Design of Feature-oriented Database for Collaborative Product Development

S. H. Tang, Y. –S. Ma∗, G. Chen and J. Y. Chen

Department of Mechanical & Aerospace Engineering, Nanyang Technological University,
Singapore, 639798

Abstract: Concurrent and collaborative engineering (CCE) has become a norm. Feature-oriented objects are ideal to support web enabled collaborative engineering services. This paper describes the development of a feature-oriented database. Under the proposed unified feature modeling approach, generic feature representation, product and part representation and geometrical representation schema are investigated.

Keywords: Feature-oriented database; Concurrent and collaborative engineering; Collaborative product development

1. Introduction

Information sharing is the prerequisite for the implementation of concurrent and collaborative engineering (CCE). Currently, almost all existing CAx applications, including computer-based CAx applications, web portals, groupware tools and product data management (PDM) systems, use files as their repositories. File-based approach has the disadvantages of data redundancy, storage waste and potential conflicts [1]. Such design is not adequate for web-based CCE environment. It can be appreciated that, instead of managing the information via each application system in the separated data format, a database management system (DBMS) can be used to manage all the product information concurrently, and at the same time in a consistent manner in order to eliminate the duplicated data. A DBMS can also provide multiple users shared access to databases and the mechanisms to ensure the security and integrity of the stored data.

Some research work has been carried out in supporting CCE with DBMS. CAD*I project was among the first to use DBMS to realize the data exchange among different CAD systems [2]. Similar research work includes [3], [4]. However, so far, only geometric data can be managed in the databases. This means high-level feature information (semantic information) is lost. Therefore, it can not support complete information integration. To represent high-level feature information in database, Hoffman et al. [5] proposed the concept of product master model to integrate CAD systems with downstream applications for different feature views in the product life cycle. Wang, et al. [6] put forward a collaborative feature-based design system to integrate different CAx systems with database support. However, these proposed databases lack of geometrical engine to support model validation on the server side. Kim et al. [7] describes an interface (OpenDIS) for the integration of a geometrical modeling kernel (OpenCascade) and a STEP database (ObjectStore). However, STEP cannot fully cover information for different CAx applications, particularly for feature-based design.

In the previous work [8] a four-layer information integration infrastructure was proposed and a feature-oriented database was design. Ideally, it will enable information sharing among CAx applications by using the unified feature model in the entire product model, and allows the manipulation of application-specific information with sub-models. However, the geometrical representation adopted in [8] is B-rep, which can only support history-dependent model evaluation. History-dependent model evaluation has the disadvantages of high computation cost and large storage space [9]. Therefore, in this paper, the design is enhanced by adopting a higher-level cellular model on the basis of B-rep to support history-independent feature model evaluation.

2. Cellular Model
Cellular model represents a part as a connected set of volumetric quasi-disjoint cells [9]. By cellular decomposition of space, cells are never volumetrically overlapped. As each cell lies either entirely inside or outside a shape extend, a feature shape can be represented explicitly as one cell or a set of connected cells in the part. The cellular model-based geometrical representation schema adopted in this research is shown in Fig. 1. Basically, there are three types of topological entities for cellular topology, which are CELL, CSHELL and CFACE. CELL has two subtypes, namely CELL2D and CELL3D. A CELL2D contains a list of CFACES, each of which point to faces that are double-sided and both-outside. A CELL3D contains a list of CSHELLs. A CSHELL represents a connected set of CFACES that bound the 3D region of the cell. A CELL is attached to the normal ACIS topology in the LUMP level which represents a bounded, connected region in space, whether the set is 3D, 2D, 1D, or a combination of dimensions. Each CFACE has a pointer to a face in the lump and use it in FORWARD or REVERSE sense. For detail of history-dependent feature model evaluation on the basis of cellular model, please refer to [8].

**Figure 1.** Cellular Topology

### 3. Schema Definition for Proposed Database

On the basis of cellular model and mapping mechanism described in [9], a feature-oriented database is designed.

#### 3.1 Geometrical Representation

A partial cellular topology-based feature-oriented database schema is created as shown in Fig. 2. In the schema definition, (1) those attributes with suffix “_id” (but without “REF”) represent object identifiers (OIDs), which are the globally unique and immutable object identifier generated by DBMS and allow the corresponding row object to be referred to from other objects; (2) those attributes with “_id (REF)” are a kind of built-in data type provided by DBMS which encapsulates a reference to a row object of a specified object type; (3) an arrow represents REF relationship between object types, e.g. attribute edge_id in the COEDGE table, which has the REF data type and is used as a reference pointing to the edge object in the edge table; and (4) abbreviation ‘F’ represents “the first”, ‘N’ represents “the next” and ‘P’ represents “the previous”.

**Figure 2.** Partial Database Schemas for Geometrical Representation.
aggregate data type, the following two methods will be adopted on the basis of mapping mechanisms.

For ordered list which contains topological-related entities (e.g. a shell contains a list of faces which are topologically related), we follow the way of ACIS native data structure. This is realized by defining link relations in the object data structure, as illustrated in Fig. 3. In the list owner entity object table, only the first member of the list will be recorded by the entity’s ID, which can uniquely identify the first list member in the list member entity object table. Then in the list member entity object table, the next list member in the list will be explicitly recorded by its ID, which is used to identify the next list member. With such a data structure, each member of the list can be identified. Note that the last list member in a list will have a NULL next_entity_id pointer.

For unordered list (e.g. a feature contains a list of constraints), the schema shown in Fig. 4 is designed to collect each member of a list from the target object table. Such a list shall be defined as REF data type with name list_id which refers to list object in the entity_list object table by list_id. A nested table called id_list stores all the list members’ ids in the nested table. Within the nested table, entity_type is used as a vector to decide from which object table we can get the list members. Entity_id uniquely identify entities from entity table. An implicit system generates nested_table_id, and correlates the parent row object with the row objects in the nested table.

5. A feature has feature_id, feature_name, part_id and domain as its attributes. The feature_id attribute is an OID, which can uniquely identify a feature object in database. Feature_name combined with topological entity name, provide basic indexing for solving persistent naming problem during feature model re-evaluation. Part_id specifies which part a particular feature belongs to. Domain has enumeration data type, which can be design, manufacturing, CAE and so on. A feature also contains a list of feature elements, a list of constraints and a list of parameters.

![Figure 3. Linked List for Ordered List.](image)

For unordered list (e.g. a feature contains a list of constraints), the schema shown in Fig. 4 is designed to collect each member of a list from the target object table. Such a list shall be defined as REF data type with name list_id which refers to list object in the entity_list object table by list_id. A nested table called id_list stores all the list members’ ids in the nested table. Within the nested table, entity_type is used as a vector to decide from which object table we can get the list members. Entity_id uniquely identify entities from entity table. An implicit system generates nested_table_id, and correlates the parent row object with the row objects in the nested table.

![Figure 4. Generic Schema for Non-ordered List.](image)

3.2 Generic Feature Representation in Database

On the basis of the previous work [8], a generic feature representation in database can be expressed as shown in Fig.
specific feature instance, each list element will be treated as an attribute of the feature.

3.3 Product and Part Representation in Database

Given the geometry and generic feature representation schema, product representation in a feature-oriented database can be created as shown in Fig. 6. In this work, we only focus on the original feature representation of the product and the related parts. Other product-level and part-level related information is not addressed completely here.

![Figure 6. Product Representation in Database.](image)

A product is identified by product_id. A product (the top assembly) may contain a number of subassemblies, parts or components which are assembled with some assembly features. All the assembly features within a module are listed in its assembly_feature_list, which is defined for the top assembly and each sub-assembly of the product. For each assembly feature, a list of part_ids (or module_ids) is used to identify the related member parts (modules) in the part (module) table. An assembly may have new level sub-assemblies which could be referred by subassembly_feature_id in the assembly feature table. In the product, subassembly, and part table, design_feature_list, manufacturing_feature_list and other_application_feature_list are stored, which are used to organize feature model for different views at different levels. For example, all design features of a modular subassembly are stored in the associated design feature table. By feature_id and feature_type, various types of design features can be identified from the specific design feature tables. At the geometric entity level, different kinds of features are built on the basis of the cellular model.

3.4 Integration Solid Modeler with Database

The solid modeler has been tightly integrated in four layers in order to manage product and process information (see Fig. 7). First, its API functions are called constantly which are encapsulated within the feature manipulation methods during the collaboration sessions between the end users and the application server. Second, all the geometrical entities are manipulated and their run-time consistency maintained through the solid modeler’s implicit runtime data structure module. Third, it also provides runtime functional support directly to the end users via commands dynamically. Fourth, the solid modeler has also to support the repository operations via the DB manager.

![Figure 7. Integration of Modeler with Database.](image)

In the proposed architecture of the web-based feature modeling system [8], database manager (DB manager) is responsible for managing the geometrical entities via the solid modeler runtime model and manipulating the data elements to be stored and extracted in the database for different applications. With the support of a solid modeler, the database manager can provide data manipulation functions such as save, restore and validate functions. These functions are fundamental to support different applications.

4. Case Study

In the prototyped system, the feature-oriented product database has been established on the basis of proposed database schemas. For database server, ORACLE 9i, an object-relational database is adopted. ACIS, solid modeler has been tightly integrated with the database. In this section, a simple part is used to illustrate how feature information is represented in the proposed feature-oriented database.

Fig. 8 illustrates the creation of a simple part with two features, namely a base_block feature and a through_slot feature. The base_block feature is created by two diagonal coordinate points, which are derived from the parameters of
the block, i.e. length, width, height and its position point. The through_slot is positioned by specifying three coplanar constraints. The first constraint is the start face of slot that coplanes with front face of base_block. The second constraint is the end face of the through_slot which is coplanar with the back face of base_block. Then, the third is the top face of the feature that is supposed to be aligned with the top plane of the base block. Other parameters such as the length, width and depth of the slot are applied as dimensional constraints. Finally, the part is created.

Figure 8. A Case Part.

Figure 9. Cellular Model of the Case Part.

The cellular model of the example part is shown in Fig. 9. There are two cells (each one contains a cshell) in the cellular model of the part; one for the base_block and the other for the through_slot. Due to the overlapping of the two cells, three double-sided faces are generated, namely, F1, F2 and F3. Each double-sided face has two corresponding cell faces, one contributes to the cell of the base_block and the other to the cell of the slot. Note that, the cellular model of the example part (built with non-regular Boolean operation) keeps three additional faces (shaded faces with names F10, F11 and F12) and two more edges (E24 and E25) shown in Fig. 9 in comparison with the traditional B-rep final part geometry obtained by regular Boolean operation as shown in Fig. 10. These extra elements help to maintain the explicit feature shape in the part model. As they have the characteristic of not-on-boundary, the B-rep evaluation of cellular model can be easily carried out by doing boundary detection and removing entities that are not on the boundary. Note that in Figs. 9 and 10, only faces, edges, vertices and two corner points are labeled; other geometrical information (e.g. co-edges, loops) is not included.

Figure 10. The Final B-rep Part Geometry.

Based on the proposed database schema, such a part can be represented in database as shown in Fig. 11. In the part table, two design features, namely, base_block feature and slot feature are recorded with IDs. By block_ID, the block feature can be recognized from block feature table. All attributes of a block feature are stored in block feature table. Among all the parameters, two positions, which are defined by two vertices (V15 and V10), are used to fix the position and orientation of block. Each vertex can be identified by vertex_ID in the vertex point table. Vertex point table contains all the vertex of the example part. All the feature elements (cell list, face list, edge list and vertex list) of the block feature are stored as a list of feature labels identified by label_IDs. Due to the space limit, only face elements (B_L0–B_La) are taken as examples to explain how feature labels can be used to get low-level feature elements. Other feature elements works in the same way. By the label_ID and the corresponding face_ID stored in feature label table, the face elements (e.g. F0, F4, F5, F6, F7, F8, F9) can be recognized in the face table. By using the slot_ID, the slot feature can be identified from the through_slot feature table. All the attributes of through_slot feature are stored in the feature table.

The position field contains a vertex_ID (V4) which uniquely identify a vertex point in the vertex point table. All the face elements of the slot feature are stored as a list of
feature labels ($S_L$-$S_{L_0}$). In feature label object table, by the label ID and the corresponding face ID, all the face elements (e.g. $F_0$, $F_1$, $F_2$, $F_3$, $F_4$, $F_5$, $F_6$) can be recognized from the face table.

Figure 11. Database Representation of the Case Part.

In the slot feature table, constraints are stored in the constraint list. By the constraint type and constraint ID, different kinds of constraints can be identified from various constraint tables, e.g. coplanar constraint and distance constraint tables. In this case, coplanar constraint with IDs of $C_0$, $C_1$, and $C_2$ are stored in the coplanar constraint table. For coplanar constraint with ID $C_0$, the constrained feature elements (constrained_entity) can be identified by feature and element names. Similarly, the referenced feature elements (referenced_entity) can be identified by feature name, element name. For $C_0$, the feature name is through_slot1. The element name is $F_{0_3}$, which point to the face in face table. Coplanar constraint with ID $C_1$ and $C_2$ are processed in the same way. Note that in this example, geometrical and topological entities such as shell, face, loop, coedge are stored across different tables (see Fig. 2). In this way, low-level geometrical data and high-level feature information can be represented in the feature-oriented database.

5. Conclusions

In this paper, the design of a feature-oriented database is enhanced on the basis of cellular topology. Detail geometrical representation, feature representation and product representation are investigated. The integration of solid modeler with feature-oriented database is described. Based on the case study and working prototype system, it can be concluded that information sharing among different applications and Web enabled engineering collaboration can be realized on the basis of feature-oriented database.

References


