplicativeExpr
- [26] MultiplicativeExpr ::= UnaryExpr | MultiplicativeExpr MultiplyOperator UnaryExpr
| MultiplicativeExpr 'div' UnaryExpr | MultiplicativeExpr 'mod' UnaryExpr
- [27] UnaryExpr ::= UnionExpr | '-' UnaryExpr
- [28] ExprToken ::= '(' | ')' | '[' | ']' | '.' | '..' | '@' | '/' | '::' | NameTest | NodeType | Operator
| FunctionName | AxisName | Literal | Number | VariableReference
- [29] Literal ::= ''' [^] * ''' | ''' [^] * '''
- [30] Number ::= Digits ('.' Digits) ? | '.' Digits
- [31] Digits ::= [0-9]+

- [32] Operator ::= OperatorName | MultiplyOperator | '/' | '/' | '|' | '+' | '-' | '=' | '!=' | '<' | '<=' | '>' | '>='
- [33] OperatorName ::= 'and' | 'or' | 'mod' | 'div'
- [34] MultiplyOperator ::= '*'
- [35] FunctionName ::= QName – NodeType
- [36] VariableReference ::= '\$' QName
- [37] NameTest ::= '*' NCName ':' '*' | QName
- [38] NodeType ::= 'comment' | 'text' | 'processing-instruction' | 'node'
- [39] ExprWhitespace ::= S

APENDIX V – XPOINTER SYNTAX

- [1] XPointer ::= AbsTerm '.' OtherTerms | AbsTerm | OtherTerms
- [2] OtherTerms ::= OtherTerm | OtherTerm '.' OtherTerm
- [3] OtherTerm ::= RelTerm | SpanTerm | AttrTerm | StringTerm
- [4] AbsTerm ::= 'root()' | 'origin()' | IdLoc | HTMLAddr
- [5] IdLoc ::= 'id(' Name ')'
- [6] HTMLAddr ::= 'html(' SkipLit ')'
- [7] RelTerm ::= Keyword? Arguments
- [8] Keyword ::= 'child' | 'descendant' | 'ancestor' | 'preceding' | 'following' | 'psibling' | 'fsibling'
- [9] Arguments ::= '(' InstanceOrAll (';' NodeType (';' Attr ';' Val)*)? ')'
- [10] InstanceOrAll ::= 'all' | Instance
- [11] Instance ::= ('+' | '-')? [1-9] Digit*
- [12] NodeType ::= Name | '#element' | '#pi' | '#comment' | '#text' | '#cdata' | '#all'
- [13] Attr ::= '*' | Name <!-- any attribute type -->
- [14] Value ::= '#IMPLIED' <!-- no value specified, no default -->
 - | '*' <!-- any value, even defaulted -->
 - | Name
 - | SkipLit <!-- exact match -->

APPENDIX VI – EXAMPLES OF PML SCHEMA

1. Coordinate:

```
<?xml version="1.0"?>
<xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.pitt.edu"
  xmlns:pml="http://www.pitt.edu"
  elementFormDefault="qualified"
  version="1.0">
  <xsd:annotation>
    <xsd:documentation>
      "coordinate.xsd"
      Specify the coordinate attributes.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:attribute name="x" type="xsd:double"/>
  <xsd:attribute name="y" type="xsd:double"/>
  <xsd:attribute name="z" type="xsd:double"/>
</xsd:schema>
```

2. Geometric Point:

```
<?xml version="1.0"?>
<xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.pitt.edu"
  xmlns:pml="http://www.pitt.edu"
  elementFormDefault="qualified"
  version="1.0">
  <xsd:annotation>
    <xsd:documentation>
      "point.xsd"
      Define geometric entity - POINT.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:include schemaLocation="coordinate.xsd"/>
  <xsd:element name="POINT">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:restriction base="xsd:string">
          <xsd:attribute ref="x" use="required"/>
          <xsd:attribute ref="y" use="required"/>
          <xsd:attribute ref="z" use="required"/>
          <xsd:attribute name="id" type="xsd:string"/>
        </xsd:restriction>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

3. Vector:

```
<?xml version="1.0"?>
<xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.pitt.edu"
  xmlns:pml="http://www.pitt.edu"
  elementFormDefault="qualified"
  version="1.0">
  <xsd:annotation>
    <xsd:documentation>
      "vector.xsd"
      Define geometric entity - VECTOR.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:include schemaLocation="coordinate.xsd"/>
  <xsd:element name="VECTOR">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:restriction base="xsd:string">
          <xsd:attribute ref="x" use="required"/>
          <xsd:attribute ref="y" use="required"/>
          <xsd:attribute ref="z" use="required"/>
          <xsd:attribute name="id" type="xsd:string"/>
        </xsd:restriction>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

4. Point Reference

```
<?xml version="1.0"?>
<xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.pitt.edu"
  xmlns:pml="http://www.pitt.edu"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  elementFormDefault="qualified"
  version="1.0">
  <xsd:annotation>
    <xsd:documentation>
      "refPoint.xsd"
      Define the reference to geometric entity - POINT.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="refPOINT">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:restriction base="xsd:string">
          <xsd:attribute name="xlink:type" use="required">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string">
                <xsd:enumeration value="simple"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:attribute>
          <xsd:attribute name="xlink:href" type="xsd:string" use="required">
            <xsd:attribute name="xlink:actuate" use="required">
              <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="onLoad"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="xlink:show" use="required">
              <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="replace"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:attribute>
          </xsd:restriction>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
```

5. Vector Reference:

```
<?xml version="1.0"?>
<xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.pitt.edu"
  xmlns:pml="http://www.pitt.edu"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  elementFormDefault="qualified"
  version="1.0">
  <xsd:annotation>
    <xsd:documentation>
      "refVector.xsd"
      Define the reference to geometric entity - VECTOR.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="refVECTOR">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:restriction base="xsd:string">
          <xsd:attribute name="xlink:type" use="required">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string">
                <xsd:enumeration value="simple"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:attribute>
          <xsd:attribute name="xlink:href" type="xsd:string" use="required">
            <xsd:attribute name="xlink:actuate" use="required">
              <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="onLoad"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="xlink:show" use="required">
              <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="replace"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:attribute>
          </xsd:restriction>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
```


6. Line:

```
<?xml version="1.0"?>
<xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.pitt.edu"
  xmlns:pml="http://www.pitt.edu"
  elementFormDefault="qualified"
  version="1.0">
  <xsd:annotation>
    <xsd:documentation>
      "line.xsd"
      Define geometric entity - LINE.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:include schemaLocation="refPoint.xsd"/>
  <xsd:include schemaLocation="refVector.xsd"/>
  <xsd:element name="LINE">
    <xsd:complexType>
      <xsd:choice>
        <xsd:sequence>
          <xsd:element ref="refPOINT"/>
          <xsd:element ref="refPOINT"/>
        </xsd:sequence>
        <xsd:sequence>
          <xsd:element ref="refPOINT"/>
          <xsd:element ref="refVECTOR"/>
        </xsd:sequence>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

7. Line Reference:

```
<?xml version="1.0"?>
<xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.pitt.edu"
  xmlns:pml="http://www.pitt.edu"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  elementFormDefault="qualified"
  version="1.0">
  <xsd:annotation>
    <xsd:documentation>
      "refLine.xsd"
      Define the reference to geometric entity - LINE.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="refLINE">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:restriction base="xsd:string">
          <xsd:attribute name="xlink:type" use="required">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string">
                <xsd:enumeration value="simple"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:attribute>
          <xsd:attribute name="xlink:href" type="xsd:string" use="required">
            <xsd:attribute name="xlink:actuate" use="required">
              <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="onLoad"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="xlink:show" use="required">
              <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="replace"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:attribute>
          </xsd:restriction>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
```

8. Plane:

```
<?xml version="1.0"?>
<xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.pitt.edu"
  xmlns:pml="http://www.pitt.edu"
  elementFormDefault="qualified"
  version="1.0">
  <xsd:annotation>
    <xsd:documentation>
      "plane.xsd"
      Define geometric entity - PLANE.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:include schemaLocation="refPoint.xsd"/>
  <xsd:include schemaLocation="refVector.xsd"/>
  <xsd:include schemaLocation="refLine.xsd"/>
  <xsd:element name="PLANE">
    <xsd:complexType>
      <xsd:choice>
        <xsd:sequence>
          <xsd:element ref="refPOINT"/>
          <xsd:element ref="refPOINT"/>
          <xsd:element ref="refPOINT"/>
        </xsd:sequence>
        <xsd:sequence>
          <xsd:element ref="refPOINT"/>
          <xsd:element ref="refVECTOR"/>
          <xsd:element ref="refVECTOR"/>
        </xsd:sequence>
        <xsd:sequence>
          <xsd:element ref="refLINE"/>
          <xsd:element ref="refLINE"/>
        </xsd:sequence>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

9. Distance Constraint:

```
<?xml version="1.0"?>
<xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.pitt.edu"
  xmlns:pml="http://www.pitt.edu"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  elementFormDefault="qualified"
  version="1.0">
  <xsd:annotation>
    <xsd:documentation>
      "conDistance.xsd"
      Define the constraint of distance.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="conDISTANCE">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="loc1">
          <xsd:complexType>
            <xsd:simpleContent>
              <xsd:restriction base="xsd:string"/>
            <xsd:simpleContent>
              <xsd:attributeGroup ref="locatorAttributes"/>
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="loc2">
            <xsd:complexType>
              <xsd:simpleContent>
                <xsd:restriction base="xsd:string"/>
              <xsd:simpleContent>
                <xsd:attributeGroup ref="locatorAttributes"/>
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="arc1">
              <xsd:complexType>
                <xsd:simpleContent>
                  <xsd:restriction base="xsd:string"/>
                <xsd:simpleContent>
                  <xsd:attributeGroup ref="arcAttributes"/>
                </xsd:complexType>
              </xsd:element>
              <xsd:element name="arc2">
                <xsd:complexType>
                  <xsd:simpleContent>
                    <xsd:restriction base="xsd:string"/>
                  <xsd:simpleContent>
                    <xsd:attributeGroup ref="locatorAttributes"/>
                  </xsd:complexType>
                </xsd:element>
              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:element>
</xsd:schema>
```

```

</xsd:complexType>
<xsd:simpleContent>
  <xsd:restriction base="xsd:double"/>
  <xsd:attribute name="xlink:type" use="required" >
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="extended"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:restriction>
<xsd:simpleContent>
</xsd:element>

<xsd:attributeGroup name="locatorAttributes">
  <xsd:attribute name="xlink:type" use="required" >
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="locator"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="xlink:label" type="xsd:string" use="required"/>
  <xsd:attribute name="xlink:href" type="xsd:string" use="required"/>
</xsd:attributeGroup>

<xsd:attributeGroup name="arcAttributes" >
  <xsd:attribute name="xlink:type" use="required" >
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="arc"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="xlink:from" type="xsd:string" use="required"/>
  <xsd:attribute name="xlink:to" type="xsd:string" use="required"/>
  <xsd:attribute name="xlink:actuate" use="required">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="onLoad"/>
        <xsd:enumeration value="onRequest"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:attributeGroup>

</xsd:schema>

```

BIBLIOGRAPHY

- [1] Zott, C., Amit, R., and Donlevy, J., “Strategies for value creation in e-commerce: best practice in Europe”, *European Management Journal*, Vol. 18, No. 5 (October, 2000), pp. 463-475.
- [2] McMeekin, R., “Manufacturing and the internet – we haven’t seen anything yet!”, *Computers & Chemical Engineering*, Vol. 24, No. 2-7 (July, 2000), pp. 161.
- [3] Sutherland, Ivan, “Sketchpad: A Man-machine Graphical Communications System”, Ph.D. Dissertation, Massachusetts Institute of Technology, 1963.
- [4] Kao, Y.C. and Lin, G.C.I., “CAD/CAM collaboration and remote machining”, *Computer Integrated Manufacturing Systems*, Vol. 9, No. 3 (July, 1996), pp. 149-160.
- [5] Parunak, H.V.D., “Distributed Collaborative Design (DisCollab): An ATP Opportunity” <http://www.mel.nist.gov/msid/groups/edt/ATP/white-paper> (Whitepaper of NIST-ATP Workshop: Tools and Technologies for Distributed and Collaborative Design, August, 1997).
- [6] *ibid.*
- [7] Blessing, L. and Wallace, K., “Supporting the Knowledge Life-Cycle”, in: Finger, S., Tomiyama, T., and Mäntylä, ed., *Knowledge Intensive Computer Aided Design, IFIP TC5 WG5.2 Third Workshop on Knowledge Intensive CAD, December 1-4, 1998, Tokyo, Japan*, (Boston, Massachusetts: Kluwer Academic Publishers, 1998), pp.21-38
- [8] Wang, Y. and Nnaji, B.O., “Functionality-Based Modular Design for Mechanical Product Customization Over the Internet”, *Journal of Design and Manufacturing Automation*, Vol.1, No.1-2, 2001, pp.107-121
- [9] WWW Consortium, <http://www.w3.org/XML/>

- [10] Horváth, L., Rudas, I.J., and Varga, T., “Some Possibilities for Including Knowledge into Models of Mechanical Parts”, *IEEE Proceedings of the 1997 International Conference on Intelligent Engineering Systems*, 1997, pp.527-532
- [11] Levesque, H.J. and Brachman, R.J., “A Fundamental Tradeoff in Knowledge Representation and Reasoning”, in: Brachman, R.J. and Levesque, H.J., ed., *Readings in Knowledge Representation* (Los Altos, California: Morgan Kaufmann, 1985), Chapter 4, pp.41-70
- [12] Sluder, R., “PDES/STEP: The Cornerstone of CALS”, *IEEE Proceedings of the 1990 and 1991 Workshop on Reliability and Maintainability Computer-Aided Engineering in Concurrent Engineering*, Leesburg, 1992, pp.227-230
- [13] Pratt, M.J., “Extension of STEP for the Representation of Parametric and Variational Models”, in: Roller, D. and Brunet, P., ed., *CAD Systems Development: Tools and Methods* (Berlin: Springer-Verlag, 1997), pp.237-250
- [14] Elmasri, R. and Navathe, S.B., *Fundamentals of Database Systems* (2nd edition; Menlo Park, California: Addison-Wesley Publishing Company, 1994), pp.39-68
- [15] *ibid.*, pp.611-661
- [16] *IDEF1X Manual USAF Integrated Computer-Aided Manufacturing Program* (D Appleton Company, USA, 1986)
- [17] Kusiak, A., Letsche, T. and Zakarian, A., “Data Modelling with IDEF1X”, *International Journal of Computer Integrated Manufacturing*, Vol.10, No.6 (November/December, 1997), pp.470-486
- [18] Verheijen, G.M.A. and Van Bekkum, J., “NIAM: an Information Analysis Method”, in: Olle, T.W., Sol, H.G. and Verijin-Stuart, A.A., ed., *Information Systems Design Methodologies: A Comparative Review: Proceedings of the IFIP WG8.1 Working Conference on Cooperative Review of Information Systems Design Methodologies, 10-14 May, 1982, Noordwijkerhout, Netherlands* (Amsterdam: North-Holland Publishing Company, 1982), pp.537-589
- [19] Eastman, C.M. and Fereshetian, N., “Information Models for Use in Product Design: A Comparison”, *Computer-Aided Design*, Vol.26, NO.7 (July, 1994), pp.551-572

- [20] Brown, D.C. and Birmingham, W.P., “Understanding the Nature of Design”, *IEEE Expert*, Vol.12, No.2 (March-April, 1997), pp.14-16
- [21] Jackson, Peter, *Introduction to Expert Systems* (2nd Edition; Workingham, England: Addison-Wesley, 1990), pp. 72
- [22] Brachman, R.J. and Levesque, H.J., ed., *Readings in Knowledge Representation* (Los Altos, California: Morgan Kaufmann Publishers, 1985), pp.xiii.
- [23] Smith, B.C., “Reflection and Semantics in a Procedural Language”, Ph.D. Dissertation, Massachusetts Institute of Technology, 1982, pp.2
- [24] Reichgelt, H., *Knowledge Representation: An AI Perspective* (Norwood, New Jersey: Ablex Publishing Co., 1991)
- [25] Davis, R., Buchanan, B., and Shortliffe, E., “Production Rules as a Representation for a Knowledge-Based Consultation Program”, *Artificial Intelligence*, Vol.8, No.1 (1977), pp.15-45
- [26] Quillian, R., “Semantic Memory”, Ph.D. dissertation, Carnegie Institute of Technology, 1966
- [27] Brachman, R.J., “On the Epistemological Status of Semantic Networks”, in: Findler, N.V., ed., *Associative Networks: Representation and Use of Knowledge by Computers* (New York: Academic Press, 1979), pp.3-50
- [28] Hartley, R.T. and Barnden, J.A., “Semantic networks: visualizations of knowledge”, *Trends in Cognitive Sciences*, Vol.1, No.5 (August 1997), pp.169-175
- [29] Minsky, M., “A Framework for Representing Knowledge”, in: Haugeland, J., ed., *Mind Design* (Cambridge, Massachusetts: The MIT Press, 1981), pp.95-128
- [30] Bobrow, D.G. and Winograd, T., “An Overview of KRL, a Knowledge Representation Language”, *Cognitive Science*, Vol.1, No.1 (1977), pp.3-46

- [31] Brachman, R.J. and Schmolze, J.G., "An overview of the KL-ONE knowledge representation system", *Cognitive Science*, Vol.9, No.2 (April-June, 1985), pp.171-216
- [32] Woods, W.A. and Schmolze, J.G., "The KL-ONE Family", *Computers and Mathematics with Applications*, Vol.23, No.2-9 (1992), pp.1-50
- [33] Woods, W.A. "What's in a Link: Foundations for Semantic Networks", in: Bobrow, D.G. and Collins, A.M., ed., *Representation and Understanding: Studies in Cognitive Science* (New York: Academic Press, 1975), pp.35-82
- [34] Brachman, R.J., Fikes, R.E., and Levesque, H.J., "KRYPTON: A Functional Approach to Knowledge Representation", *IEEE Computer*, Vol.16, No.10 (1983), pp.67-73
- [35] Brachman, R.J., McGuinness, D.L., Patel-Schneider, P.F., and Borgida, A., "Reducing CLASSIC to Practice: Knowledge Representation Theory Meets Reality", *Artificial Intelligence*, Vol.114, No.1-2 (October, 1999), pp.203-237
- [36] McGuinness, D.L. and Wright, J.R., "An Industrial-Strength Description Logic-Based Configurator Platform", *IEEE Intelligent System*, Vol.13, No.4 (July/August, 1998), pp.69-77
- [37] McGuinness, D.L. and Wright, J.R., "Conceptual Modelling for Configuration: A Description Logic-Based Approach", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM*, Vol.12, No.4 (September, 1998), pp.333-344
- [38] Wilensky, R., "Knowledge Representation - A Critique and A Proposal", in: Kolodner, J.L. and Riesbeck, C.K., ed., *Experience, Memory, and Reasoning* (Hillsdale, New Jersey: Lawrence Erlbaum Associates, 1986), Chapter 2, pp.15-28
- [39] Borgida, A., Greenspan, S. and Mylopoulos, J., "Knowledge Representation as the basis for Requirements Specification", *IEEE Computer*, Vol.18, No.4 (April, 1985), pp.82-91
- [40] Mylopoulos, J., Borgida, A., Jarke, M. and Koubarakis, M., "Telos: Representing Knowledge About Information Systems", *ACM Transactions on Information Systems*, Vol.8, No.4 (October, 1990), pp.325-362

- [41] Greenspan, S., Mylopoulos, J., and Borgida, A., "On Formal Requirements Modeling Language: RML Revisited", *Proceedings of the 16th International Conference on Software Engineering, 16-21 May, 1994* (Los Alamitos, California: IEEE Computer Society Press, 1994), pp.135-147
- [42] Genesereth, M. R. And Fikes, R. E., *Knowledge Interchange Format, version 3.0, Reference Manual*, Logic-92-1 (Stanford, California: Computer Science Department, Stanford University, 1992)
- [43] Gruber, T. R., "A Translation Approach to Portable Ontology Specifications", *Knowledge Acquisition*, Vol.5, No.2 (1993), pp.199-220
- [44] Finin, T., Fritzson, R., McKay, D., and McEntire, R., "KQML as an agent communication language", *Proceedings of the Third International Conference on Information and Knowledge Management, November 29 - December 2, 1994, Gaithersburg, MD USA* (ACM Press, 1994), pp.456-463
- [45] Compatangelo, E. and Rumolo, G., "EDDL_{DP} + TDDL_{DP} = a Double-Level Approach to Domain Knowledge Modelling", in: Kangassalo, H. and Nilsson, J.F., ed., *Information Modelling and Knowledge Bases VIII* (Amsterdam: IOS Press, 1997), pp.279-295
- [46] Compatangelo, E. and Rumolo, G., "An Engineering Framework for Domain Knowledge Modelling", in: Charrel, P.J., Jaakkola, H., Kangassalo, H. and Kawaguchi, E., ed., *Information Modelling and Knowledge Bases IX* (Amsterdam: IOS Press, 1998), pp.51-65
- [47] Compatangelo, E., Donini, F.M. and Rumolo, G., "Engineering of KR-Based Support Systems for Conceptual Modelling & Analysis", in: Jaakkola, H., Kangassalo, H. and Kawaguchi, E., ed., *Information Modelling and Knowledge Bases X* (Amsterdam: IOS Press, 1999), pp.115-131
- [48] Coyne, R.D., Rosenman, M.A., Radford, A.D., Balachandran, M., and Gero, J.S., *Knowledge-Based Design Systems* (Reading, Massachusetts: Addison-Wesley Publishing Company, 1990)
- [49] Veth, B., "An Integrated Data Description Language for Coding Design Knowledge", in: ten Hagen, P.J.W. and Tomiyama, T., ed., *Intelleged CAD System I: Theoretical and Methodological Aspects* (Berlin: Springer-Verlag, 1987), pp. 295-313

- [50] Tomiyama, T., "Object Oriented Programming Paradigm for Intelligent CAD Systems", in: Akman, V., ten Hagen, P.J.W. and Veerkamp, P.J., ed., *Intelligent CAD System II: Implementational Issues* (Berlin: Springer-Verlag, 1989), pp. 3-16,
- [51] Veerkamp, P., Akman, V., Bernus, P., and ten Hagen, P.J.W., "IDDL: A Language for Intelligent Interactive Integrated CAD Systems", in: Akman, V., ten Hagen, P.J.W. and Veerkamp, P.J., ed., *Intelligent CAD System II: Implementational Issues* (Berlin: Springer-Verlag, 1989), pp. 58-74
- [52] Tomiyama, T., "Advances in Intelligent CAD Systems", in: Ören, T.I., ed., *Advances in Artificial Intelligence in Software Engineering, Vol.1* (Greenwich, Connecticut: JAI Press Inc., 1990), pp.251-283
- [53] Eastman, C.M., "A Data Model for Design Knowledge", in: Carrara, G. and Kalay, Y.E., ed., *Knowledge-Based Computer-Aided Architectural Design* (Amsterdam: Elsevier, 1994), pp.95-122
- [54] Salustri, F.A. "Ontological Commitments in Knowledge-Based Design Software: A Progress Report", in: Finger, S., Tomiyama, T., and Mantyla, M., ed., *Knowledge Intensive Computer Aided Design, IFIP TC5 WG5.2 Third Workshop on Knowledge Intensive CAD, December 1-4, 1998, Tokyo, Japan* (Boson: Kluwer Academic Publishers, 1998), pp.41-72
- [55] Ozawa, M., Cutkosky, M.R., and Howley, B.J., "Model Sharing among Agents in a Concurrent Product Development Team", in: Finger, S., Tomiyama, T., and Mantyla, M., ed., *Knowledge Intensive Computer Aided Design, IFIP TC5 WG5.2 Third Workshop on Knowledge Intensive CAD, December 1-4, 1998, Tokyo, Japan* (Boson: Kluwer Academic Publishers, 1998), pp.143-165
- [56] Spooner, D.L., "Towards an Object-Oriented Data Model for a Mechanical CAD Database System", in: Dittrich, K.R., Dayal, U., and Buchmann, A.P., ed., *On Object-Oriented Database Systems* (Berlin: Springer-Verlag, 1991), pp.190-205
- [57] Hoare, C.A.R., "Notes on Data Structuring", in: Dahl, O.J., Dijkstra, E.W. and Hoare, C.A.R., ed., *Structured Programming* (London: Academic Press, 1972), pp.83-174
- [58] Smith, J.M and Smith, D.C.P., "Database Abstractions: Aggregation", *Communications of ACM*, Vol.20, No.6 (June 1977), pp.405-413

- [59] Smith, J.M. and Smith, D.C.P., "Database Abstractions: Aggregation and Generalization", *ACM Transactions on Database Systems*, Vol.2, No.2 (June 1977), pp.105-133
- [60] MathML, <http://www.w3.org/Math/>
- [61] Chemical Markup Language, <http://www.xml-cml.org/>
- [62] Synchronized Multimedia Integration Language, <http://www.w3.org/AudioVideo/>
- [63] Scalable Vector Graphics, <http://www.w3.org/Graphics/SVG/Overview.htm8>
- [64] Electronic Business Extensible Markup Language, <http://www.ebxml.org/>
- [65] Open Financial Exchange, <http://www.ofx.net/>
- [66] Wireless Markup Language, <http://www.wapforum.org/>
- [67] OASIS XML Cover Pages, <http://www.oasis-open.org/cover/xml.html>
- [68] Ratchev, S.M., Shiau, J. and Valtchanov, G., "Distributed Product and Facility prototyping in Extended Manufacturing Enterprises", *International Journal of Production Research*, Vol.38, No.17 (2000), pp.4495-4506
- [69] Kahn, H., Filer, N., Williams, A. and Whitaker, N., "A Generic Framework for Transforming EXPRESS Information Models", *Computer-Aided Design*, Vol.33, No.7 (June 2001), pp.501-510
- [70] Burkett, W.C., "Product Data Markup Language: A New Paradigm for Product Data Exchange and Integration", *Computer-Aided Design*, Vol.33, No. 7 (June 2001), pp.489-500
- [71] Szykman, S., Senfaute, J., and Sriram, R.D., "The Use of XML for Describing Functions and Taxonomies in Computer-Based Design", *Proceedings of the 1999 ASME Design Engineering Technical Conferences, September 12-15, 1999, Las Vegas, Nevada* (New York: ASME), paper No. DETC99/CIE-9025

- [72] Szykman, S., Sriram, R.D., Bochenek, C., Racz, J.W., and Senfaute, J., "Design Repositories: Engineering Design's New Knowledge Base", *IEEE Intelligent Systems*, Vol.15, No. 3 (May-June 2000), pp.48-55
- [73] W3C XML 1.0 recommendation, <http://www.w3.org/TR/REC-xml/>
- [74] Goldfarb, Charles F., *The SGML Handbook* (Oxford: Oxford University Press, 1990), 4.183, pp.125
- [75] XML Schema work draft, <http://www.w3.org/TR/xmlschema-0/>, <http://www.w3.org/TR/xmlschema-1/>, <http://www.w3.org/TR/xmlschema-2/>,
- [76] XML Xlink, <http://www.w3.org/TR/xlink/>
- [77] Hillyard, R.C. and Braid, I.C., "Analysis of Dimensions and Tolerances in Computer-Aided Mechanical Design", *Computer-Aided Design*, Vol.10, No.3 (May 1978), pp.161-166
- [78] Light, R. and Gossard, D., "Modification of Geometric Models Through Variational Geometry", *Computer-Aided Design*, Vol. 14, No. 4 (July 1982), pp.209-214
- [79] Perez, A. and Serrano, D., "Constraint Based Analysis Tools for Design", *ACM Proceedings on the 2nd Symposium on Solid Modeling and Applications, 1993, Montreal, Quebec, Canada*, pp.281-291
- [80] Lamure, H. and Michelucci, D., "Solving Geometric Constraints by Homotopy", *Proceedings of the Third ACM Symposium on Solid Modeling and Applications, May 17-19, 1995, Salt Lake City, Utah*, pp.263-269
- [81] Ge, J-X, Chou, S-C, and Gao, X-S, "Geometric Constraint Satisfaction Using Optimization Methods", *Computer-Aided Design*, Vol.31, No.14 (December, 1999), pp.867-879
- [82] Mullineux, G, "Constraint Resolution Using Optimisation Techniques", *Computers & Graphics*, Vol.25, No.3 (June 2001), pp.483-492

- [83] Sunde, G., "Specification of Shape by Dimensions and Other Geometric Constraints", In: Wozny, M.J., McLaughlin, H.W., and Encarnacao, J.L., ed., *Geometric Modeling for CAD Applications, IFIP WG 5.2 Working Conference on Geometric Modeling for CAD Applications, May 12-16, 1986, Rensselaerville, New York*, (Amsterdam: North-Holland, 1988), pp.199-213
- [84] Yamaguchi, Y., and Kimura, F., "A Constraint Modeling System for Variational Geometry", In: Wozny, M.J., Turner, J.U. and Preiss, K., ed., *Geometric Modeling for Product Engineering, IFIP WG 5.2/NSF Working Conference on Geometric Modeling, September 18-22, 1988, Rensselaerville, New York*, (Amsterdam: North-Holland, 1990), pp.221-233
- [85] Aldfeld, B., "Rule-Based Approach to Variational Geometry", In: Smith, A., ed., *Knowledge Engineering and Computer Modelling in CAD, Proceedings of the 7th International Conference on the Computer as a Design Tool, September 2-5, 1986, London*, pp.59-67
- [86] Aldefeld, B., "Variation of Geometries Based on a Geometric-Reasoning Method", *Computer-Aided Design*, Vol. 20, No. 3 (April 1988), pp.117-126
- [87] Roller, D., "An Approach to Computer-Aided Parametric Design", *Computer-Aided Design*, Vol. 23, No. 5 (June 1991), pp.385-391
- [88] Lee, J.Y. and Kim, K., "Geometric Reasoning for Knowledge-based Parametric Design Using Graph Representation", *Computer-Aided Design*, Vol. 28, No. 10 (October 1996), pp. 831-841
- [89] Buchberger, B., Collins, G. and Kutzler, B., "Algebraic Methods for Geometric Reasoning", *Annual Review of Computer Science*, Vol. 3 (1988), p.85-120
- [90] Kondo, K., "PIGMOD: Parametric and Interactive Geometric Modeller for Mechanical Design", *Computer-Aided Design*, Vol. 22, No. 10 (December 1990), pp.633-644
- [91] Kondo, K., "Algebraic Method for Manipulation of Dimensional Relationships in Geometric Models", *Computer-Aided Design*, Vol. 24, No. 3 (March 1992), pp.141-147
- [92] Gao, X-S and Chou, S-C, "Solving Geometric Constraint Systems II: A Symbolic Approach and Decision of Rc-constructibility", *Computer Aided-Design*, Vol.30, No.2 (February 1998), pp.115-122

- [93] Chyz W., “Constraint Management for CSG”, Master Thesis, Massachusetts Institute of Technology, 1985
- [94] Gossard, D.C., Zuffante, R.P. and Sakurai, H., “Representing Dimensions, Tolerances, and Features in MCAE Systems”, *IEEE Computer Graphics & Applications*, Vol. 8, No. 3 (March 1988), pp.51-59
- [95] Owen, J.C., “Algebraic Solution for Geometry from Dimensional Constraints”, *ACM Proceedings of the First Symposium on Solid Modeling Foundations and CAD/CAM Applications, 1991, Austin, Texas*, pp.397-407
- [96] Kramer, G.A., “A Geometric Constraint Engine”, *Artificial Intelligence*, Vol. 58 (1992), pp.327-360
- [97] Hsu, C.Y., and Bruderlin, B., “Constraint Objects – Integrating Constraint Definition and Graphical Interaction”, *ACM Proceedings of the Second Symposium on Solid Modeling and Applications, 1993, Montreal, Quebec, Canada*, pp.467-468
- [98] Solano, L. and Brunet, P., “Constructive Constraint-Based Model for Parametric CAD Systems”, *Computer-Aided Design*, Vol. 26, No. 8 (August 1994), pp.614-621
- [99] Bouma, W., Fudos, I., Hoffmann, C., Cai, J. and Paige, R., “Geometric Constraint Solver”, *Computer-Aided Design*, Vol. 27, No. 6 (June 1995), pp.487-501
- [100] Anantha, R., Kramer, G.A., and Crawford, R.H., “Assembly Modelling by Geometric Constraint Satisfaction”, *Computer-Aided Design*, Vol.28, No.9 (September 1996), pp.707-722
- [101] Latham, R.S. and Middleditch, A.E., “Connectivity Analysis: a Tool for Processing Geometric Constraints”, *Computer-Aided Design*, Vol.28, No.11 (November 1996), pp.917-928
- [102] Fudos, I. and Hoffmann, C.M., “A Graph-Constructive Approach to Solving systems of Geometric Constraints”, *ACM Transactions on Graphics*, Vol.16, No.2 (April 1997), pp.179-216

- [103] Lee, J.Y. and Kim, K., "A 2-D Geometric Constraint Solver Using DOF-Based Graph Reduction", *Computer-Aided Design*, Vol. 30, No. 11 (September 1998), pp.883-896
- [104] Hoffmann, C.M., Lomonosov, A., and Sitharam, M., "Decomposition Plans for Geometric Constraint Systems, Part I: Performance Measures for CAD", *Journal of Symbolic Computation*, Vol.31, No.4 (2001), pp.367-408
- [105] Hoffmann, C.M., Lomonosov, A., and Sitharam, M., "Decomposition Plans for Geometric Constraint Systems, Part II: New Algorithms", *Journal of Symbolic Computation*, Vol.31, No.4 (2001), pp.409-427
- [106] Shih, C.H. and Anderson, B., "A Design/Constraint Model to Capture Design Intent", *ACM Proceedings of the Fourth Symposium on Solid Modeling and Applications, May 14-16, 1997, Atlanta, Georgia*, pp.255-264
- [107] Pratt, M.J., "Requirements Analysis for an Explicit Constraints Schema for STEP", *ISO TC184/SC4/WG3 N502 (T1/Parametrics) May 10th, 1996*, <http://www.nist.gov/sc4/paramet/short/iso/n502.pdf>
- [108] Shah, J.J. and Rogers, M.T., "Functional Requirements and Conceptual Design of the Feature-based Modeling System", *Computer-Aided Engineering Journal*, Vol.5, No.1 (February 1988), pp.9-15
- [109] Shah, J.J. and Rogers, M.T., "Expert Form Feature Modeling Shell", *Computer-Aided Design*, Vol.20, No.9 (November 1988), pp.515-524
- [110] Hoffmann, C.M. and Juan, R., "Erep – An Editable, High-Level Representation for Geometric Design and Analysis", In: Wilson, P., Wozny, M., and Pratt, M., ed., *Geometric Modeling for Product Realization* (North-Holland, 1993) pp.129-164
- [111] Chen, X. and Hoffmann, C.M., "Towards Feature Attachment", *Computer-Aided Design*, Vol.27, No.9 (September 1995), pp.695-702
- [112] Chen, X and Hoffmann, C.M., "On Editability of Feature-based Design", *Computer-Aided Design*, Vol.27, No.12 (December 1995), pp.905-914

- [113] Hoffmann, C.M., "EREP Project Overview", In: Roller, D. and Brunet, P., ed., *CAD Systems Development* (Berlin: Springer, 1997), pp.32-40
- [114] Hoffmann, C.M. and Joan-Arinyo, R., "On User-defined Features", *Computer-Aided Design*, Vol.30, No.5 (April 1998), pp.321-332
- [115] Middleditch A. and Reade, C., "A Kernel for Geometric Features", *Proceedings of the Fourth ACM Symposium on Solid Modeling and Applications, May 14-16, 1997, Atlanta, Georgia*, pp.131-140
- [116] National Institute of Standards and Technology, <http://www.nist.gov/sc4/paramet/short/engen/edm46.pdf>
- [117] Pratt, M.J. and Anderson, B.D., "A Shape Modeling Applications Programming Interface for the STEP Standard", *Computer-Aided Design*, Vol.33, No.7 (June 2001), pp.531-543
- [118] National Institute of Standards and Technology, *Product Data Exchange Specification: The First Working Draft*, NISTIR88-4004, February 1988
- [119] Shah, J.J. and Mathew, A., "Experimental Investigation of The STEP Form-Feature Information Model", *Computer-Aided Design*, Vol.23, No.4 (May 1991), pp.282-296
- [120] Roy, U. and Liu, C.R., "Feature Based Representational Scheme of a Solid Modeler for Providing Dimensioning and Tolerancing Information", *Robotics and Computer-Integrated Manufacturing*, Vol.4, No.3/4 (1988), pp.335-345
- [121] Wang, N. and Ozsoy, M., "A Scheme to Represent Features, Dimensions, and Tolerances in Geometric Modeling", *Journal of Manufacturing Systems*, Vol.10, No.3 (1991), pp.233-240
- [122] Gomes, A.J.P. and Teixeira, J.C.G., "Form Feature Modelling in a Hybrid CSG/Brep Scheme", *Computers & Graphics*, Vol.15, No.2 (1991), pp.217-229
- [123] Rossignac, J.R., "Issues on Feature-based Editing and Interrogation of Solid Models", *Computers & Graphics*, Vol.14, No.2 (1990), pp.149-172

- [124] Kim, C. and O’Grady, P.J., “A Representation Formalism for Feature-based Design”, *Computer-Aided Design*, Vol.28, No.6/7 (June 1996), pp.451-460
- [125] Goldberg, D., “What Every Computer Scientist Should Know About Floating-Point Arithmetic”, *ACM Computing Surveys*, Vol.23, No.1 (March 1991), pp.5-48
- [126] Segal, M. and Séquin, C. H., “Consistent Calculations for Solids Modeling”, *Proceedings of the First Annual Symposium on Computational Geometry, June 05-07, 1985, Baltimore, Maryland*, p.29-38
- [127] Ottmann, T., Theimt, G., and Ullrich, C., “Numerical Stability of Geometric Algorithms”, *Proceedings of the Third Annual Symposium on Computational Geometry, June 08-10, 1987, Waterloo, Ontario, Canada*, p.119-125
- [128] Dobkin D. and Silver, D., “Recipes for Geometry and Numerical Analysis, Part I: An Empirical Study”, *Proceedings of the Fourth ACM Symposium on Computational Geometry, June 06-08, 1988, Urbana-Champaign, Illinois*, pp.93-105
- [129] Hoffmann, C.M., “The Problems of Accuracy and Robustness in Geometric Computation”, *IEEE Computer*, Vol.22, No.3 (March 1989), pp.31-41
- [130] C.M. Hoffmann, *Geometric and Solid Modeling: An Introduction* (San Mateo, California: Morgan Kaufmann, 1989), Chapter 4, pp.111-154
- [131] Sugihara, K. and Iri, M., “A Solid Modeling System Free from Topological Inconsistency”, *Journal of Information Processing*, Vol.12, No.4 (1989), pp.380-393
- [132] Benouamer, M.O., Jaillon, P., Michelucci, D., and Moreau, J.M., “A ‘Lazy’ Solution to Imprecision in Computational Geometry”, *Proceedings of the Fifth Canadian Conference on Computational Geometry, August 5-9, 1993, Waterloo, Ontario, Canada*, pp.73-78
- [133] Yap, C.K., “Towards Exact Geometric Computation”, *Proceedings of the Fifth Canadian Conference on Computational Geometry, August 5-9, 1993, Waterloo, Ontario, Canada*, pp.405-419

- [134] Fortune, S., "Polyhedral Modelling with Exact Arithmetic", *Proceedings of the Third ACM Symposium on Solid Modeling and Applications, May 17-19, 1995, Salt Lake City, Utah*, pp.225-234
- [135] Fortune, S. and Van Wyk, C.J., "Static Analysis Yields Efficient Exact Integer Arithmetic for Computational Geometry", *ACM Transactions on Graphics*, Vol.15, No.3 (July 1996), pp.223-248
- [136] Hoffmann, C.M., Hopcroft, J.E., and Karasick, M.S., "Towards Implementing Robust Geometric Computations", *Proceedings of the Fourth ACM Symposium on Computational Geometry, June 06-08, 1988, Urbana-Champaign, Illinois*, pp.106-117
- [137] Hoffmann, C.M., Hopcroft, J.E., and Karasick, M.S., "Robust Set Operations on Polyhedral Solids", *IEEE Computer Graphics and Applications*, Vol.9, No.6 (November 1989), pp.50-59
- [138] Segal, M., "Using Tolerances to Guarantee valid Polyhedral Modeling Results", *Computer Graphics*, Vol.24, No.4 (August 1990), pp.105-114
- [139] Sederberg, T.W. and Farouki, R.T., "Approximation by Interval Bezier Curves", *IEEE Computer Graphics and Applications*, Vol.12, No.5 (September 1992), pp.87-95
- [140] Hu, C.Y., Patrikalakis, N. M., and Ye, X., "Robust Interval Solid Modeling, Part I: Representations", *Computer-Aided Design*, Vol.28, No.10 (October, 1996), pp.807-817
- [141] Abrams, S.L., Cho, W., Hu, C.-Y., Maekawa, T., Patrikalakis, N.M, Sherbrooke, E.C., and Ye, X., "Efficient and Reliable Methods for Rounded-Interval Arithmetic", *Computer-Aided Design*, Vol.30, No.8 (July 1998), pp.657-665
- [142] Capoleas, V., Chen, X., and Hoffmann, C.M., "Generic Naming in Generative Constraint-based Design", *Computer-Aided Design*, Vol.28, No.1 (January 1996), pp.17-26
- [143] Atallah, Mikhail J., ed., *Algorithms and Theory of Computation Handbook* (Boca Raton: CRC Press, 1999)

- [144] Kripac, J. “A Mechanism for Persistently Naming Topological Entities in History-based parametric Solid Models (Topological ID System)”, *Proceedings of the Third ACM Symposium on Solid Modeling and Applications, May 17-19, 1995, Salt Lake City, Utah*, pp.21-30
- [145] Kripac, J., “A Mechanism for Persistently Naming Topological Entities in History-based Parametric Solid Models”, *Computer-Aided Design*, Vol.29, No.2 (February 1997), pp.113-122
- [146] Wu, J., Zhang, T., Zhang, X., and Zhou, J., “A Face Based Mechanism for Naming, Recording and Retrieving Topological Entities”, *Computer-Aided Design*, Vol.33, No.10 (September 2001), pp.687-698
- [147] Shapiro, V. and Vossler, D.L., “What is a Parametric Family of Solids”, *Proceedings of the Third ACM Symposium on Solid Modeling and Applications, May 17-19, 1995, Salt Lake City, Utah*, pp.43-54
- [148] Stewart, N.F., “Sufficient Condition for Correct Topological Form in Tolerance Specification”, *Computer-Aided Design*, Vol.25, No.1 (January 1993), pp.39-48
- [149] Raghothama, S. and Shapiro, V., “Necessary Conditions for Boundary Representation Variance”, *Proceedings of the Thirteenth ACM Annual Symposium on Computational Geometry, June 4-6, 1997, Nice, France*, pp.77-86
- [150] Raghothama, S. and Shapiro, V., “Topological Framework for Part Families”, *Proceedings of the Seventh ACM Symposium on Solid Modeling and Applications, June 17-21, 2002, Saarbrücken, Germany*, pp.1-12
- [151] Lachlan, R. *An Elementary Treatise on Modern Pure Geometry* (London: Macmillan, 1893), Chapter 1, pp.4-5
- [152] Bell, E.T., *The Development of Mathematics*, 2nd ed., (New York: McGraw-Hill, 1945), Chapter 15, pp.340
- [153] C.M. Hoffmann, *Geometric and Solid Modeling: An Introduction* (San Mateo, California: Morgan Kaufmann, 1989), Chapter 5, pp.193-203

- [154] Dwyer, P.S., “Computation with Approximate numbers”, In: Dwyer, P.S., ed., *Linear Computations* (New York: Wiley, 1951), pp.11-34
- [155] Warmus, M., “Calculus of Approximations”, *Bulletin of Academy of Poland Sciences*, Cl.III, Vol.IV, No.5 (1956), pp.253-259
- [156] Warmus, M., “Approximations and Inequalities in the Calculus of Approximations: Classification of Approximate Numbers”, *Bulletin of Academy of Poland Sciences*, Series of Mathematics, astr: et phys., Vol.IX, No.4 (1961), pp.241-245
- [157] Sunaga, T., “Theory of Interval Algebra and Its Application to Numerical Analysis”, *RAAG Memoirs* 3, pp.29-46
- [158] Moore, R.E., *Interval Analysis* (Englewood Cliffs, New Jersey: Prentice-Hall, 1966)
- [159] Hansen, E.R., “Interval Arithmetic in Matrix Computations, Part I”, *SIAM Journal on Numerical Analysis*, Vol. 2, (1965), pp.308-320
- [160] Moore, R.E., *Methods and Applications of Interval Analysis* (Philadelphia: SIAM, 1979)
- [161] Alefeld, G. and Herzberger, J., *Introduction to Interval Computations* (New York: Academic Press, 1983)
- [162] Moore, R.E., ed., *Reliability in Computing: The Role of Interval Methods in Scientific Computing* (Boston: Academic Press, 1988)
- [163] Hansen, E., *Global Optimization Using Interval Analysis* (New York: Marcel Dekker, 1992)
- [164] Alefeld, G., and Mayer, G., “Interval Analysis: Theory and Application”, *Journal of Computational and Applied Mathematics*, Vol.121, No.1-2 (September 2000), pp.421-464
- [165] Jaulin, L., Kieffer, M., Didrit, O., and Walter, E., *Applied Interval Analysis* (London: Springer, 2001)

- [166] Mudur, S.P. and Koparkar, P.A., “Interval Methods for Processing Geometric Objects”, *IEEE Computer Graphics and Applications*, Vol.4, No.2 (February 1984), pp.7-17
- [167] Toth, D.L., “On Ray Tracing Parametric Surfaces”, *Computer Graphics*, Vol.19, No.3 (July 1985), pp.171-179
- [168] Kalra, D. and Barr, A.H., “Guaranteed Ray Intersections with Implicit Surfaces”, *Computer Graphics*, Vol.23, No.3 (July 1989), pp.297-304
- [169] Moore, M. and Wilhelms, J., “Collision Detection and Response for Computer Animation”, *Computer Graphics*, Vol.22, No.4 (August 1988), pp.289-298
- [170] Von Herzen, B., Barr, A.H. and Zatz, H.R., “Geometric Collisions for Time-Dependent Parametric Surfaces”, *Computer Graphics*, Vol.24, No.4 (August 1990), pp.39-48
- [171] Duff, T., “Interval Arithmetic and Recursive Subdivision for Implicit Functions and Constructive Solid Geometry”, *Computer Graphics*, Vol.26, No.2 (July 1992), pp.131-138
- [172] Snyder, J.M., Woodbury, A.R., Fleischer, K., Currin, B., and Barr, A.H., “Interval Methods for Multi-Point Collisions Between Time-Dependant Curved Surfaces”, *ACM Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, September 1993, New York, NY, pp.321-334
- [173] Wallner, J., Krasauskas, R., and Pottmann, H., “Error Propagation in Geometric Constructions”, *Computer-Aided Design*, Vol.32, No.11 (September, 2000), pp.631-641
- [174] Lin, H., Liu, L., and Wang, G., “Boundary Evaluation for Interval Bezier Curve”, *Computer-Aided Design*, Vol.34, No.9 (August, 2002), pp.637-646
- [175] Ratschek, H. and Rokne, J., *New Computer Methods for Global Optimization* (New York: Ellis Horwood Limited, 1988), Ch.2, pp.28-29
- [176] Hansen, E., “An Overview of Global Optimization Using Interval Analysis”, In: Moore, R.E., eds., *Reliability in Computing: The Role of Interval Methods in Scientific Computing* (Boston: Academic Press, 1988), , pp.289-305

- [177] Nickel, K., “How to Fight the Wrapping Effect”, In: Nickel, K., eds., *Proceedings of the International Symposium on Interval Mathematics, September 23-26, 1985, Freiburg i. Br., Germany*, pp. 121-132
- [178] Yamamura, K., “An Algorithm for Representing Functions of Many Variables by Superpositions of Functions of One Variable and Addition”, *IEEE transactions on Circuits and Systems – I: Fundamental Theory and Application*, Vol.43, No.4 (April, 1996), pp.338-340
- [179] Kolev, L.V., “A New Method for Global Solution of Systems of Non-linear Equations”, *Reliable Computing*, Vol.4, No.2 (May, 1998), pp.125-146
- [180] Kolev, L.V., “Automatic Computation of a Linear Interval Enclosure”, *Reliable Computing*, Vol.7, No.1 (February, 2001), pp.17-28
- [181] Hansen, E.R. and Greenberg, R.I., “An Interval Newton Method”, *Applied Mathematics and Computation*, Vol.12, No.2-3 (May, 1983), pp.89-98
- [182] Collins, G.E. and Johnson, J.R., “Quantifier Elimination and the Sign Variation Method for Real Root Isolation”, *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation, July 17-19, 1989 Portland, Oregon*, pp.264-271
- [183] Collins, G.E. and Akritas, A.G., “Polynomial Real Root Isolation Using Descarte’s Rule of Signs”, *Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation, August 10-12, 1976, Yorktown Heights, New York*, pp.272-275
- [184] Nnaji, B.O., Wang, Y., Kim, K.Y., and Muogboh, O.S., “PEGASUS: A Service-Oriented Product Engineering System Over the Internet”, *IIE Transactions*, under review, 2002