A Fine-grain and Feature-oriented Product Database for Collaborative Engineering

Y.–S. Ma, S.–H. Tang, and G. Chen

School of Mechanical and Aerospace Engineering
Nanyang Technological University, Singapore

Abstract

Traditionally, product databases are either purely geometric or meta-linked to CAD files. The first type lacks feature semantics and hence is too rigid for collaborative engineering. The second type is dependent on CAD files which are system sensitive and has too large information grain size that makes information sharing and engineering collaboration difficult. This chapter introduces a fine-grain and feature-oriented product database design. It is ideal to support Web-enabled collaborative engineering services. For this purpose, a four-layer information integration infrastructure is proposed. A solid modeler is incorporated to provide low-level geometrical modeling services. The novelty of this research includes three aspects: (1) a generic feature definition for different applications in the form of EXPRESS-schemas; (2) the integration of a solid modeler with feature-oriented database by mapping from EXPRESS-defined feature model to the runtime solid modeler data structure as well as to the targeted database schema; and (3) Modeler-based generic algorithms for feature validation and manipulation via the database. A modeler-supported history-independent approach is developed for feature model re-evaluation.

Keywords: Product database; Collaborative engineering; Feature-based modeling

6.1 Introduction

Due to the stiff competition and rapid changes of globalization, shortening time-to-market has become the critical success factor for many companies [1, 2]. As a result, concurrent and collaborative engineering (CCE) has become a norm. CCE has been recognized as the systematic approach to achieve the integrated, concurrent design of products and their related processes, including manufacturing and support [3], via collaborations across virtual project teams of different business partners.

In a CCE environment, many engineers with diverse skills, expertise, temperament and personalities are responsible for different tasks. The vast amount of knowledge and information involved in product development is certainly more than any individual can manage. Many computer-aided software tools have been incorporated into the product development process, which include Computer-Aided Design (CAD), Computer-Aided Process Planning (CAPP), Computer-Aided Engineering (CAE), and Computer-Aided Manufacturing (CAM) tools. However, information sharing among these applications has not been very well handled so far. Currently, almost all the existing CAx applications, which include individual installations, project Web portals, groupware tools and PDM (product data management) systems, are based on files as their repositories. File-based approach has large information grain-size that results in data redundancy, storage space waste and potential conflicts [4]. Therefore, such design is no longer adequate for web-based CCE environment. It can be appreciated that, instead of managing the information via each application system in the separated data formats, 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 shared user-access to databases and the mechanisms to ensure the security and integrity of the stored data.

Some research work has been carried out in product DBMS (database management system). CAD*I, a research project by ESPRIT (European strategic program for research and development in information technology) was among the first to use DBMS to realize the data exchange among different CAD systems [5]. Similar research work includes [6], [7] and [8]. However, in these product databases, only geometric data can be managed. This means high-level feature information (semantic information) is lost. Therefore, it cannot support complete information integration.

Currently, most of the CAx systems are feature-based because features are a very useful data structure that associates engineering semantics with tedious geometrical data entities. Therefore, feature information must be represented such that engineering meaning is fully shared among CAx applications. To represent high-level feature information in database, Hoffman *et al.* proposed the concept of product master model to integrate CAD systems with downstream applications for different feature views in the product life cycle [9]. Wang, *et al.* [10, 11] put forward a collaborative feature-based design system to integrate different CAx systems with database support. However, these proposed databases lack geometrical engine to support model validation.

A geometrical modeling kernel, which is also referred to as a modeling engine, provides lower-level geometrical modeling service. Therefore, it can be integrated with database to support feature management operations, such as saving, restoring and updating, and hence product model integrity and consistency can be maintained. In the previous work [12, 13], a four-layer information integration infrastructure is proposed based on the architecture of a feature-oriented database. Ideally, it will enable information sharing among CAx applications by using the unified feature model [14] in the EPM (Entire Product Model), and allows the manipulation of application-specific information with sub-models. However, the method to provide low-level geometrical modeling services remains as a major question for research.

Martino et al. [15] proposed an intermediate geometry modeler to integrate design and other engineering processes with a combined approach of "design-by-feature" and "feature recognition". Bidarra [16, 17] and Bronsvoort [18, 19] proposed a semantic feature model by incorporating ACIS into webSPIFF, a web-based collaborative system. However, the above-mentioned research has little discussion on the integration of solid modeler with database, and it is not clear whether they have managed product data in files or with a database. Kim et al. [20] described an interface (OpenDIS) for the integration of a geometrical modeling kernel (OpenCascade) and a STEP database (ObjectStore). However, their work cannot ensure full information integration because STEP cannot cover feature information for different feature-based CAx applications.

Traditionally, feature information cannot be exchanged among different applications. More recently, researchers, such as Bhandarkar *et al*. [21], Dereli *et al*. [22] and Fu *et al*. [23], proposed different algorithms to identify useful feature information from the exchanged part models. Although feature extraction [24] and identification can partially recognize some feature information, information loss still occurs because these approaches depend on pure geometric data. For example, feature relationships (constraints) cannot be recovered from the geometric data model.

In order to enable higher-level feature information sharing among different applications, many researchers [25, 26, 27] proposed to use design information as the input and derive downstream application feature models by feature conversion. However, their works support only one-way link which means they can only convert from design features to other application features. In [28, 29], a multi-view feature modeling approach that can support multi-way feature conversion by feature links, is proposed. Separately, an "associative feature" definition was developed in [30, 31] for establishing built-in links among related geometric entities of an application-specific and multi-facet feature while self-validation methods were defined for keeping feature validation and consistency. Compared with one-way feature conversion approach, these multi-facet feature representations are promising for supporting multi-view product modeling.

The concept of unified feature model was first proposed by Geelink *et al.* [32]. The interactive definitions for design and process planning features were focused. However, the constraints defined were limited within one application feature model. Therefore, different application views could not be integrated in their model. Chen *et al.* [14] proposed a new unified feature modeling scheme by introducing inter-application links for higher-level feature information sharing among different CAx applications. The unified feature model is essentially a generic semantic feature model for different CAx applications covering three-level relations among geometric and non-geometric entities. The unified feature model includes a knowledge-based model by incorporating rules and the necessary reasoning functions [33, 34].

This chapter focuses on the investigation of mechanisms to integrate a solid modeler with a feature-oriented database, such that multi-application information sharing can be realized over the Web. This chapter consists of seven sections. After this introduction, Section 6.2 gives a generic definition of features with the consideration of unification of applications. Section 6.3 investigates the mapping mechanisms between the proposed feature type, consisting of properties and methods, and a solid modeler data structures. Section 6.4 explores the integration of the solid modeler and database with key algorithms, e.g. feature validation, constraint solving. Section 6.5 describes the method for solid modeler-supported feature model evaluation. A case study is presented in Section 6.6. Section 6.7 gives the conclusions.

6.2 Generic feature model

To consider integrating a solid modeler with the feature-oriented database, the mapping method between the database schemas and the feature definitions based on the solid modeler entities is critical. A unified feature model allows different applications to define different features with a set of well-defined generic types [14]. It is essential that each feature type has well-defined semantics [16]. The semantic attributes specified in each feature definition have to be associated with the structured elements of the given feature type. Such elements include feature shape representation with parameters, constraints that all feature instances should satisfy, and the non-geometric attributes to be used for embedded semantic properties, such as classifications, names, labels, and relations. All types of constraints are used for capturing design intent in the context of a product model. A generic feature representation schema is described in Fig. 6.1. Note that the original information model is described in EXPRESS-G. Details for the convention of EXPRESS-G are shown in Fig. 6.2 [35].

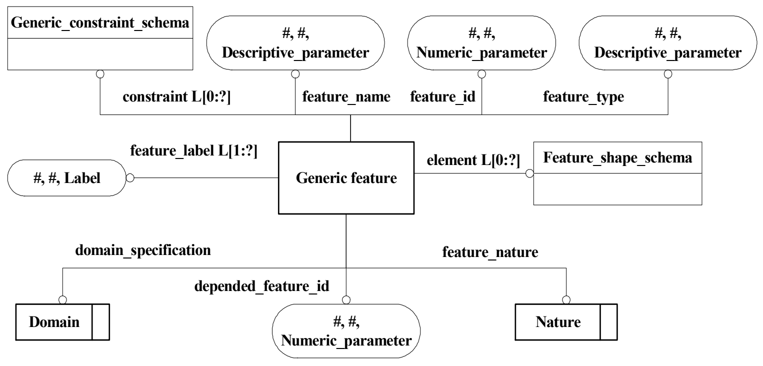


Fig. 6.1. Generic feature representation schema

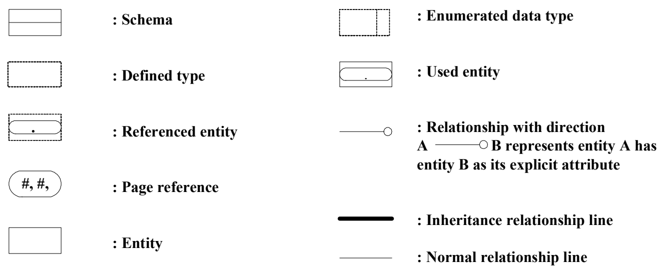

: Schema
: Defined type
: Referenced entity
#, #,
: Page reference
: Entity
: Enumerated data type
: Used entity
: Relationship with direction
A B represents entity A has entity B as its explicit attribute
: Inheritance relationship line
: Normal relationship line

Fig. 6.2. Convention of EXPRESS-G

6.2.1 Feature shape representation

To represent the shape of a feature means defining feature geometrical and topological constraints or relations with parameters and associating these parameters with feature manipulation (creation, modification and deletion) functions. The parameters are used to provide user interfaces to create and modify features in the modeling operations.

6.2.2 Constraint definition

Constraints must be explicitly defined in the feature model to specify relationships among features, geometric or topological entities. Such constrints provide invariant characteristics of a feature type in the product model. Constraints may have various types (e.g. geometric constraints, tolerance constraints and others). In generic feature definition, constraints are regarded as attributes attached to a set of associated entities, e.g. geometric and non-geometric entities or even features. Although different types of constraints may have different attributes, they fall into a few common types, which can be generalized as shown in Fig. 6.3.

Constraint_ID: It is the identifier of a constraint instance.

Constraint_name: It specifies the name of a constraint instance.

Owner_ID: It uniquely identifies which feature a constraint belongs to.

Constraint_expression: It represents the relationship between the constrained elements and reference elements.

Constrained_entity_ID list: It is used to specify a list of constrained entities with reference to the referenced entities.

Referenced_entity_ID list: It can be used to uniquely identify other related reference entities.

Constraint_strength: It has an enumeration data type, which may include several levels, such as *required, strong, medium or weak*. It represents the extent that the constraint needs to be imposed when constraints conflict with each other.

Constraint_sense: It is used to specify the direction between constrained entities and referenced entities. It has the select data type which maybe *directed* and *undirected.* A constraint is directed if all members of a set or list of constrained entities are constrained with respect to one or more referenced entities. A constraint is undirected if there are no referenced entities and the constraint is required to hold between all possible pairs of a set of constrained entities. Stated differently, in the undirected constraint, there is no difference between constrained entities and referenced entities. For example, if a directed constraint is applied to two lines (line1 and line2), which requires line2 to be parallel with reference to line1, it implies that line1 existed in the model before line2 was created. The corresponding undirected constraint would simply assert that line1 and line2 are parallel, with no implied precedence in their order of creation.

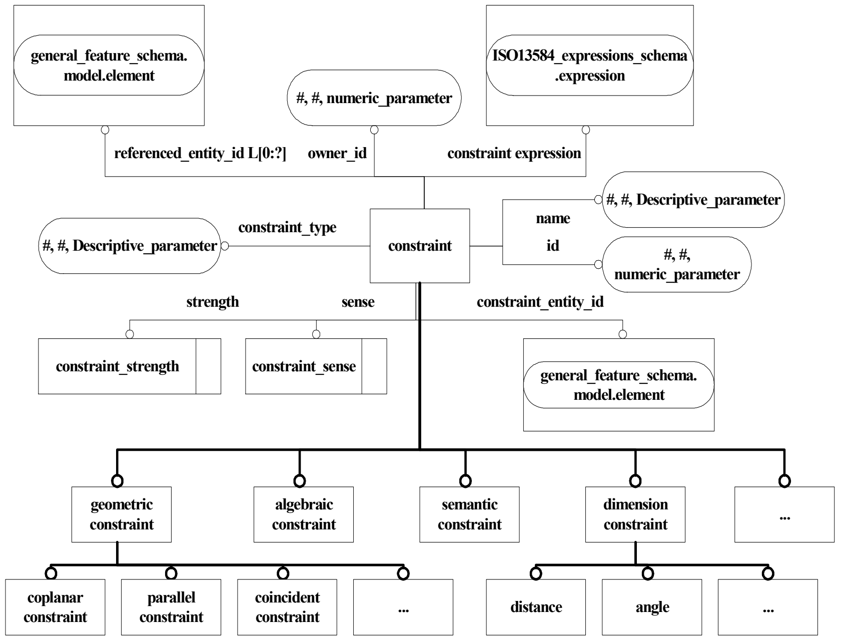


Fig. 6.3. Constraint representation schema

Constraint solving functions: They are responsible for solving constraint according to constraint types.

Other manipulation functions: These functions may include attributes access functions, behavior control functions, etc.

6.2.3 Other feature properties

Other feature properties can be defined as follows:

General feature attributes- *Feature_name* and *feature_id*

General feature attributes such as *feature_name* and *feature_id* shall be realized with the instantiation of a specific feature according to the *application_specific* feature definition. These attributes are necessary when searching for the relevant feature properties during feature modeling operations.

Depended_feature_id_list

To maintain feature relationship, *depended_feature* shall be explicitly defined in feature definition. Feature dependency relation definition is described by Biddara [16, 17] as "feature *f1* directly depends on feature *f2* whenever *f1* is attached, positioned or, in some other way, constrained relative to *f2*". *Depended_feature_id_list* plays an important role in maintaining feature dependency graph, and furthermore, feature relations during feature modeling operations.

Feature label

A feature label is attached as an attribute to every face of a particular feature instance. In a feature, its member face labels are defined as a list of strings in the definition, to record feature face elements. Then the face corresponding to the label is referred to as the owner.

Domain specification

Domain specification has the ENUMERATION data type, which represents the application scope such as *design*, *manufacturing*, *assembly* and others. By specifying the different domains, multi-views can be supported with certain filtering and synchronizing mechanisms.

Nature

The nature of a feature also has ENUMERATION data type. It could be either positive or negative. A positive value means the instances of the feature are created by adding material. A negative value means forming a feature instance is realized by subtracting material.

6.2.4 Member Functions

Four groups of member functions are required to support the generic feature class. *Attribute access functions* shall be defined to manage a feature's attributes. Some functions are common to all types of features, e.g. *backup()*. Others are feature-specific such as *findOwner()*, *findConstraint()*, *getParameter()*, *setParameter()*, etc. Object technology with a proper polymorphism design can be applied well here.

Modeling operation functions (e.g. *splitOwner()*, *mergeOwner()*) are used to control the behaviors of feature during a modeling operation, e.g. splitting, merging, or translation.

Feature evaluation and validation functions are responsible for feature model modification. Feature validation functions are used to validate feature geometry and solving constraints after each feature modeling operation. These functions will be discussed in detail in section 6.4.

In order to persistently manage product and process information, which includes feature information, geometrical data and other information, *saving and restoring functions* of the database, which are the interactions between the run-time feature model and the database, must be defined in individual feature classes because these functions have to organize information for different applications according to the functional requirements. Details will be explained in section 6.4.

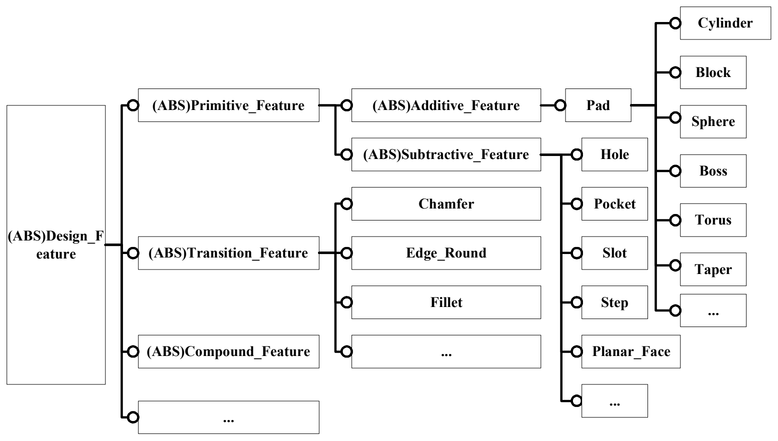


Fig. 6.4. Design feature representation schema

6.2.5 Application-specific Feature Model

Application-specific feature model can be defined on the basis of generic feature model. As shown in Fig. 6.4, the design feature type has three subtypes: primitive feature, transition feature and compound feature. The primitive feature type is separated into two subtypes, additive and subtractive features. Additive feature is represented as "pad", which covers all instance features formed by adding material such as cylinder, taper, sphere, boss, block, torus and so on. Subtractive feature type represents all features such as hole, pocket, and slot that are formed by subtracting material. The transition feature type includes chamfer, edge_round and fillet, which are always associated with other primitive features. The compound feature type is a union of several primitive features. For each specific design feature type, it has predefined explicit geometry, topology, parameterization and constraints specifications. For example, a design feature slot can be defined as shown in Fig. 6.5.

6.3 Mapping mechanisms

To provide lower-level geometrical modeling services, a geometrical modeling kernel is required. In this work, ACIS, a commercial package, is incorporated into the proposed system. An EXPRESS-defined and extended STEP feature model, which includes geometrical and generic feature representation schemas, is mapped to the data representation schemas in ACIS such that the proposed system will have the required fine grain functionality. On the other hand, this feature model would also need to be mapped to the target database schema so that it can be interfaced with a consistent repository.

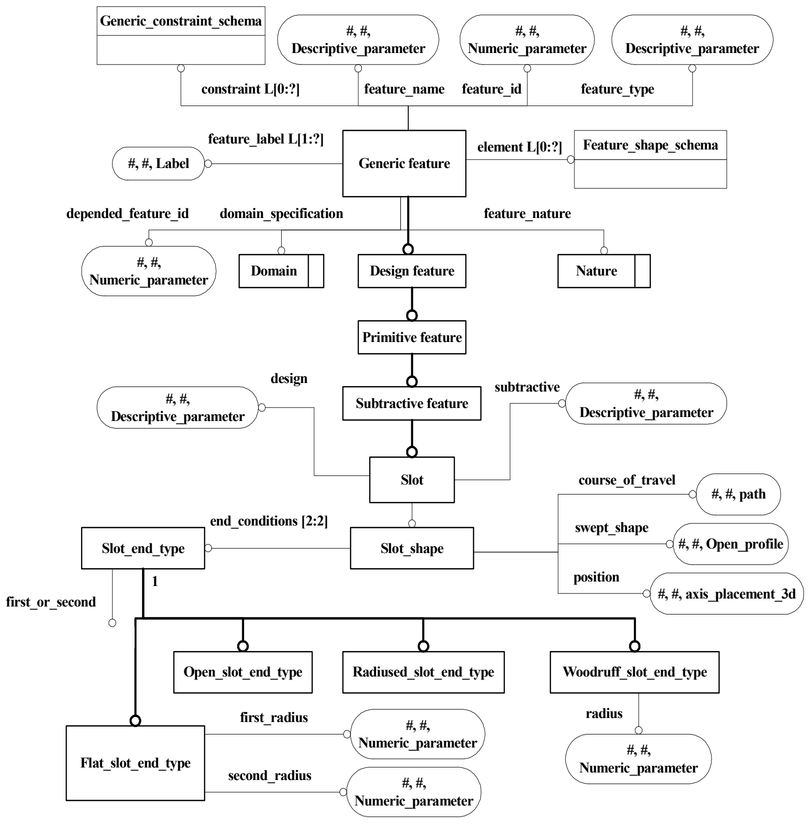


Fig. 6.5. Slot feature definition in EXPRESS-G

6.3.1 Mapping from extended EXPRESS model to ACIS workform format

6.3.1.1 Geometry mapping

In this research, in order to explicitly maintain feature shape and associative relations in the product model, a cellular model is adopted. Cellular model represents a part as a connected set of volumetric quasi-disjoint cells [36]. By cellular decomposition of space, cells are never volumetrically overlapped. As each cell lies either entirely inside or outside a shape volume, 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. 6.6. 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 *CFACE*s, each of which points to faces that are double-sided and both-outside. A *CELL3D* contains a list of *CSHELL*s. A *CSHELL* represents a connected set of *CFACE*s 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.

As cellular model is directly supported in an ACIS, cellular husk is adopted. Therefore, geometry mapping is one-to-one straight forward.

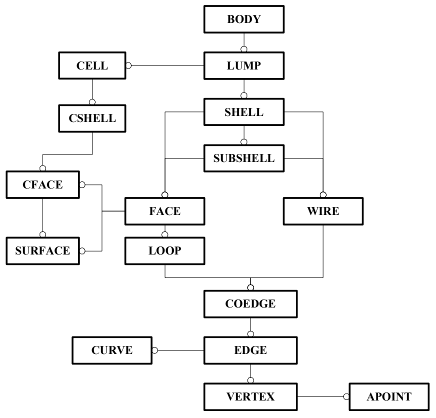


Fig. 6.6. Partial geometrical representation schema according to cellular topology [36]

6.3.1.2 Generic feature definition under ACIS framework

ACIS provides ENTITY-ATTRIBUTE architecture [36], under which we can specify user-defined attributes (features, constraints or others). The following rules are developed and used by the authors for defining features, constraints and other attributes in ACIS:

Use simple attributes to represent properties such as the material of a body or color of a face.

Use complex attributes to represent properties such as features, dimensions, tolerance, or constraints.

Use bridging attributes to link an ENTITY with some application-specific and parametric variables, such as dimensions.

Use instruction attributes placed on entities to force certain behavior.

Attributes of features and constraints may have various data types, e.g. string, integer or ENTITY pointer.

Aggregating data type has been defined as ENTITY_LIST. The ENTITY_LIST is a variable length associative array of ENTITY pointers and provides common functions for the manipulation of its members, e.g. *add* ENTITY, *look up* ENTITY and *[]* operator for accessing list member by position.

Enumeration data type can be simulated by defining a string as the enumeration member or simply using an integer data type.

Selecting data type can be simulated by using an abstract class and defining specific types on top of the abstract class.

On the basis of the above proposed mapping rules, a generic feature definition is created as shown in Fig. 6.7.

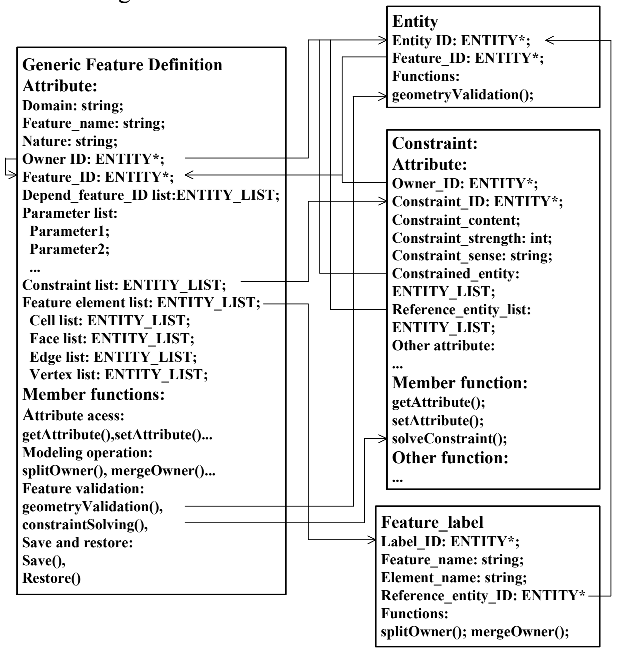


Fig. 6.7. Generic feature definition with ACIS entities

6.3.2 Database representation schema

According to the mapping mechanisms proposed in [12], a geometrical representation schema as well as generic feature representation schema in the database has been developed. For details, please refer to [12].

6.4 The Integration of the solid modeler and the database

The solid modeler has been tightly integrated in four layers in order to manage product and process information (see Fig. 6.8). 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.

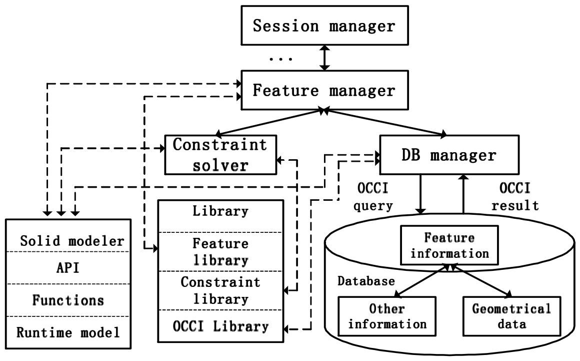


Fig. 6.8. Partial integration diagram of a solid modeler and the feature-oriented database

This chapter focuses on the forth layer. In the proposed architecture of the web-based feature modeling system [12], database (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. In the following sub-sections, feature validation methods together with the generic *save* and *restore* algorithms are explained. In order to manage the connection between the DB manager and the database during saving and restoring processes, OCCI (Oracle C++ Call Interface) [37] is adopted as the bridge (see Fig. 6.8).

6.4.1 Feature model re-evaluation and constraint solving

Once feature operations are specified via User Interfaces (UIs), the product model needs to be modified and updated. This process is achieved through feature evaluation. The geometrical model has to be managed to ensure the consistency. Here, the run-time product model should be generated via the integrated solid modeler and managed based on the database records. All feature evaluation operations call solid modeler APIs to realize the geometrical procedures while the rest of the functions are implemented separately. In this way, the bottom-level geometrical operations are readily looked after by the solid modeler; hence, the development effort is significantly reduced. Details of feature model re-evaluation will be explained in section 6.5.

Theoretically, feature validation functions include two kinds: those dealing with the geometry, and those dealing with constraints. With the incorporation of a solid modeler, geometry validation functions are not really necessary under the proposed design because the solid modeler is responsible for manipulating and validating feature geometry. On the other hand, constraint-solving functions need to call specific algorithms defined in the individual constraint sub-classes to solve different kinds of constraints according to their types. Globally, all the constraints are maintained by the Constraint Manager in a constraint graph for EPM (Entire Product Model), which contains sub-graphs for specific application views. Constraint manager solves constraints by calling the corresponding solvers according to different constraint types. For example, SkyBlue algorithm [38] can be used to solve local algebraic constraints in design domain; Degrees of Freedom analysis algorithm [39] can be used to solve geometrical constraints in design domain. If conflict of intra-application constraints occurs, local constraints solver can determine automatically which constraint should be satisfy first according to the value of *constraint_strengh*, which is an attribute of constraint defined in section 6.2. Inter-application constraints can also be solved under the control of constraint manager according to the value of *domain_strength*. For the definition of *domain_strength,* also refer to section 6.2. The value of *domain_strength,* which regulates priority sequence of different domains, can be predefined, or is set by an authorized user. Any conflict of inter-application constraints will be detected by constraint manager after which the constraints solver can trigger the corresponding applications to reevaluate the product model according to *domain_strength*. Only when all constraints are checked and feature geometry is validated, does feature validation finish.

6.4.2 *Save* algorithm

To elaborate, during the saving process, the solid modeler has to extract all the information from its runtime data structure and then save them into the database after a format conversion according to the mapping relations and the database mapping schema described in [12]. The *Save* algorithm can be expressed in the steps as follows (see Fig. 6.9):

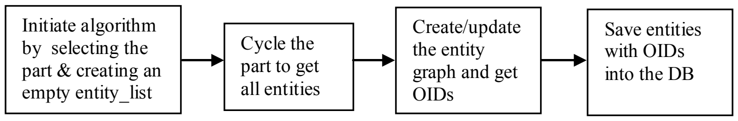

Initiate algorithm by selecting the part & creating an empty entity_list
Cycle the part to get all entities
Create/update the entity graph and get OIDs
Save entities with OIDs into the DB

Fig. 6.9. ***Save*** algorithm

- Select the part to be saved. Create an empty entity list and add the part attributes to be saved to the list;
- Cycle all entities (features, topological entities, such as solids, shells, faces, and geometrical entities, such as lines, planes, curves, and surfaces) from the part and add them to a graph map so that object pointers can be fixed as unique database Object Identifiers (OID). ACIS API functions, e.g. *api_get_xxxx()*, are used to get all saved ENTITIES;

- Use such object pointers to call *save* functions of the specific class (e.g. *point.save()*, *vertex.save()* or *feature.save()*) to save part data to the database.

6.4.3 Restore algorithm

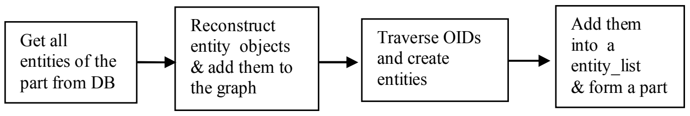

Get all entities of the part from DB
Reconstruct entity objects & add them to the graph
Traverse OIDs and create entities
Add them into a entity_list & form a part

Fig. 6.10. *Restore* algorithm

In a reverse way, the uploading process is triggered when the product model is being established during the session initiation from the database.
Restore algorithm has the following steps (see Fig. 6.10):

- All the entities of a part are retrieved from the database by searching their linked Object Identifiers (OIDs);
- Reconstruct new objects, e.g. features, geometrical entities, topological entities. Upon reconstruction, all the objects will be validated;
- Add all the entities to a newly generated object graph map;
- Convert these OIDs to genuine pointers;
- Create an entity list and add all the entities to the list to form the part. Validation, e.g. geometry and feature validation will be carried out during this procedure.

6.5 Feature model re-evaluation

6.5.1 Problems of historical-dependent system

For most parametric and history-based modeling systems, feature model is re-evaluated by re-executing whole or part of the model history. The disadvantages of this method are the high computational cost and the considerable amount of storage space [16]. Moreover, history-based model re-evaluation causes ambiguous feature semantics due to the static chronological feature creation order in the model history. This is illustrated in the example shown in Fig. 6.11. The simple part consists of a base block and a through hole. Later on, the designer wants to modify the part by adding another block and extending the depth of hole so that he can get the expected part model as shown in Fig. 6.11(b). However, sometimes unexpected modeling results as shown in Fig. 6.11(c) can be generated by the history-based reevaluation, because the feature creation order is *baseblock->hole->block*. In order to get the expected part model, the precedence order, in this example, should be changed to *baseblock->block->hole*. This semantic problem is caused by the static precedence order in the model history on which model re-evaluation is based. From this example, it is clear that the precedence relation among features should be dynamically maintained and updated after each modeling operation.

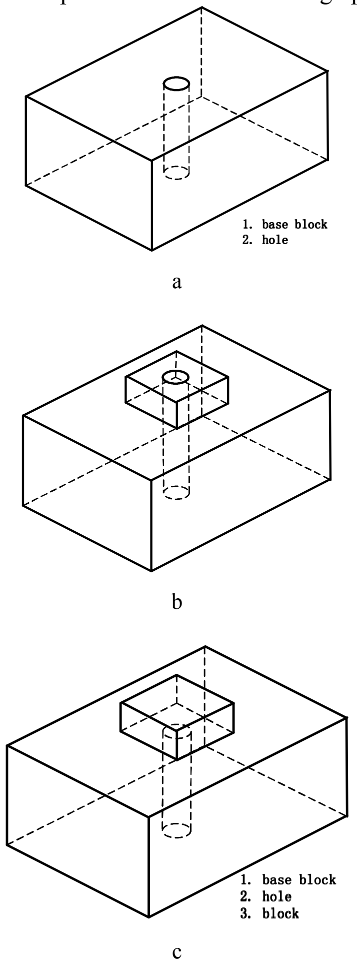


Fig. 6.11. Semantic problem for historical-dependent system a. example part at the initial state; b. expected result after modification; c. result of history-based re-evaluation after modification

6.5.2 Dynamically maintaining feature precedence order

In this work, feature precedence order is maintained dynamically based on a feature dependency graph. Relations between independent features can be determined by feature overlapping detection. Feature dependency relations are explicitly defined in the feature definition as explained in section 6.2. The following rules are proposed for feature precedence determination. Note that, explicit rules always overrule implicit rules during dynamic maintenance of the global precedence order of all features. Stated differently, the explicit rules will be first used to determine the precedence relation; while if the global precedence order cannot be uniquely generated, implicit rules will be then considered to get a unique one.

Rule 1 (explicit rule)

For two dependent features, if feature f_2 depends on feature f_1, then f_1 precedes f_2 [16].

It is easy for us to derive from rule 1 that:

For n dependent features, if:

$$f_1 \rightarrow f_2 \rightarrow f_3 \rightarrow \ldots \rightarrow f_n$$

Then, there exist:

$$O_1 < O_2 < O_3 < \ldots < O_n$$

where:

$f_i \rightarrow f_j$: represents feature dependency relation(e.g. $f_1 \rightarrow f_2$ means f_2 depends on f_1);

O_i : represents the precedence order of feature f_i .

$O_i < O_j$: represents the j^{th} feature is ordered after the i^{th} feature.

Rule 2 (explicit rule)

For a feature in the feature dependency graph, if it depends on two or more features, the precedence order of this feature comes after the latest feature it depends on (we call it latest depended feature or LDF).

Note that in the feature dependency graph, LDF is always the feature that has the longest length of path (LLP) from the root node of the graph among all depended features of a particular feature.

Path: a path in a graph is a walk whose nodes are all distinct;

Walk: a walk in a graph is a finite alternating sequence of nodes and edges between its starting node and ending node;

Length of path: the length of a path is the number of edges that form the path.

Rule 3 (implicit rule)

For a group of features that have random precedence order, the feature creation sequence will be used to determine their precedence relations.

The feature creation sequence is defined as an attribute attached to the feature instance to record the sequence of the feature among all features in the part.

Rule 4 (implicit rule)

For two independent features, if they do not overlap with each other, the precedence relation between them is determined by LLP of these two features. There exists:

$O_1 < O_2$ if $LLP_1 < LLP_2$

In the case of $LLP_1 = LLP_2$, the precedence order can be determined by rule 3.

Rule 5 (implicit rule)

For two independent features with same natures (both negative or both additive), if they overlap with each other, the precedence relation between them is random and should be determined by LLP of these two features. There exists:

$O_1 < O_2$ *if* $LLP_1 < LLP_2$

In the case of $LLP_1 = LLP_2$*, the precedence order can be determined by rule 3.*

Rule 6 (explicit rule)

For two independent features (f_1 *and* f_2*) with different natures, if the overlap of these two features is caused by some modeling operation of* f_2*, then feature* f_1 *precedes feature* f_2 *[16].*

Based on the above rules for feature precedence determination, after each modeling operation, the following algorithm shown below is used to dynamically maintain feature precedence relations.

- Find all the features of the part and add them to a graph map (unsorted).
- Partially sort the graph map according to the existing feature dependency graph. This is done by using the algorithm shown in Fig. 12 on the basis of rules 1 ~ 3.
- Sort the partially sorted graph with reference to the overlapping detection result based on rules 4 ~ 6.

In this way, a global feature precedence order can be updated dynamically.

```
(For i=1; i<n-1; i++)
{ (for j=i+1, j<n; j++)
   {if ($Px_j$>$Px_i$)
     {$X_m$= $X_i$;
      $X_i$=$X_j$;
      $X_j$=$X_m$;
     }
   }
}
```

Here:
X_i represents any feature in the feature set;
X_j represents depended feature of X_i;
Px_i represents the position of feature X_i in the feature map;

Fig. 6.12. Algorithm for precedence order generation [40]

6.5.3 History-independent feature model re-evaluation

First of all, re-evaluating the feature model requires that feature elements (cells, faces, edges and vertices) are correctly identified in the cellular model. This can be achieved by cellular entity owner list control.

6.5.3.1 Adding a New Feature Instance

This is carried out as follows:

- Create the shape of the new feature (one cell shape);
- Attach labels of the feature to each face of the feature instance; and
- Carry out Boolean operation (with the ‘non-regular’ option).
- During non-regular Boolean Union, intersection detection will be carried out for each cell (C_i) in the cellular model and the newly added feature cell (C). Upon cellular decomposition, the owner list of each cell and cell face should be controlled by the following rules [41]:
- The new cells that are in the intersection of C and C_i are assigned with an owner list that is the union of the owner lists of C and C_i;
- Other non-intersecting cells resulting from the decomposition get their owner lists which are the same as the original cells (either C or C_i);
- The new cell faces lying on the boundary of both C and C_i get the owner list that is the union of the owner lists of the overlapping cell faces from which it originates;
- The new cell faces lying on the boundary of either C or C_i inherit the owner list from their respective original cell faces;
- The remaining new cell faces get an empty owner list.

Fig. 6.13(a) illustrates the creation of a *slot* feature on the *base_block*. The shape of the *slot* is first created as a one-cell shape. Then *non-regular-Boolean Union* is carried out to create the cellular model of the part. During the operation, upon intersection analysis, cell decomposition is performed. On the basis of above rules for cell and cell face owner list control, the result of the modeling operation is shown in Fig. 6.13(b). Note that there are two cells in the cellular model. One is the original *base_block* cell (has *block* feature in its owner list). The other is a new cell generated by cell decomposition, namely the *slot* cell (which has *block* and *slot* in its owner list). Three double-side faces separate these two cells. Each double-side face has two corresponding cell faces (e.g. CF_8 and CF_9); one (CF_8) is for the *block* cell boundary, the other (CF_9) is for the *slot* cell boundary.

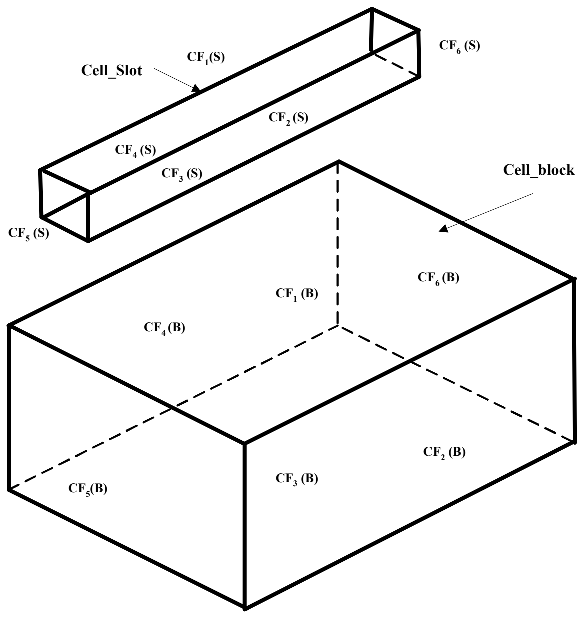
Note that CF_i represents i^{th} cell face; *S* represents *slot* feature; *B* indicates *block* feature; and *( )* indicates the labeled entity’s owner list.

6.5.3.2 Deleting a Feature Instance

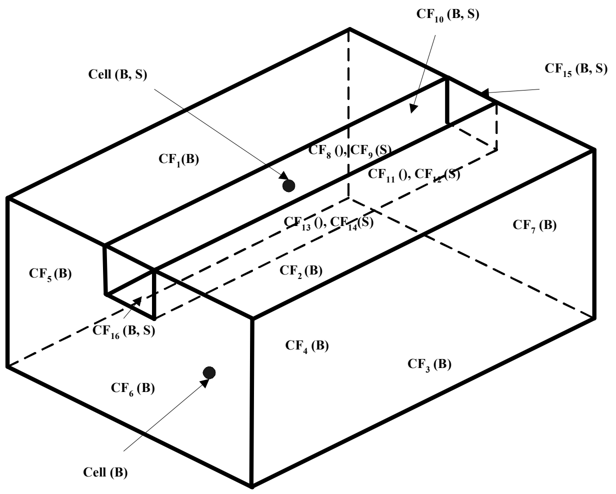
This is carried out as follows (assume no other feature depends on the feature to be deleted) [16]:

- Traverse through all the cells and cell faces to remove from their owner list the feature to be deleted;
- Remove all the cells which has empty owner list. This can be realized by removing all one-side faces bounding the cell;
- Merge adjacent cells which have the same owner list. This can be realized by removing all double-side faces that separate the two cells;

- Clean up the model by merging the adjacent faces that have the same geometry and whose cell faces have the same owner list.



a



b

Fig. 6.13. Creation of slot feature on the base block a. base block and slot shape; b. result of modeling operation

As shown in Fig. 6.14, to delete the slot feature from the cellular model, all cells and cell faces in the cellular model are traversed through to remove from their owner list the slot feature.

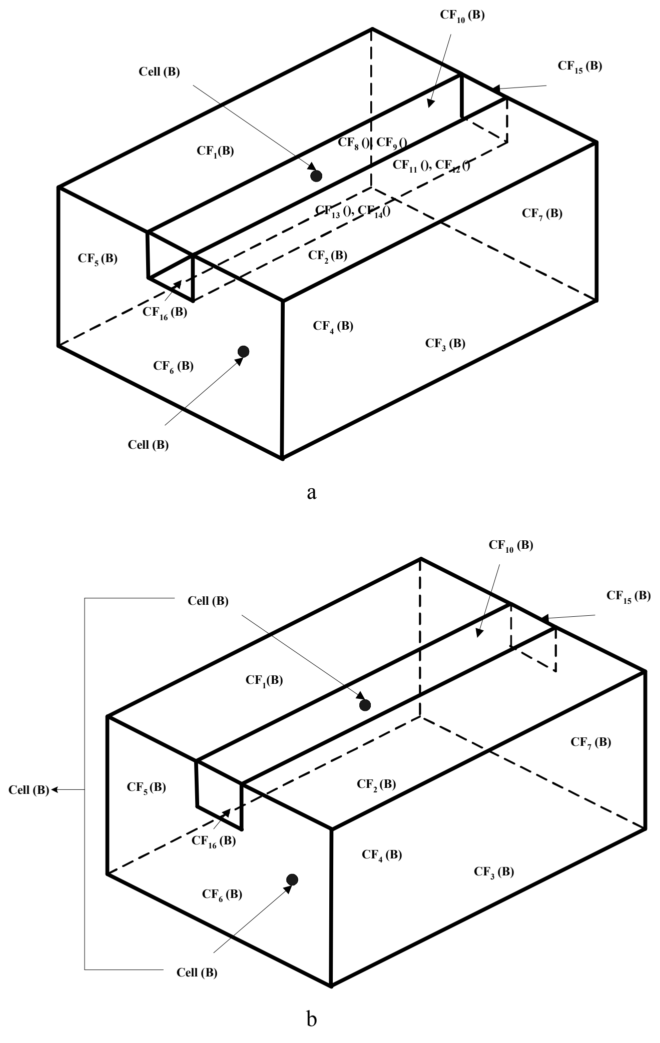
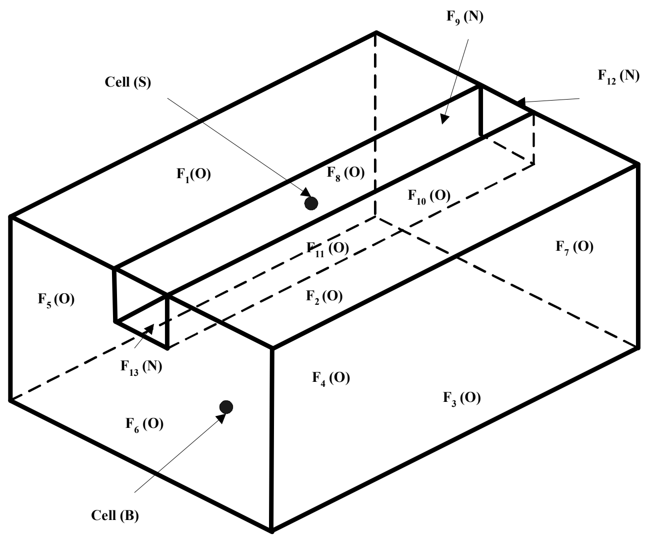
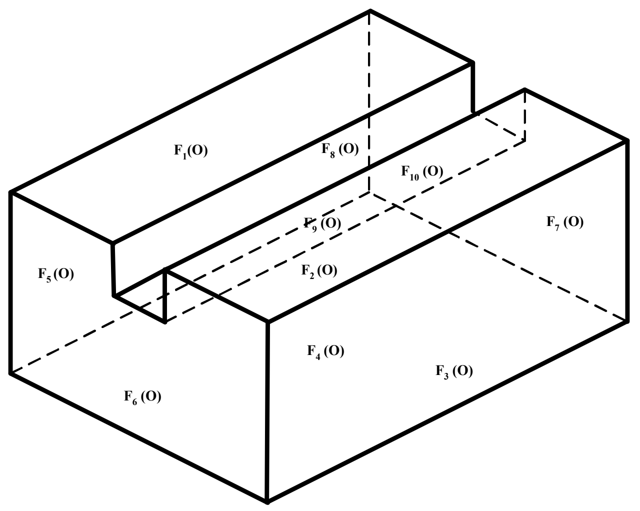


Fig. 6.14. Feature deletion a. remove slot from owner list of cell and cell face; b. merge two cells

The result is shown in Fig. 6.14(a). Then as two cells have the same owner list, the block feature, these two cells are merged by removing three double-side faces (the underlying faces of CF8 and CF9, CF11 and CF12, CF13 and CF14) that separate them. The result is shown in Fig. 6.14(b). Finally, adjacent faces that have the same geometry and same owner list are merged. This option carries out the merging of the underlying faces of CF5 and CF16, CF7 and CF15, as well as CF1, CF2 and CF10. The result is the same as block feature shown in Fig. 6.14(a) before creating the slot.



a



b

Fig. 6.15. B-Rep evaluation a. boundary detection; b. boundary evaluation

6.5.3.3 Modifying a Feature Instance

To modify a feature instance, we shall first check which features are really involved in the modeling operation. This checking is based on the feature dependency graph and the global feature precedence order. Next, features involved will be removed from the part model. Finally, features with modified properties will be added to the model.

6.5.3.4 B-rep Evaluation

As the cellular model of a part contains much more information than only the boundary of the final part, which means it also contains faces that are not on the boundary. Therefore, the B-rep evaluation of the cellular model requires boundary detection of every face in the cellular model. The following rules are used to carry out the boundary detection of faces in the cellular model:

- For each single-side face of a cell, the nature of the cell determines whether it is on-boundary or not. Additive nature of the cell means the face is *on-boundary*; subtractive nature of the cell means the face is *not-on-boundary*.
- For each double-side face of a cell, if the cell has a different nature with the partner cell that shares the same face, this double-side face is *on-boundary*; otherwise, it is *not-on-boundary*.

Note that the nature of a cell is determined by the nature of its last owner in the cell owner list. The sequence of cell owner list is dynamically maintained according to the unique feature precedence order, see section 6.5.2. On the basis of boundary detection of each face in the cellular model, the B-rep evaluation of the cellular model can be carried out in steps as follows:

- Walk through all the faces and find all the faces that are *not-on-boundary*;
- Remove all the cells and the faces that are *not-on-boundary*;
- Merge adjacent faces that have the same geometry.

Also taking the part shown in Fig. 6.13(b) as an example, on the basis of rules for boundary detection, the result of boundary detection is shown in Fig. 6.15(a). Then, the B-Rep evaluation of the cellular model can easily be realized by removing three faces (F_9, F_{12} and F_{13}) that are *not-on-boundary*. The result is shown in Fig. 6.15(b). Note that in Fig. 6.15, O represents *on-boundary*; N represents *not-on-boundary*.

6.6 A Case Study

The proposed feature-oriented database has been implemented coupled with a geometrical modeling kernel, ACIS. Design features and constraints have been defined and some example parts have been tested. Fig. 6.16 illustrates the creation of an example part which is made up of a *base_block,* a *vertical_support,* a *rib,* a *cylinder and* two *through_hole* features.

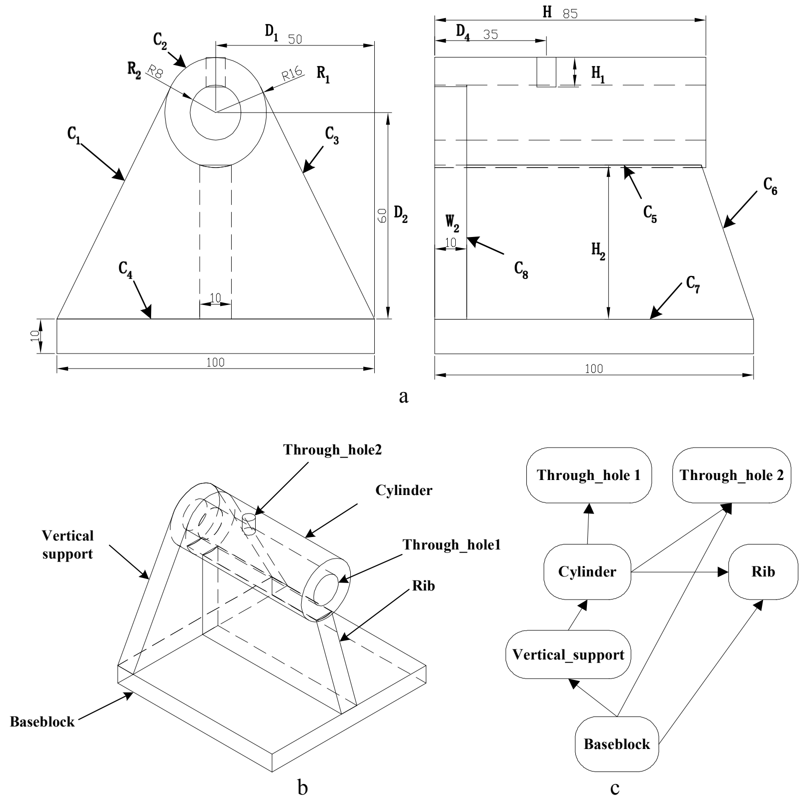


Fig. 6.16. A case part a. creation of the case part; b. modeling result; c. feature dependency graph

All the parameters and constraints are listed in Tab. 6.1. The precedence order of the part can be generated according to the rules described in section 6.5.2 as follows: *Base_block* → *vertical_support* → *cylinder* → *rib* → *through_hole 1* → *through_hole 2*.

If the designer wishes to add a *cylinder_boss* feature on the top of *cylinder* overlapping with the *hole2* as shown in Fig. 6.15, feature overlapping detection and semantic constraint checking (semantic constraints here refer to *through_hole2* must have both top and bottom face *not_on_boundary*) will be carried out. In this case, constraint conflict happens because the semantic constraint of the *through_hole2* cannot be satisfied if the current precedence relation is kept. Therefore, a message will be generated by the system to prompt the user on how to solve such problems (via changing the precedence order of those two features as shown in Tab. 6.2).

After modification, the feature precedence order of the part will be changed from: base_block→ vertical_support → cylinder → rib → through_hole1→

through_hole2→ boss to: base_block → vertical_support → cylinder → rib → through_hole1→ boss → through_hole2**.**

Tab. 6.1. Features, constraints and parameters in the example part

Feature	Constraints and parameters
Baseblock	Determined by two position points (0,0,0) and (100,100,10). length of baseblock = 100; width of baseblock = 100; height of baseblock = 10
Vertical_ support	Geometric constraints: verticalsupport_start coplane with baseblock_left; Radius of arc C_2 , R_1 =16; Center of arc C_2 determined by two distance constraints: D_1=50; D_2=60; C_1 tangent to C_2 ; C_3 tangent to C_2; Extrusion length W_2=10
Cylinder	Geometric constraint: Cylinder_top coplane with baseblock_left; Center of cylinder_top concentric to arc C_2 ; Height of cylinder H = 85
Rib	Distance constraint: distance between C5 and C7, $D_3 = D_2 - (R_1 - \sqrt{R_1^2 - (W_1/2)^2})$; Extrusion length W_1=10
Through_hole1	Geometric constraints: Through_hole1_top coplane with cylinder_top; Through_hole1_bottom coplane with cylinder_bottom; Center of through_hole1 concentric to the center of cylinder; Radius of through_hole1 = 8
Through_hole2	Radius of through_hole2 R_3 =3; Through_hole2_topcenter determined by three distance constraints: D_1, D_2+R_1, and D_4 ; Height of through_hole2 $H_2 = R_1 - \sqrt{R_2^2 - R_3^2}$

Tab.6.2. Redefining two features

Feature	Constraints and parameters
Cylinder boss	Radius of cylinder boss R_4 = 6; The top center of cylinder boss determined by three distance constraints: D_1, D_4 and D_5 (distance between top center of cylinder boss and top of base block in Z axis; Height of cylinder boss $H_2 = D_5 - D_2 - \sqrt{R_1^2 - R_4^2}$
Through_hole2	Radius of through_hole2 R_3 =3; Top center of through_hole2 coplane with top center of cylinder boss ; Top center of through_hole2 concentric with top center of cylinder boss; Height of through_hole2 $H_2 = D_5 - \sqrt{R_2^2 - R_3^2}$

Therefore, the result part model after modification can be generated as shown in Fig. 6.17(a). The dependency graph of the modified part can be expressed as shown in Fig. 6.17(b).

Subsequently, the designer wishes to remove the *boss* feature. According to the feature dependency graph shown in Fig. 6.17(b) and the latest feature precedence order, the removal of the *boss* feature can be done by removing the boss as well as the *hole* feature that depends on the *boss* feature. The final part after removing the *boss* is shown in Fig.6.18.

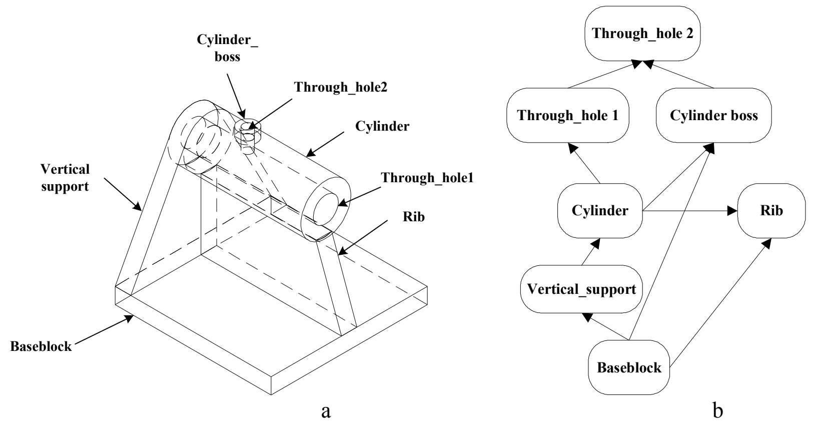


Fig. 6.17. Feature model after adding a boss a. modeling result; b. feature dependency graph after adding a boss

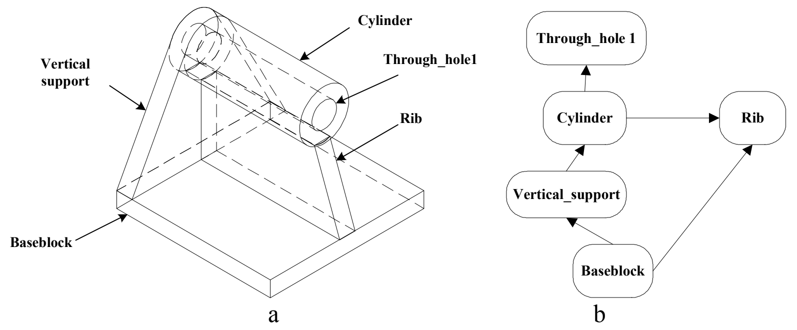


Fig. 6.18. Feature model after removing the boss feature a. modeling result after removing the boss feature; b. feature dependency graph

6.7 Conclusion

In this chapter, the integration of a fine-grain, feature-oriented database and a solid modeler, is presented. The mapping mechanisms, from EXPRESS-defined generic feature model entities to ACIS workform format, and the integration with the repository database schema are described. Generic algorithms for feature manipulation with the solid molder and database methods are illustrated. Finally, a modeler-supported, history-independent feature model re-evaluation approach is described in detail. Based on the working prototype system, it can be concluded that solid modeler can be effectively integrated with the feature-oriented database to provide low-level geometrical modeling services. This kind of integration can further enable information sharing among different applications and Web enabled engineering collaboration.

6.8 Acknowledgements

The authors gratefully acknowledge the support of technical staff in Design Research Center and CAD/CAM lab of Nanyang Technological University.

6.9 References

[1] Nwagboso C, Georgakis P and Dyke D, (2004), "Time compression design with decision support for intelligent transport systems deployment," *Computers in Industry,* 54:291-306.

[2] Terwiesch C and Loch CH, (1999) Measuring the effectiveness of overlapping development activities. *Management Science,* 26:44-59.

[3] Prasad B., (1996) *Concurrent Engineering Fundamentals: Integrated Product and Process Organization*, Prentice Hall.

[4] Mittra SS, (1991) *Principles of Relational Database Systems*, Prentice Hall.

[5] Raflik, M, (1990) *CAD*I Database-An Approach to an Engineering Database,* ECSC-EEC-EAEC.

[6] Regli WC and Gaines DM, (1997) A repository for design, process planning and assembly. *Computer-Aided Design*, 29:895-905.

[7] Kang SH, Kim N, Kim CY, Kim Y and O'Grady P, (1997) Collaborative design using the world wide Web. *Technical Report*, Dept. Industrial Engineering, Seoul National University, Korea.

[8] Loffredo D, (1998) Efficient database implementation of EXPRESS information models. *PhD Thesis*, Rensselaer Polytechnic Institute, New York.

[9] Hoffmann CM and Arinyo RJ, (1998) CAD and the product master model. *Computer-Aided Design*, 30(11):905-918.

[10] Wang HF and Zhang YL, (2002) CAD/CAM integrated system in collaborative development environment. *Robotics and Computer Integrated Manufacturing*, 18: 135-145.

[11] Wang HF, Zhang YL, Cao J, Lee SK and Kwong WC, (2003) Feature-based collaborative design. *Journal of Material Processing Technology*, 139:613-618.

[12] Tang SH, Ma YS and Chen G, (2004) A feature-oriented database framework for web-based CAx applications. *Computer-Aided Design & Applications*, 1(1-4): 117-125.

[13] Tang SH, Ma YS and Chen G, (2004) A Web-based collaborative feature modeling system framework. *Proceedings of the 34th International MATADOR conference*, 31-36.

[14] Chen G, Ma YS, Thimm G and Tang SH, (2004) Unified feature modeling scheme for the integration of CAD and CAx. *Computer-Aided Design & Applications*, 1(1-4):595-602.

[15] Martino TD, Falcidieno B and Habinger S, (1998) Design and engineering process integration through a multiple view intermediate modeler in a distributed object-oriented system environment. *Computer-Aided Design*, 30(6):437-452.

[16] Bidarra R and Bronsvoort WF, (2000) Semantic feature modeling. *Computer-Aided Design,* 32:201–225.

[17] Bidarra R, van den Berg E and Bronsvoort WF, (2001) Collaborative modeling with features. *Proceedings of DETC'01 ASME Design Engineering Technical Conferences*, Pittsburgh, Pennsylvania.

[18] Bronsvoort WF, Bidarra R, Dohmen M, van Holland W and de Kraker KJ, (1997) Multiple-view feature modeling and conversion. In: Strasser, W., Klein, R., and Rau, R. (Eds.), *Geometric Modeling: Theory and Practice - The State of the Art*, Springer, Berlin, 159-174.

[19] Bronsvoort WF, Bidarra R and Noort A, (2001) Semantic and multiple-view feature modeling: towards more meaningful product modeling. In: Kimura F, (Ed.) *Geometric Modeling - Theoretical and Computational Basis towards Advanced CAD Applications,* Kluwer Academic Publishers, 69-84.

[20] Kim J and Han S, (2003) Encapsulation of geometric functions for ship structural CAD using a STEP database as native storage. *Computer-Aided Design*, 35:1161–1170.

[21] Bhandarkar MP, (2000) STEP-based feature extraction from STEP geometry for agile manufacturing. *Computers in Industry*, 41:3-24.

[22] Dereli T and Filiz H, (2002) A note on the use of STEP for interfacing design to process planning. *Computer-Aided Design*, 34:1075-1085.

[23] Fu MW, Ong SK, Lu WF, Lee IBH and Nee AYC, (2003) An approach to identify design and manufacturing features from a data exchanged part model. *Computer-Aided Design*, 35:979–993.

[24] Holland P, Standring PM, Long H and Mynors DJ, (2002) Feature extraction from STEP (ISO10303) CAD drawing files for metal-forming process selection in an integrated design system. *Journal of Materials Processing Technology*, 125-126:446-455.

[25] Suh YS and Wozny MJ, (1997) Interactive feature extraction for a form feature conversion system. In: Hoffmann CM and Bronsvoort WF (Eds.), *Solid Modeling '97, Fourth Symposium on Solid Modeling and Applications*, 11-122, New York, ACM Press.

[26] Li WD, Ong SK and Nee AYC, (2002) Recognizing manufacturing features from a design-by-feature model. *Computer-Aided Design*, 34:849-868.

[27] Gao J, Zheng DT, and Gindy N, (2004) Extraction of machining features for CAD/CAM integration. *International Journal of Advanced Manufacturing Technology*, 24:573–581.

[28] Noort A, Hoek GFM and Bronsvoort WF, (2002) Integrating part and assembly modeling. *Computer-Aided Design*, 34:899-912.

[29] Bronsvoort WF and Noort A, (2004) Multiple-view feature modeling for integral product development. *Computer-Aided Design*, 36:929-946.

[30] Ma YS and Tong T, (2003) Associative feature modeling for concurrent engineering integration. *Computers in Industry*, 51: 51-71.

[31] Ma YS, Britton GA, Tor SB, Jin LY, Chen G and Tang SH, (2004) Design of an feature-object-based mechanical assembly library. *Computer-Aided Design & Applications*, 1(1-4):379-403.

[32] Geelink R, Salomons OW, Van Slooten F, Van Houten FJAM and Kals HJJ, (1995) Unified feature definition for feature based design and feature based manufacturing. *Proceedings of the 15th Annual International Computers in Engineering Conference and the 9th Annual ASME Engineering Database Symposium*, 517-533.

[33] Chen G, Ma YS, Thimm G and Tang SH, (2005) Knowledge-based reasoning in a unified feature modeling scheme, *Computer-Aided Design & Applications*, 2(1-4): 173-182.

[34] Chen G, Ma Y S, Ming XG, Thimm G, Lee SSG, Khoo LP, Tang SH and Lu WF, (2005) A unified feature modeling scheme for multi-applications in PLM. *Proceedings of The 12th ISPE International Conference on Concurrent Engineering (CE2005): Research and Applications -Next Generation Concurrent Engineering*, Sobolewski M and Ghodous P (Eds.), ISPE, Dallas, 343-348.

[35] ISO, (1994) Industrial Automation Systems and Integration — Product Data Representation and Exchange — Part 11: Description Methods: The EXPRESS Language Reference Manual, *ISO 10303-11*, Geneva.

[36] *ACIS Online Help User's Guide*. Available at http://www.spatial.com.

[37] *ORACLE online documentation*. Available at http://www.oracle.com.

[38] Sannella M, (1993) The SkyBlue constraint solver and its applications. *First Workshop on Principles and Practice of Constraint Programming*.

[39] Kramer GA, (1992) *Solving geometric constraints systems: a case study in kinematics*, The MIT Press, USA.

[40] Wirth N, (1976) *Algorithms + Data Structure = Programs*, Prentice-Hall.

[41] Bidarra R, Madeira J, Neels WJ and Bronsvoort WF, (2005) Efficiency of boundary evaluation for a cellular model. *Computer-Aided Design*, 37:1266-1284.