Proceedings of DETC'03
ASME 2003 Design Engineering Technical Conferences and
Computers and Information in Engineering Conference
Chicago, Illinois, USA, September 2-6, 2003

DETC2003/CIE-48228

THE DESIGN EXEMPLAR: THE FOUNDATION FOR A CAD QUERY LANGUAGE

Ameya Divekar,
Research Assistant
Department of Mechanical Engineering
Clemson University
Clemson, SC 29634-0921

Joshua D. Summers[1],
Assistant Professor
Department of Mechanical Engineering
Clemson University
Clemson, SC 29634-0921

ABSTRACT

Design engineers create models of design artifacts with commercial Computer Aided Design (CAD) solid modeling systems and manage the data files through Product Data Management (PDM) systems. These systems stop short of providing support for querying and retrieving data from "within" the CAD data files. A true CAD query language that allows designers the flexibility to describe queries against single and multiple CAD files would be of great benefit for design engineers. This query language ought to be both data-centric and user-centric in nature. The design exemplar, a data-structure that provides a standard representation of design knowledge based upon a general constraint validation and satisfaction algorithm, is shown here to be a concept upon which a CAD query language may be developed. The first required extension of the design exemplar is the inclusion of logical connectives. Some insights into the different levels at which the extensions may be implemented are discussed. Also, some applications retrieving geometric data using this query language are demonstrated. The query language, as it evolves, is expected to support geometric retrieval across domains and offer an all-purpose approach to geometric retrieval.

1 INTRODUCTION

With the proliferation of CAD use, the amount of knowledge that is generated in the course of engineering design will in turn increase. As a result from this immense collection of knowledge, designers will need tools to access the information. Just as the growth of the Internet spawned the development of new information retrieval tools and languages, the CAD market is facing the need to be capable of providing quick, intuitive retrieval tools to its users. The design exemplar is shown to be a concept upon which a CAD query language might be developed and thus support these needed retrieval tools.

This paper seeks to identify the needs of a CAD-specific query language based upon an analysis of the essential characteristics and the tasks performed by traditional query languages. Query languages are non-procedural, high-level computer languages that are primarily focused towards retrieving data held in files and databases. They could also be used for updates, deletions, and additions [1]. Logical connectives, representing the union, intersection, and difference operands of relational algebra, have been identified as necessary extensions to the design exemplar for evolution into a true CAD query language. This paper discusses the different levels at which the logical connectives ought to be implemented. Potential applications across domains where the query language may be used to retrieve geometric information are discussed.

The relational, or hierarchical, data model found in many legacy applications, and the query languages supporting this model, have solved problems facing most data-processing organizations dealing with business data, as the data could be structured according to the data model. In contrast, "spatial" data processing in domains like GIS (Geographic Information Systems) and CAD/CAM (Computer Aided Design/Computer Aided Manufacturing) has not been solved adequately because the database systems require data to be expressed lexically [2].

Database technologies have been investigated for directly supporting CAD data, though no specific database technology has been shown to fully support CAD data [3]. The first normal form constraint in relational database technologies needs data to be expressed as atomic values and requires expensive join operations and may also lead to data inconsistencies while updating data. Complex valued databases were not found adequate in solving data inconsistencies either. Object databases required additional programming for accessing data. Deductive databases allowed access to the location of components in the model hierarchy.

[1] Corresponding author at joshua.summers@ces.clemson.edu

Designers may wish to find CAD models that look similar to an existing model in order to begin a design from a baseline model. In other words they are interested in locating "globally" similar CAD models. Designers often have a concept in mind during the embodiment stage and may look for models that have a similar concept. Designers may also seek to extract implicit information that is not explicitly stored in these models, but which may be inferred. For example, designers may want to find all CAD models in the database that contain a boss. The information "contain a boss" is not stored explicitly in the CAD models. In other words, designers may be looking for "local" similarity between characteristics in CAD models and the desired concept. Thus, designers may wish to query design models based upon both "global" and "local" similarities. While a number of systems [4, 5, 6, 7, 8, 9] have been developed for searching CAD models based on global similarity, flexible and domain independent "local" similarity search driven by the user is not realized yet.

Feature recognition approaches, evolving over the last 25 years, have been commonly used in the CAD domain for retrieving geometric information [10]. The limitations of features were attributed to feature dependence on entities and relationships and, hence, their failure to embody the semantics of the geometric data. The design exemplar goes beyond features and can convert application specific problems to domain independent ones [11]. The design exemplar is a data-structure that provides a standard representation of engineering design knowledge based upon a canonically derived set of entities and constraints for representing topologic and geometric problems [11]. It leverages a standard domain independent vocabulary for geometric mechanical design problems to represent both explicit and implicit design concepts. This vocabulary allows the designer to express geometric queries in the form of the design exemplar. The data-structure of the design exemplar, when coupled with the generic design exemplar algorithm, has the capability to provide for the basic design tasks of pattern-matching, querying, validation and modification. These capabilities of retrieving similar characteristics and changing them are based upon the generic design exemplar algorithm for constructing and submitting constraint problems to a general constraint solving system [11].

Based upon the belief that there is a need for designers to have flexible capability of defining design characteristic queries upon CAD models and the evidence from the literature that existing query languages are not sufficient for supporting CAD data interrogation, the design exemplar is proposed as a foundation for the development of a CAD query language. This paper begins with a discussion on geometric query languages and identifies the needs of a CAD query language. An introduction to the design exemplar is provided. The design exemplar was investigated as a CAD query language [12] discussing limitations and proposing extensions. This paper offers some insights into the initial work in implementing the extensions proposed in [12]. Finally, examples of how the design exemplar may be used as a query language in existing CAD systems are presented.

2 BACKGROUND

Query languages are non-procedural computer languages, where the user specifies what is to be done and not how it is to be done. They are primarily focused on retrieving data held in files and databases. The user has control over the desired functional result. While the principle behind a query language is not limited to any domain specialization; practicality suggests that specific domains may require a customized language, with extended vocabulary and syntactic rules. For example, in a scientific system, a query language might require facilities for handling matrices and numbers with varying units of measurement. The power of a query language to select the data relevant to the user and keep out the irrelevant data could be assessed in terms of the ability to materialize the target data from the underlying data structure and also the variety of ways in which the conditions can be constructed and connected.

This section discusses query dialogues, query formulation and processing, query expression, the components of a query language, and the necessary functions performed by a query language.

2.1 Dialogue

There are two main types of query dialogues: user driven and system driven [1]. In the system driven dialogue, the user responds to messages initiated in the query system. In systems where all records of data have the same structure and same quantity of data, a system driven approach may be useful. User driven dialogue implies that the user employs a language to define his query. Most of the so-called "English-like" query languages today fall into this class. Commands have a defined syntax. Further, only predicates from a specified list may be applied. In CAD/CAM, every record (model) is likely to have varying amounts of data. Moreover, users would be restricted if they are asked to express the desired concept to be queried, through a predefined dialogue driven by the system. Hence, queries in CAD should be user-driven so that the designers have latitude in expressing their concepts in a complete manner.

2.2 Query formulation and processing

A common example of query formulation within the constrained mode of dialogue is the following:

FIND target WHERE qualification

The target data may be a file name, workspace, record name, or collection of data item names from one or more records. The qualification is a set of conditions involving data items in both the target and the related data. The conditions may be numerical or character string comparisons [1].

Query processing generally operates in one of the following two basic ways:

- One record that satisfies the conditions is made available to the user. To retrieve all the records satisfying the conditions, the query statement is repeated in a procedural loop.
- All records that satisfy the conditions are made available to the user. This is implemented either by printing the selected records as they are retrieved or by using a workspace to store the selected records (or their references, pointers, etc.). In this case the display generally shows a count of the number of records selected. Such a workspace helps a user refine queries and provide more flexibility but is not often available.

Complex queries can be built by performing simple comparisons either by using logical connectives (AND, OR, MINUS) in a single query or by nesting. In CAD/CAM, either

approach may be appropriate, as it is irrelevant whether a single CAD model is processed at a time and the results shown or a list of the CAD models is displayed, allowing the users to browse through each record.

2.3 Query expression

Queries may be expressed in different manners and it is important that the appropriate mode of querying be identified for the particular application domain. McWherter, et al., [13] have identified four ways in which queries could be expressed for 2D matching and image retrieval:

1. A *textual* query based on keywords.
2. A query by *example* uses similarity measures derived off a set of query images provided as input.
3. Query by *sketch* looks for image segments matching the sketched profile.
4. *Iconic* queries use templates representing the important aspects of the desired image to identify images with similar features.

Users need to express spatial concepts when they query the database of CAD models. This implies that lexical descriptions of the query in these spatial situations will be ambiguous and may easily lead to misinterpretations. Dialogue boxes and menu driven queries provide little help as they use the same syntax and grammar as lexically expressed languages, only providing support of external memory for the user [14]. Further, since, by its very nature, CAD data is spatial and graphical, it is logical to express it in terms of explicit spatial concepts. Egenhofer [14] argues that users prefer to sketch spatial queries, as they more readily support human spatial thinking. Hence, it is clear that a graphical query language will be more appropriate than a query language that requires the user to formulate the query lexically. Users of a CAD query language may also want to store already formulated queries that may be combined with others to formulate more complex and compound queries. This may also result in savings of time and effort while formulating new queries and hence such a facility of combining pre-existing queries may be expected of a CAD query language.

2.4 Components of a query language

The components of the de-facto query language SQL (Structured Query Language) [15] that are essential in making it a "query language" are identified here. These components include data-types, predicates, and logical connectives. A data type is a set of data with values having predefined characteristics. Variables are instances of one of these data types. Some commonly supported categories of data-types in SQL are numeric, character, Boolean, date/time, and objects. The predicate is a condition that can be evaluated to produce a truth-value of true, false, or unknown. The result is achieved by applying the predicate to a given row of a table. Some examples of these predicates are =, <>, >, <, between, IN, LIKE, IS NULL, IS NOT NULL, or EXISTS. These are also referred to as "comparison operators". They are critical components of a query language as they enable the construction of conditions that allow the users to access the data of interest to them. SQL provides, at a minimum, for the logical connectives: AND, OR, MINUS. These represent the intersection, union, difference of relational algebra. These operators enable the construction of compound conditions within a single query. These operators may also be used to combine two queries so that the resulting retrieved sets of both queries may be obtained.

2.5 Tasks performed by a query language

A query language is expected to perform the following tasks on a database: retrieval, updates, deletions, and additions [1]. Consider a table in a relational database that contains information about the names of employees and the department to which they belong. Table 1 lists the tasks that are typically performed by a query language and provides an example for each task and expresses the query in SQL.

Table 1 - Tasks Performed by a Query Language

Task	Example	Query with SQL
Data Retrieval	Retrieving names of employees from 'CS' Dept.	SELECT Employee_name FROM Employee_table WHERE Dept_name = 'CS'
Data Addition	Adding a record of employee Joe.	INSERT into Employee_table VALUES ('Joe', 'CS')
Data Modification	Changing the Department of Joe from 'CS' to 'ME'.	UPDATE Employee_table SET Dept_name = 'ME' WHERE Employee_name = 'Joe'
Data Deletion	Deleting all employees from 'CS' Department.	DELETE FROM Employee_table WHERE Dept_name = 'CS'

This section identified the qualifications of a query language based on the de-facto query language (SQL). These qualifications are the various components a query language may have and the various tasks it may be expected to perform. Table 2 summarizes the qualifications of a query language by listing the various components and tasks expected from a query language. Each of these is discussed in detail later.

Table 2 - Qualifications of a Query Language

Components	Data-Types
	Predicates
	Logical Connectives
Tasks	Retrieval
	Addition
	Update
	Delete

In addition to the qualifications of a query language a "CAD" query language may be expected to meet the requirements of a spatial query language as given by [14]. Users must be able to treat spatial data at a level independent from internal coding such as x-y co-ordinates. The results should be displayed in graphical form, as it is the most natural form to analyze geometric data. It should be possible to combine one query result with the results of one or more previous queries giving rise to a dynamic interaction. Graphical presentations may require the display of context in addition to the information sought. An extended dialog allowing selection by pointing and direct selection of a result as a reference to an upcoming query is required. Graphical presentation of query results may require dedicated language tools. Labels are important in understanding drawings so that users are able to select specific instances of objects.

3 GEOMETRIC QUERY MECHANISMS

This section describes the existing systems aiming at retrieving geometric information. It is divided into two sections, the first part describes the query languages and/or

mechanisms, while the second part describes the geometric query engines. Special emphasis is placed on the distinction between querying geometric models based upon local similarity and global similarity.

3.1 Query Languages and Mechanisms

There have been many proposals at developing a geometric query language. Most of them have attempted to extend SQL to spatial data. Egenhofer [2] gives a detailed comparison of the query languages GEOQL, Extended SQL, PSQL, KGIS, and TIGRIS. These are all extensions of SQL for the GIS domain.

Chan and Zhu [16] have developed a geometric query language in the GIS domain built on SQL(QL/G). They have extended the features of SQL and accommodated geometric data types and operators. The language uses the geometric data types like REGION, LINE, or POINT that may be adequate to serve the needs in 2D applications like GIS. In CAD where geometric data is defined in three dimensions, these data types may not be sufficient, but could be extended, theoretically, to accommodate data types found in CAD. Further, in GIS applications the geometric data, such as the location of a city, can be uniquely stated whereas in CAD models the locations may change with translation, rotation, and scaling of models.

Another query language that offers support for geometric data types is PostgreSQL [17]. Geometric data types like point, line, line segment, box, path, polygon, and circle can be represented in the language. The language also has a number of geometric predicates that are mapped to a set of operators. The geometric predicates allow the comparison of geometric data-types. For example, predicates can check whether an entity is to the right of, left of, above, below another entity. From the list of data-types and predicates supported, it can be observed that this query language supports 2D geometry only.

Silva, et al., [18] have developed a geometric query mechanism for process planning. This approach allows for querying against a single part file. They assume that the part exists in a feature-based system and "tag" a feature type to each feature. They have defined a list of relations and operations and they process the CAD models to form a representation of the part in the form of tables. These tables have all the relations and the entities in the part satisfying them. A query is processed against this tabulated information using a list of defined operations. This method requires preprocessing of the CAD models before they can be queried.

Yang, et al., [19] have developed a query approach to retrieve features from part files. They use the Attribute Adjacency Graph (AAG) for representing the features of the parts. Their language relies heavily upon preprocessing the CAD data into a new representative format. The transformation module in their system transforms the B-Rep structure of the part files to a feature database. For the transformation they use a set of definitions to which dictates the type of feature in which each instance will fit. They classify the features into a set of primary and secondary features and describe "parent-child" or "same-level" spatial relations between them. The relation table contains all the relations between the features. This approach makes use of SQL to query preprocessed data of CAD models populated in the database. This approach allows designers to query against a database of CAD models and retrieve the features desired by the users. However, the approach has limited capability to query CAD models because only those features and relations defined *a priori* in the system may be used. Moreover, there is no provision for quantitative predicates to compare the dimension values of features.

Rosenthal, et al., [20] used an analogy between the data-structures used for the data about parts, which they referred to as part hierarchies, and the part-of relationship. They have defined geometric envelopes as the bounding boxes that totally contain a part. The geometric predicates may be used to formulate intersection queries, proximity queries and containment queries. The queries retrieve the parts that intersect, lie totally or within a distance from the given volume in an assembly. They classified the predicates as either downward monotonic or upward monotonic, where downward monotonic predicates imply that if a part satisfies the predicate all the descendants of the part in the hierarchy also satisfy the predicate and vice-versa for upward monotonic. Also, they classified the attributes as order preserving or inverse order preserving depending on whether the attributes have lower or higher values for the descendants. The queries are suitable for environments in which the parts are arranged hierarchically and may find application in situations where the engineer would want information regarding the parts that are contained in an assembly, as well as the number of those parts available in the inventory. However, this approach may not be useful in situations where the designer wants information about the features of a particular part.

Koonce, et al., [21] have considered the querying of the data contained in the Express modeling format. This query language is a textual language built on SQL with the processing logic based on the Object Protocol Method [22] and is designed to aid the moving of files from one CAD system to another. This allows proprietary systems to only query data in the STEP files that may be of relevance to them.

Kriegel, et al., [23] have approached spatial database integration for novel CAD applications into off-the-shelf database systems. Their approach relies upon voxelized geometries of VRML (Virtual Reality Modeling Language) models. To enable the generated voxel set to be used as a spatial key, it is transformed to an interval sequence on a space-filling curve and stored in a tree. Their approach relies upon an approximation of the VRML models, which are approximations of original CAD models. Their system allows users to query spatial regions for the parts that may lie completely within a selected region, intersect the selected region, or lie within the specified distance of the selected region. The queries offered by this system are of limited value, as they do not encapsulate the semantics of the data.

From the literature review, it may be seen that there are a number of approaches to retrieving geometric information in the CAD domain while some query languages were developed in the GIS domain. However, to the best of our knowledge, this is a first attempt to derive the essential components of a query language and develop a CAD query language based on the requirements derived from a standard query language.

In comparing these different approaches, a set of comparisons are offered (Table 3). These comparisons include the dimension of the geometric information (2D vs. 3D), the scope of querying (single file vs. multiple files), the processing of data required (pre-processing vs. no processing), the application domain, and the form of querying.

Table 3 - Comparison of Query Approaches

Query Approaches	2D/3D	Single (S)/ Multiple (M)	Pre-processing Required (Y/N)	Application. Domain	Form of Querying
[16]	2D	S	N	GIS	Mixed Lexical/ Graphical
[21]	3D	S	?	CAD	Lexical
[18]	3D	S	Y	CAD	Lexical
[19]	3D	M	Y	CAD	Lexical
[20]	3D	M	N	CAD	Lexical and Guided
[17]	2D	S	N/A	General	Lexical
[23]	3D	M	Y	CAD	Unknown

3.2 Geometric Search Engines

Designers may be looking for models that are globally similar to an existing model. There are many researchers working on retrieving globally similar models and the research has culminated into many geometric search engines. Some popular 3D geometric search engines include: 3D Digital Library [4], 3D Geometry Search [6], ShapeSearch.net [24], Geometric Search Engine [9], 3D Search Engine [25], 3D Object Recognition [5], 3D Shape Retrieval Engine [26], Berchtold, et al., [27], and Peabody, et al., [28].

There are conceptually three different types of queries in the context of CAD models:

1. Retrieving CAD models that match the constructed model globally.
2. Retrieving CAD models that match the characteristics expressed in the query.
3. Retrieving parts from an assembly using geometric information.

Table 4 identifies the type of queries employed by different search engines and approaches.

Table 4 – Comparison of Different Approaches

Retrieving globally similar models	Retrieving CAD models that match the characteristics expressed in the query.	Retrieving parts from an assembly using geometric information.
All systems categorized under geometric search engines	[19] [18]	[20] [23]

Most of the work in geometric query languages has been done in the field of GIS, with a majority of them being mere extensions of SQL. In GIS systems users deal with maps, and the geometric data in the maps is populated in the database. Thus, the whole database is the information from a map and users usually query against positions of various entities like rivers, cities, towns etc. in the map. It is important to realize that CAD-users are dealing with geometry of a single CAD model as well as the entire database of CAD models. This is an important distinction between CAD query systems and GIS query systems.

4 EXEMPLAR DEFINITION

Exemplars provide a standard representation of mechanical engineering design knowledge based upon a canonically derived set of entities and constraints for representing topologic and geometric design problems [29]. Initially conceived to provide a data structure that could be used to reason about explicit and implicit characteristics of a design model, the exemplar has been at the genesis of feature recognition systems [30], modeling of standard design procedures [31], rule validation and querying [11], case based design (CBD) [32], CAD query language [12], and view transformations [33]. Exemplars are bi-partite graphs, where one set is composed of a number of entities and the other set, a number of relationships or constraints, such that every member of the first set is related to at least one member of the other set and no member of the same set. The exemplar is composed of two pairs of orthogonal sub bipartite graphs (Figure 1) of entities and constraints: match/extract (used for retrieval) and alpha/beta (used for modification). The entities and constraints are based on those derived by Bettig and Shah [34]. A bi-partite graph representation scheme was chosen as it fits well with boundary representation for geometry and equation set modeling for parametric representations.

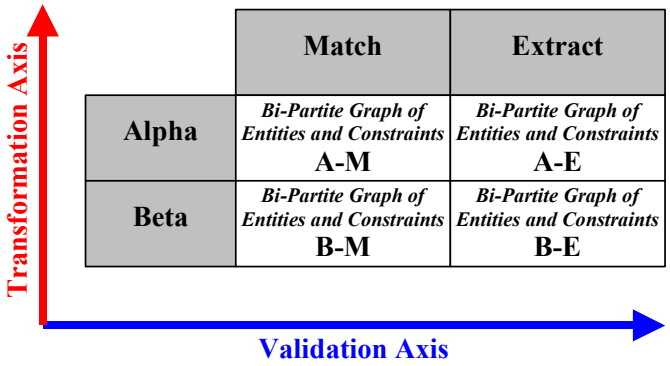


Figure 1 - Components of the Design Exemplar (based upon [29])

One sub-graph (match) of the exemplar corresponds to the entities and constraints that are explicitly stored in the model. In a B-Rep model, this may often consist of the entities related by the boundary constraints and other constraints that the designer may have explicitly imposed on the model. These are represented by solid lines in the representation scheme. The other sub-graph (extract) represents the information that is not stored explicitly in the model, but may be inferred through reasoning. The "*extract*" part of the exemplar is represented by dashed lines in the representation scheme (Figure 3). The extract part represents the relations that must hold true in addition to the matched part, thus facilitating reasoning about the matched part of the exemplar. The transformation axis of the exemplar represents the alpha and beta sub graphs of the exemplar and allow for modification of models from alpha state to the beta state.

While querying, designers might be looking merely for pattern matches to their specified queries or models that satisfy the conditions specified in the extract part of the exemplar, in addition to the pattern match. They might also want to modify/add/delete information to an existing model. Table 5

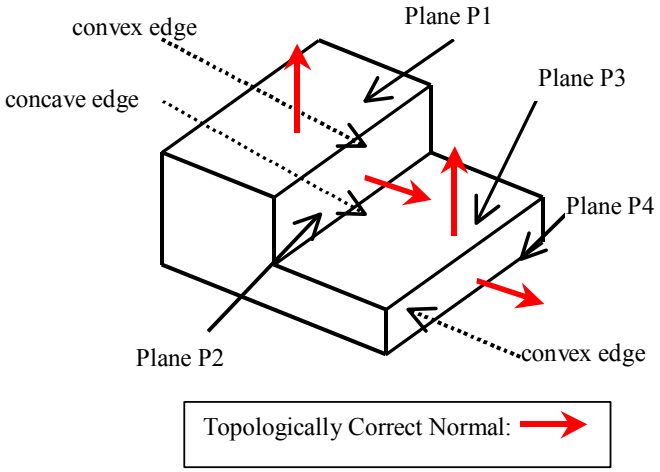
illustrates the pairing between the different tasks expected of a query language with the sub graphs of the design exemplar.

Table 5 - Query Language Tasks vs. Exemplar Sub-Graphs

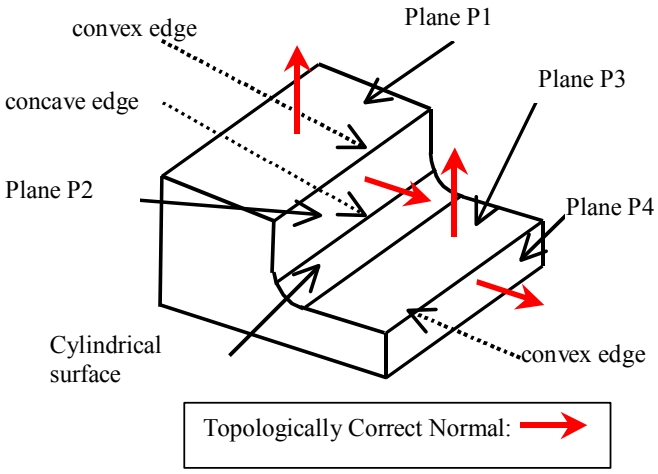
Query language task	Alpha		Beta	
	Match	Extract	Match	Extract
Retrieval	●	❍		
Modification	●	❍	●	❍
Addition	●	❍	●	❍
Deletion	●	❍	●	❍

The solid marks indicate the sub graphs that must exist while the empty ones indicate the optional sub graphs. For the retrieval task, only the pattern match may be sufficient and more sophisticated queries may be formed with the alpha extract sub graph along with the alpha match. The modification, addition and deletion tasks essentially center on locating the characteristic to be modified and then transforming to the final desired state. These require the alpha and beta sub graphs, with the alpha sub graph locating the characteristic to be modified and the beta sub graph representing the final state desired.

The step feature, as shown in Figure 2a, consists of the four planes and every two planes sharing an edge with the correct convexity. It is desired to retrieve the step feature and add a round at the concave edge. As this is an example of model modification, the exemplar will have both the alpha and beta sub-graphs and the transformation from the alpha sub-graph to the beta sub-graph will result in modification of the model. The information regarding the planes and the curves that bound them is found explicitly in the model and constitutes the match portion of the q_step exemplar as shown in Figure 2b. The information about the convexity of the edges may not be necessarily stored explicitly in the CAD model and needs to be reasoned. This constitutes the extract portion of the exemplar and is represented by dashed lines. The convexity of the edges is determined by a sequence of steps. Topologically correct normals to the planes are determined and a cross product of the vectors is taken. The resultant vector is then compared with the correct direction of the curve to infer the convexity of the edge.



(a) Sharp Step Feature



(b) Filleted Step Feature

Figure 2 - Example of a Step Feature

To explain the q_step exemplar better, it is described in two parts, the first queries the convexity of an edge and the other retrieves the step feature and adds a round feature to it thus transforming Figure 2a to Figure 2b. Figure 3 shows the exemplar representation for retrieving the step feature and modifying it. The entities and relationships that only exist in the initial model are "alpha only", those only in the transformed model are "beta only" and those that are in both the initial and the transformed model are "alpha and beta". The concave edge present in the initial model is replaced by two lines that bound a cylindrical surface along with the two circular curves.

While a query to retrieve a rather simple feature was demonstrated here, a range of more complicated features have been retrieved using the exemplars [30]. Also, the modification of geometric information has been demonstrated using the exemplar.

The various aspects of a query language were discussed in Section 2. The qualifications were summarized and the appropriate qualities with respect to a CAD query language were outlined. The design exemplar is discussed here as it relates to these various aspects of a query language. The design exemplar allows queries to be expressed through a graphical interface and the queries are user-driven. The implementation of the design exemplar allows users to sketch the queries and hence allows expression of spatial queries in a graphical manner [14].

The exemplar was investigated against the de-facto query SQL [12]. Table 6 shows the components and tasks performed by the exemplar as they relate to the qualifications outlined by Table 2. The table shows representative data-types and predicates and the complete list may be found in Bettig [29].

The design exemplar is also shown to satisfy the requirements of a spatial query language given by Egenhofer [14], Table 7 is offered as an investigation of the different requirements.

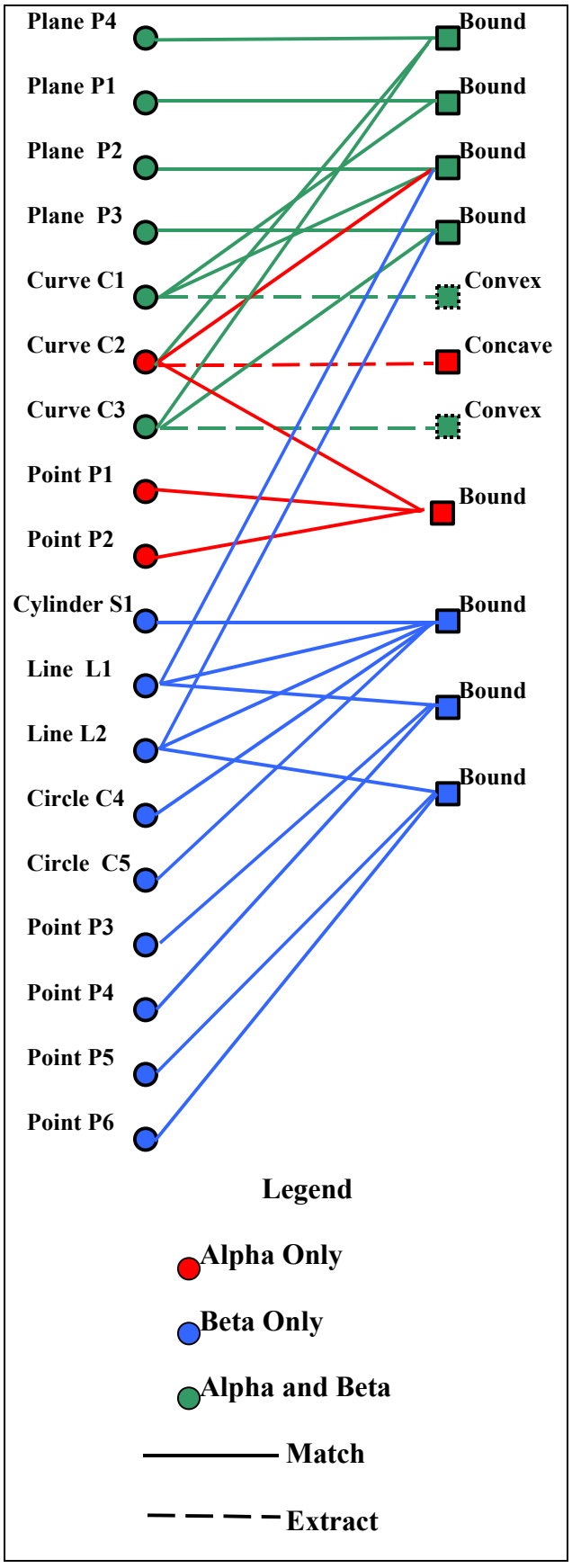


Figure 3 - Exemplar to Modify a Sharp Step to a Filleted Step

Table 6 – Query Language Qualifications vs. Design Exemplar

	Qualifications of a query language	Design Exemplar
Components	Data-types	Real parameter, Integer parameter, Vector, Rotation Matrix (Algebraic), Point, Direction, Line, Plane, Circle, Ellipse, Cylinder, Sphere (Geometric), Solid volume (Topologic), Form Features, Part, Assembly (Semantic)
	Predicates	Scalar equations, Scalar inequalities, Fixed Tables, Vector equation, Cross Product(Algebraic) Distance Angle Radius, Focal Distance, Distance to resolved geometry, Control points, Knot values, Continuity conditions, In_Set, Map Coincident ,Incident Parallel ,Right Angle(Geometric) Boundary, Length, Area, Volume, Directed-Left-Of, Curve Direction, Curve Direction TC, Surface Normal, Surface Normal TC, Same Direction(Topologic)
	Logical Connectives	AND OR, NOT, MINUS (to be implemented)
Tasks	Retrieval	Pattern Matching (Alpha/Match) Query Extraction (Alpha/Match and Alpha/Extract) Design Validation (Alpha/Match and Alpha/Extract)
	Modification, Addition, Deletion	Model Modification (Alpha/Match, Alpha/Extract, Beta/Match, Beta/Extract)

Table 7 - Requirements of a Spatial Query Language vs. the Design Exemplar

Requirements of a spatial query language	Does exemplar comply?
Ability to treat spatial data at a level independent from internal coding such as x-y co-ordinates.	Yes
Display results in graphical form	Yes
Combine one query result with results of one or more previous queries.	Yes
Display of context in addition to information sought	Yes
Extended dialog allowing selection by pointing and direct selection of a result as a reference to an upcoming query.	No
Labels to aid understanding of models so that users are able to select specific instances of objects.	Limited

The design exemplar has the data types and the predicates that may be expected of a CAD query language. While the AND logical connective is inherently implemented in the exemplar, the NOT, OR, and MINUS logical connectives need to be implemented. The exemplar has been shown to perform all the tasks expected of a query language as well as the desired characteristics of a spatial query language [12]. The next section offers some insights implementing the logical connectives, believed to be the necessary extensions to the exemplar in making it a query language. Table 8 compares the

tasks that can be performed by prevalent query languages and systems.

Table 8 - Comparison of Design Exemplar with Other Query Languages/Systems

Query System or Language	Typical tasks performed	Can exemplar do tasks?
Chan, et al.(1996)	Querying positions and attributes of entities in the map.	Yes
Koonce, et al., (1998)	File management on STEP files	Limited
Beaman, et al., (1991)	Querying a CAD model for features that can be machined from a certain dirn.	Yes
Ou-Yang, et al., (1999)	Querying a database of CAD models for features with specified relations	Yes
Rosenthal, et al., (1994).	Region queries and part quantities	Limited
PostGre SQL	Relative/ Absolute positions of entities	Yes
Kriegel, et. al, (2001)	Volume Query Collision Query Clearance Query	Limited

5 IMPLEMENTATION OF THE CONNECTIVES

A unique aspect of the design exemplar as a CAD query language is that it enables the user to use the vocabulary of CAD modeling to formulate the queries. To elucidate further, the query may be expressed at various levels of data abstraction. Therefore, it seems natural that the logical connectives also be implemented at the possible levels of abstraction. The following levels of abstraction seem to be adequate in implementing the logical connectives: entity, constraint, exemplar, and super-exemplar. It may be noted that the levels need to be coherent with the exemplar vocabulary and are implemented accordingly.

5.1 Entity level

The first level of abstraction at which logical connectives might be applied is the entity level. Consider a query to retrieve radially oriented holes in cylindrical components such as crankshafts, camshafts, etc. While a normal query retrieves all the CAD models that contain holes, a query constraining the top curve of the hole to "NOT" be bounded by a planar surface will retrieve the desired holes. Figure 4 shows a representative figure of a CAD model where a hole is not found on a planar face. Holes that are found on non-planar faces may require additional fixturing costs in manufacturing. Therefore, it may be desirable to find holes that are found on any surface except planar. The most straightforward approach to this would be to restrict the type of surface upon which a hole is found. Figure 4 illustrates a hole on a non-planar surface.

The exemplar to retrieve holes similar to the ones described above has been shown in Figure 5. The cylindrical surface of the holes is bound by two curves, C1 and C2. These curves are again bounded by the top surface and the bottom surface. Here, a "NOT" logical connective is added to the entity S1 to restrict it from being of planar type. In this manner, only those holes that are not bounded by a planar surface are retrieved.

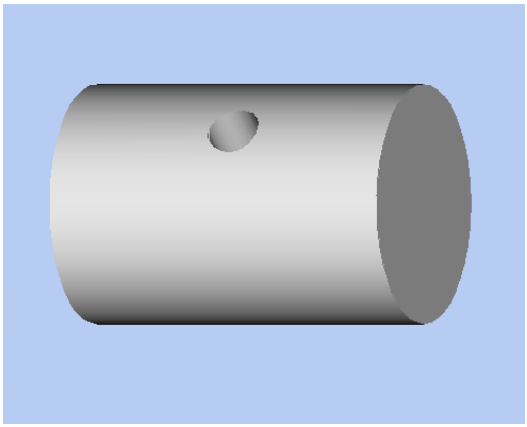


Figure 4 - Hole Not Bounded by a Planar Face

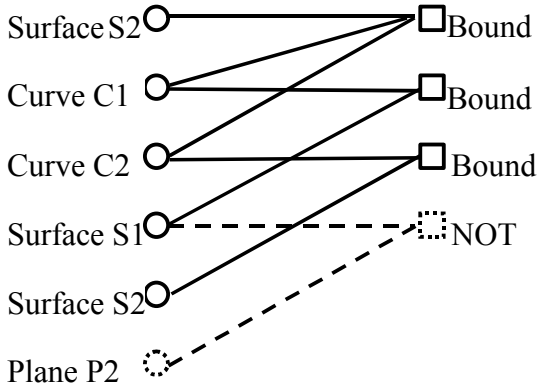


Figure 5 - Exemplar for Retrieving a Hole not Bounded by a Planar Surface

5.2 Constraint level

The query language based on the exemplar is believed to be a step beyond features because of its ability to retrieve values of key parameters, dimensions, and locations of parts, which may be of special significance in embodiment and detailed design that requires sizing and geometric arrangement [10]. Implementation of the logical connectives at the constraint level helps in the formulation of more pertinent queries and eventually contributes to the evolution of the exemplar as a query language. Consider a query that seeks bosses in CAD models but at the same time puts a restriction that their height may be between some values a and b.

$$Eq1 = (h > a)\ AND\ (h < b)$$

Thus it seems important that the logical connectives be implemented at the constraint level.

Consider the retrieval of datum planes perpendicular to each other or parallel to each other as shown in Figure 6. The "OR" logical connective may be used to formulate the corresponding exemplar. The implementation of the logical connectives at the constraint level will enable the expression of all such conditions. In this situation, the designer may be looking for planes that might be good candidates for datums. Typical datums are sought that are perpendicular or parallel to each other. To provide the flexibility of defining a query to find either situation, an "OR" connective may be used between the parallel and perpendicular constraints.

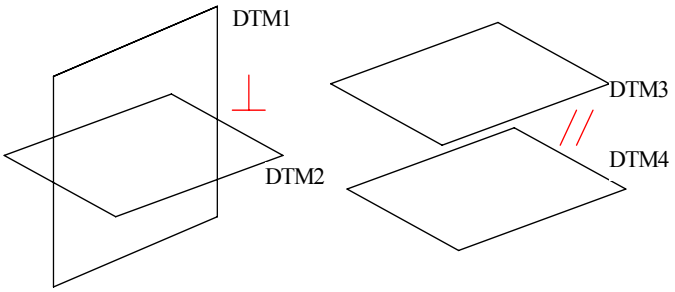


Figure 6 - Retrieving Datum Planes Parallel or Perpendicular to Each Other

An exemplar to retrieve such datum planes is shown in Figure 7, where the datum planes form the match portion of the exemplar and the parallel and perpendicularity constraints have to be reasoned, and hence constitute the extract portion of the exemplar. The "OR" logical connective is used here and provides the information that either of the two constraints is acceptable. The logical connectives may only be applied between two or more constraints.

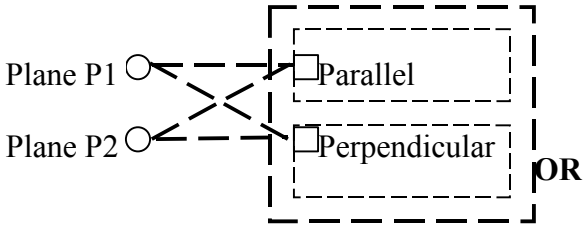


Figure 7 - Exemplar to Retrieve Parallel or Perpendicular Datum Planes

5.3 Exemplar Level:

The power of the logical connectives in formulating more appropriate and powerful queries may be realized by implementing them at the exemplar level. In many queries it may be necessary to impose conditions on different patterns of entities and constraints. Implementation at this level of abstraction will be through the concept of "blocks" [29] (Bettig, 1999). These blocks will be bi-partite graphs by themselves and appropriate logical connectives may be used with these blocks to formulate related queries. Consider a query formulated to retrieve CAD models (Figure 8) that have a boss feature without a hole. The top surface of the boss should not be bound by a curve that is concave. A single exemplar may be written that expresses this query by looking for bosses that are bounded on the top surface by only one edge loop. To express this, a "NOT" block may be created around secondary edges bounding the top surface. In this manner, only the figure on the left would be retrieved.

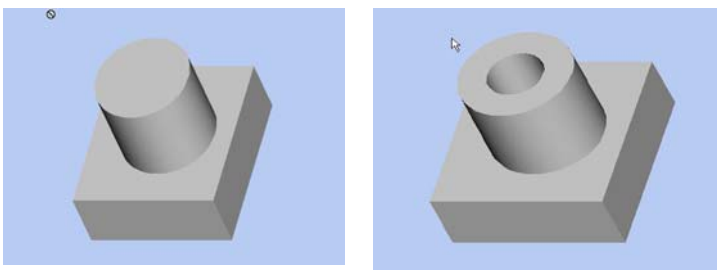


Figure 8 – Boss Models (with and without hole)

To retrieve CAD models that contain bosses without a hole, an exemplar has been written and is shown in Figure 9. As seen from Figure 8, the top surface of the boss is bounded by two curves in case of the boss with the hole bosses with holes. In other words, we need to exclude this characteristic while retrieving bosses that do not have holes. This may be achieved by expressing the above characteristic through a boundary constraint and a curve and applying the NOT block to the undesired pattern. The logical connectives at this level may be applied to a combination of entities and constraints, unlike the earlier levels of abstraction.

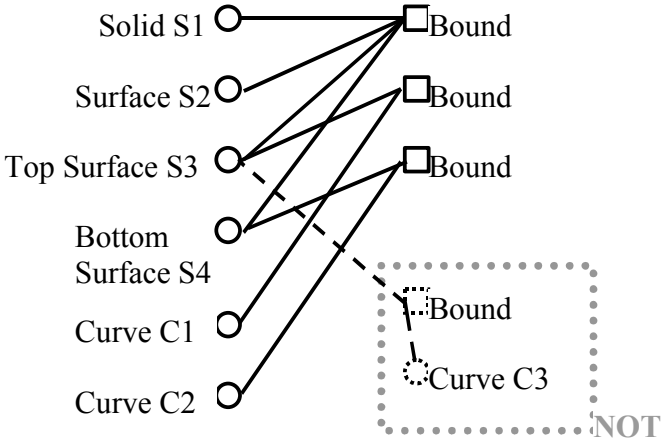


Figure 9 - Exemplar to Retrieve Bosses without Holes

5.4 Super-Exemplar Level:

The super-exemplar level of data abstraction is defined here as the level at which the logical connectives may be applied between different exemplars. This allows the designer to query CAD models and selectively filter out or combine different queries to retrieve the desired information. Consider an example where the user is interested in retrieving CAD models that contain holes or pockets. Exemplars to query these may be combined using the "OR" logical connective. Thus, this query may be formulated as "q_hole **OR** q_pocket". Consider Figure 10 where four different models are represented. There is no restriction on the interaction between the hole and the pocket characteristics. All that is being sought are those models that have either a hole or a pocket or both. The results from applying a hole query and a pocket query are illustrated. At this level, the processing algorithm will use the results of individual queries and combine them using predicate logic to deduce the final result of the query. With extended predicate logic more complex and compound queries may be formulated.

An abstracted representation of a query to find models with either a pocket, a hole, or both is represented in Figure 11. The q_pocket and q_hole design exemplars are represented as black boxes for simplicity [32]. An "OR" block surrounds them, indicating that if either characteristic is true, then the total query is true.

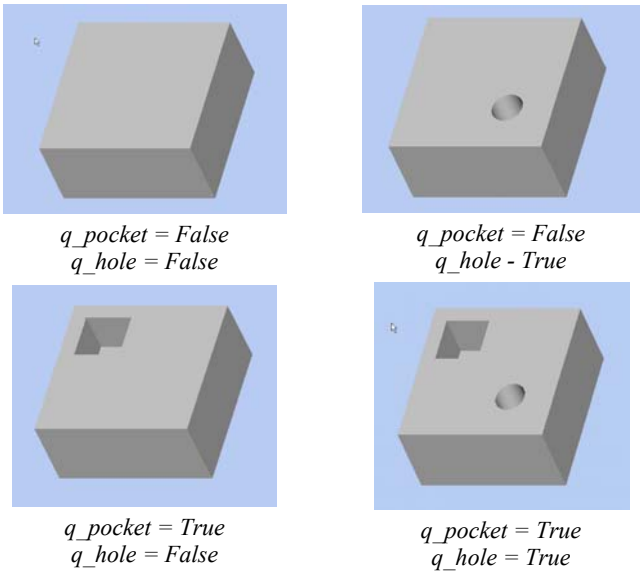


Figure 10 – Hole and Pocket Models

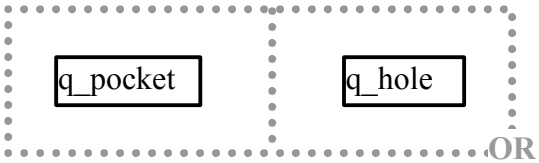


Figure 11 – Design Exemplar to Retrieve Models Having Either a Hole or a Pocket

Table 9 illustrates the truth tables from the different logical connectives that are required at each level. Many of these connectives may be assembled from each other. From the users perspective, this is irrelevant. The operation and implementation of solving for these connectives is hidden from the user.

Table 9 - Logic Truth Tables for Logical Connectives

A	B	AND	OR	NOT (A)	NAND	NOR
True	True	True	True	False	False	False
True	False	False	True	False	True	False
False	True	False	True	True	True	False
False	False	False	False	True	True	True

6 APPLICATIONS OF THE QUERY LANGUAGE

The power of such a CAD query language may be truly realized by leveraging it in applications that require the retrieval of geometric information. Often, automated Design for Manufacturing/Assembly (DFX) systems have a common overall approach that may be described by the following steps:

- Retrieve geometric information: Currently, this is achieved through an appropriate feature recognition approach for different systems. The feature recognition approach in each system is tailored to suit the particular domain for which the system is designed.
- Analysis of the geometric information: An algorithm designed for the particular system operates on this information retrieved in the previous step.

It is believed that while the algorithm might naturally be different for different systems, the CAD query language based on the design exemplar provides a common approach to retrieve the geometric information irrespective of the domain of the application. Three applications across various domains, where the query language could be used to retrieve geometric information, are demonstrated here with some of the required queries formulated as design exemplars.

Lockett and Guenov [35] have developed a system that assists designers of products for casting and injection molding. It recognizes that a mid-surface abstraction of the part's geometry is a suitable approach to represent injection molding components. It generates a mid-surface approximation of the model, extracts the geometric and topologic information from the model, and generates a graphical representation of the geometry that is used as the basis for searching for features from a feature library. Finally, a set of manufacturing rules is applied to the model and design alternatives are recommended. The features commonly encountered in the injection molding components are fins, holes, bosses, T-junctions, X-junctions, ribs and buttresses. Figure 12 demonstrates a design exemplar written to recognize the T-junction feature. The T-junction feature consists of the six planes that satisfy the parallel and perpendicularity constraints. The six planes of the exemplar form the match portion while the other constraints form the extract. Additional design exemplars may be developed for the other queries that are required for this system. The CAD developer has the flexibility to define the queries in the language of the design exemplar without necessitating the development of system-specific search algorithms.

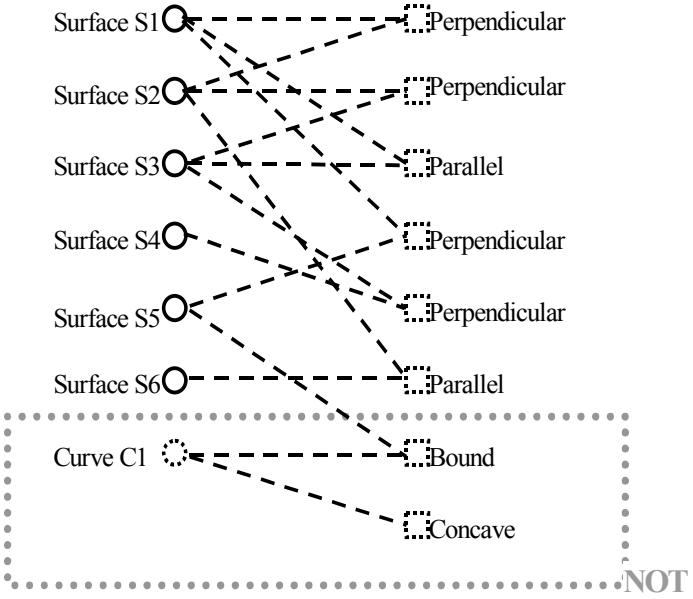


Figure 12 - T-Junction Design Exemplar

Rangel and Shah [36] integrate CAD and CAM by implementing a Computer Aided Process Planning (CAPP) system within commercial CAD software. Normally, three major tasks constitute automatic process-planning systems: machining feature-recognition, operations planning, and setup planning. As an example of where the design exemplar might be employed as a query language for the feature recognition system, consider the example of determining the convexity of curves. Using this exemplar, more complex exemplars might be developed to recognize cut-on features, cut-through features, or cut-around features [37]. The exemplar shown in Figure 13 uses the topologically correct normals to the surfaces and a cross product of the normal vectors V1 and V2 to find the

resultant vector V4. The topologically correct direction of the curve C1 is extracted and the direction of the corresponding vector V3 is compared with that of vector V4 to determine the convexity of the edge.

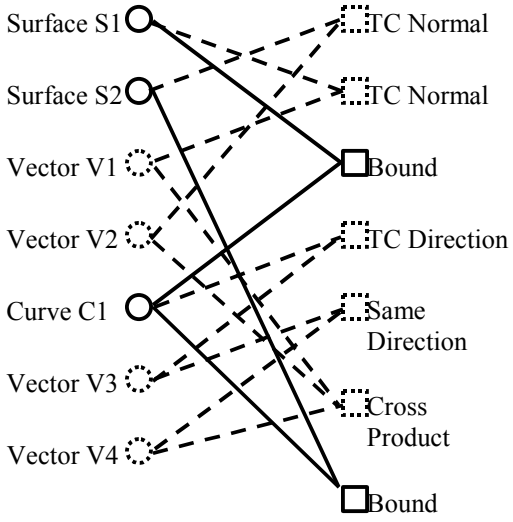


Figure 13 - Exemplar to Determine the Convexity of an Edge

Joshi and Dutta [38] consider feature simplification of sheet metal components, usually modeled as free-form surface models, to generate an efficient finite element mesh. NURBS surfaces are commonly used to represent the complex freeform shapes. A critical step in this approach is the recognition of holes and fillets from these surfaces. Since these are NURBS-based freeform surfaces, the holes are not represented as circles but a set of edges. In this work, the definition of a hole is any loop of edges that has no surface on the inside. The algorithm for recognizing a hole in these free-form surfaces relies on finding free edges on the surfaces and then filter out the edges that form the external loop. A query to retrieve the free edges from a surface is demonstrated in Figure 14.

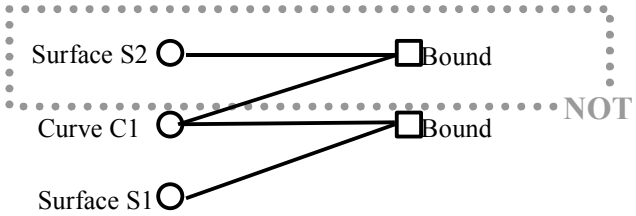


Figure 14 - Exemplar to Retrive the Free Edges from a Surface

The above applications illustrate the potential capability of this CAD query language to retrieve geometric information across domains. It may be noted that this is significant, as in most previous query systems [18, 19, 21] only domain dependent applications were demonstrated or a limited set of features was used. Moreover, the implementation of the design exemplar allows for the CAD models in the database to be stored in the STEP format, thus making it independent of the commercial CAD system used for modeling.

7 DISCUSSION

This paper focused on the current status of the project aiming to develop a complete CAD query language. The necessary extensions identified will be implemented at various levels of data abstraction and will enable the users to formulate queries to retrieve only the relevant data.

The design exemplar offers the expected functionalities of query language. It effectively performs the functions of retrieving, adding, modifying and removing geometric data from CAD models. It satisfies the general format of a query:

FIND target WHERE qualification

The exemplar finds the CAD files (*target*), which satisfy the match and extract portions of the exemplars (*qualification*). The exemplar relies upon the algebraic, semantic, topologic, and geometric entities and constraints; facilitating multiple levels of domain independent design queries as can be seen in [30, 32, 37].

With the implementation of the logical connectives the design exemplar is expected to evolve into a CAD query language. The potential users of such a query language may include anyone retrieving geometric information for various purposes, but two groups of people are anticipated to benefit the most from such a query language.

The first group of people, as discussed earlier are designers looking for CAD models that match the concept in mind. Since this query language also supports modification of geometric data, designers may automate such tasks using the modification queries. For example, a m_counter_bore exemplar may be used to remove any counter-bores found in the database of CAD models. The query may be made more specific by including more constraints in the extract and may allow such a modification to be applied to only the desired holes.

The second group of people anticipated to benefit from this query language is researchers developing various automatic DFX systems. The query language may eliminate the need to develop special retrieval algorithms and enable the researchers to focus on the processing of the geometric information retrieved by the query language.

The design exemplar has been shown to be a first step towards the development of a CAD query language. It overcomes the limitations of earlier approaches, operating in a domain independent environment and using either a pre-defined library of queries or user defined queries. The design exemplar allows users to build their own queries (user-centric) while enabling them to operate with the CAD vocabulary (data-centric) at the same time. This query language goes beyond features by enabling the users to encapsulate the semantics of the geometric data and provides for comparison of CAD models based on key parameters and dimensions. It is believed that as the exemplar based query language evolves, it will play a vital role, not only in retrieving geometric information during the various design stages but also in automation of applications throughout the design process. Current work aims at implementing the extensions and also deriving a similarity measure by which the CAD models may be indexed based on the results of previous queries. It is believed that this query language may ultimately provide the desired general approach to retrieve geometric information.

REFERENCES

[1] Query Language Group, 1981, *Query Languages: a Unified Approach*, Heyden & Son Ltd., The British Computer Society, London, UK.

[2] Egenhofer, M., 1994, "Spatial SQL: A Query and Presentation Language", *IEEE Tranactions on Knowledge and Data Engineering*, 1994, **6 (1)**, pp. 86-95.

[3] Liu, M., 1999, "On CAD Databases", *Proceedings of IEEE Canadian Conference Electrical & Computer Engineering (CCECE'99)*, Edmonton, Canada, May 9-12, 1999.

[4] Arizona State University, 2002, "ASU – Prism", http://3dk.asu.edu.

[5] Brown University, 2002, "3D Object Recognition" http://www.lems.brown.edu/ ~cmc/3DRecog/ overview.html

[6] IBM - Tokyo Research Lab, 2002, "3D Geometric Search", 2002, http://www.trl.ibm.com/ projects/3dweb/SimSearch_e.htm

[7] Kazhdan M., 2001, "Reflective Symmetry Detection in Three Dimensions", *Workshop on Shape-Based Retrieval and Analysis of 3D Models*, Princeton, New Jersey, October 28-30, 2001.

[8] Min, P., Chen, J., Funkhouser, T., 2001, "Evaluating Sketch Query Interfaces for a 3D Model Search Engine", *Workshop on Shape-Based Retrieval of 3D Models*, Princeton, New Jersey, October 28-30, 2001

[9] Sandia Laboratories, 2002, "Geometric Search Engine", http:www.sandia.gov/src/Working_with_Us/Organization_Chart/IS_Principles/Geometric_Search_Engine/geometric_search_engine.html.

[10] Shah, J., Anderson, D., Kim, Y., Joshi, S., 2001, "A Discourse on Geometric Feature Recognition from CAD Models", *Journal of Computing and Information Science in Engineering*, **1**, pp. 41-51.

[11] Bettig, B., Shah, J., Summers, J., 2000, "Domain Independent Characterization of Parametric and Geometric Problems in Embodiment Design", *ASME DETC-2000*, DAC-14259, Baltimore, MD.

[12] Divekar, A., Summers, J., 2003, "Investigation of the Design Exemplar as a CAD Query Language", *International Conference on Engineering Design*, ICED 03, Stockholm, Sweden, August 2003.

[13] McWherter, D., Peabody, M., Regli W., Shokoufandeh, A., 2001,"Clustering Techniques for Databases of CAD Models", *Technical Report DU-MCS-01-01,* Department of Mathematics and Computer Science, Drexel University, Philadelphia, PA.

[14] Egenhofer, M.: 1997, "Query Processing in Spatial-Query-By-Sketch", *Journal Of Visual languages And Computing*, **8 (4)**, pp. 403-24.

[15] Hursch, C., Hursch, J., 1991, *SQL Structured Query Language*, Windcrest, Inc., Blue Ridge Summit, PA.

[16] Chan, E., Zhu, R., 1996, "QL/G – A Query Language for Geometric Data Bases", *Proceedings of the 1st International Conference on GIS in Urban Regional and Environmental Planning*, Samos, Greece, April 1996, pp. 271-86.

[17] PostgreSQL, 2002, "PostgreSQL", http:// techdocs.postgresql.org.

[18] Silva, R., Wood, K., Beaman, J, 1991, "An Algebraic Approach to Geometric Query Processing in CAD/CAM Applications", *Symposium on Solid Modeling Foundations and CAD/CAM Applications*, Austin, TX, p. 73.

[19] Ou-Yang, C., Liou, P.Y: 1999, "Applying the Topological Relationships of Form Features to Retrieve Part Models from a CAD System", *IIE Transactions*, **31**, p.323-37.

[20] Rosenthal, A., Heiler, S., Manola, F., 1984, "An Example of Knowledge Based Query Processing in a CAD/CAM DBMS", *Proceedings of the tenth International Conference on Very Large Data Bases*, 1984, pp. 363-70.

[21] Koonce, D., Huang, L., Judd, R. 1998, "EQL: An Express Query Language", *Computers In Engineering*, **35 (1-2)**, pp. 271-74.

[22] Chen, Markowitz,V.,1996, The Object-Protocol Model Version 4.1, Lawrence Berkeley Labs, *Technical Report*, LBNL-32738, 1996.

[23] Kriegel, H, Müller, A, Potke, M, Seidl, T: 2001, "DIVE: Database Integration for Virtual Engineering", *Demonstration Proceedings 17th Int. Conference on Data Engineering (ICDE),* Heidelberg, Germany, pp. 15-18.

[24] Heriot-Watt University, 2002, "ShapeSearch.net", http://www.hw.ac.uk/mecWWW/research/Dsearch/3Dsearch.htm

[25] Princeton, 2002, "3D Search Engine", http://www.cs.princeton.edu/gfx/proj/shape

[26] Universiteit Utrecht, 2002, "3D Shape Retrieval Engine" http://www.cs.uu.nl/centers/give/imaging/3Drecog/3Dmatching.html

[27] Berchtold, S., Keim, D., 1998 "Section Coding: A Similarity Search Technique for the Car Manufacturing Industry", *IADT*, pp. 256-63.

[28] Peabody, M, Regli, W., Mc Wherter, D., Shokoufandeh, A, 2001, "Clustering Solid Models for Database Storage", *Technical Report DU-MCS-01-04*, Department of Mathematics and Computer Science, Drexel University, Philadelphia, PA.

[29] Bettig, B., 1999, "A graph Based Geometric Solving System for Mechanical Design Problems", *Ph.D. Dissertation*, Arizona State University, Tempe, AZ.

[30] Venkatamaram, S., Summers, J., Shah, J., 2001, "An Investigation of Integration of Design by Features and Feature Recognition", *Feature Modeling and Advanced Design for the Life Cycle Systems*, IFIP, Valenciennes, France.

[31] Bettig, B., Summers, J., Shah, J., 2000, "Domain Independent Knowledge Representation of Parametric and Geometric Problems in Embodiment Design", *Knowledge Intensive CAD*, IFIP 5.2, KIC-4, Parma, Italy, No. 17.

[32] Summers, J., Lacroix, Z., Shah, J., 2002, "Case-Based Design Facilitated by the Design Exemplar", *Seventh International Conference on Artificial Intelligence in Design, '02*, ed. J. Gero, Kluwer Academic Press, Netherlands, pp. 453-76.

[33] Summers, J., Lacroix, Z., Shah, J., 2001, "Collaborative Mechanical Engineering Design: Representation and Decision Issues", *Le Travail Humain: Modeling Cooperative Activities in Design*, No. 8, Presses Universitaires de France, Paris, France.

[34] Bettig B., Shah, J., 2000, "Derivation of a Standard Set of Geometric Constraints for Parametric Modeling and Data Exchange", *Computer Aided Design*, **33 (1)**, pp 17-33.

[35] Lockett, H., Guenov, M., 2002, "An Intelligent Manufacturing Advisor for Casting and Injection-Molding Based on a Mid-Surface Approach", *ASME DETC2002*, CIE-34497, Montreal, Canada.

[36] Rangel, F., Shah, J., 2002, "Integration of Commercial CAD/CAM System With Custom CAPP Using ORBIX Middleware and CORBA Standard", *ASME DETC2002*, DAC-34069, Montreal, Canada.

[37] Nandakumar, S., 2000, "Classification, Parameterization, and Recognition of NC Features with Sculptured Surfaces", *MS Thesis*, Arizona State University, Tempe, AZ.

[38] Joshi, N., Dutta, D., 2002, "Feature Simplification in Surface Models for Efficient Finite Element Mesh Generation." *ASME DETC2002*, CIE-34494, Montreal, Canada.

