3

Fine Grain Feature Associations in Collaborative Design and Manufacturing – A Unified Approach

Y.-S. Ma[1], G. Chen[2] and G. Thimm[2]

[1] Department of Mechanical Engineering, Faculty of Engineering
University of Alberta, Edmonton, Alberta T6G 2E1, Canada
Email: yongsheng.ma@ualberta.ca

[2] School of Mechanical and Aerospace Engineering
Nanyang Technological University, Singapore
Emails: chengang66@pmail.ntu.edu.sg, mgeorg@ntu.edu.sg

Abstract
In the context of concurrent and collaborative engineering, the validity and consistency of product information become important. However, it is difficult for the current computer-aided systems to check the information validity and consistency because the engineers' intent is not fully represented in a consistent product model. This chapter consolidates a theoretic unified product modelling scheme with fine grain feature-based methods for the integration of computer-aided applications. The scheme extends the traditional feature concept to a flexible and enriched data type, unified feature, which can be used to support the validity maintenance of product models. The novelty of this research is that the developed unified feature scheme is able to support entity associations and propagation of modifications across product lifecycle stages.

3.1 Introduction

Product development comprises several lifecycle stages, such as conceptual design, detailed design, process planning, machining, assembly, *etc*. Commonly, computer-aided tools (called 'CAx systems' hereafter) are used to support activities associated to these stages. Traditionally, stand-alone CAx systems for individual stages produce separate models, such as a product design or a process plan. The existing CAx technologies have difficulties in maintaining the integrity of the comprehensive product model as inter-stage data transfer or sharing is insufficiently supported, especially for non-geometric data. Furthermore, validity checking of product models is difficult as the engineering knowledge applied in product designs or process plans is usually not stored with the product model as the existing technology does not allow for this. Recently, due to the drive for industrial globalisation and mass customisation, the trend of concurrent and collaborative engineering has led to tight integration of product and process domains as well as CAx systems [3.1].

This research accommodates product model validity and consistency by proposing a comprehensive product model consisting of linked geometric and non-geometric data throughout all product lifecycle stages based on feature technology with consideration of knowledge engineering, system integration, and collaboration. The goal of this research is to establish a paradigm in product modelling across multiple lifecycle stages. The multiple aspects of product modelling are integrated in a systematic and scalable manner. The paradigm is expected to allow multiple applications to share a consistent product model with supporting mechanisms and to maintain its integrity and validity.

3.2 Literature Review

Traditional application integration approaches focus on geometric data sharing. For example, system integration between design and reverse engineering, rapid prototyping, co-ordinate measuring machine, mesh generation for CAE, and virtual reality has been widely studied [3.2–3.7]. The most common approach to support application integration is using geometric data file exchange via a set of neutral formats, such as the Initial Graphics Exchange Specification (IGES) or the STandard for the Exchange of Product model data (STEP) [3.8]. This situation is no longer satisfactory to support modern product lifecycle management [3.1]. To support application integration fully, more comprehensive data sharing is needed than provided by the existing IGES or STEP standards.

Features combine geometric and non-geometric entities. Therefore, compared with geometric models, more complex relations exist in feature models. Managing these relations, especially the non-geometric ones, is essential for the validity of a product model. Relations in a feature-based product model can be classified as shown in Table 3.1.

Table 3.1. Summary of research on relations in a feature-based application

Relation	Related entity	Representation	Source
Geometric relations	Between geometric entities	Geometric constraints	[3.9, 3.10]
	Between features	Interaction constraints	[3.11, 3.12]
Non-geometric relations	Between features and the corresponding geometric entities	Features referred to the corresponding geometric entities	[3.13–3.15]
	Between features and other non-geometric entities, such as functions, behaviours, assembly methods, machines, cutting tools	Tables, graph, rules, *etc.*	[3.16–3.25]

3.2.1 Geometric Relations

Many publications focus on geometric relations in a feature model [3.9]. All these relations are explicitly declared and represented as geometric constraints, which maintain the geometric integrity of features. However, unintentional feature

interactions, may also affect the validity of features [3.11, 3.12]. These interactions usually cannot be prevented by geometric or algebraic constraints. This work will show that the geometric feature interactions can only be managed through the associations between the feature model and the geometric model.

3.2.2 Non-geometric Relations

Non-geometric relations refer to dependency relations involving non-geometric entities. For example, in process planning, the clamping faces or accessing faces are required and are to be preserved when machining a feature and they are associated to the machining processes and sequence used. Furthermore, two features, which do not spatially overlap, even belong to different product lifecycle stages, may interact with each other. How to represent these non-geometric feature relations has not been fully investigated.

Non-geometric relations also exist among features and non-geometric entities. For example in functional design stage, functional-form matrixes, bipartite function-feature graphs, design flow chain and key characteristics, and mapping hierarchy are used to link features to product functions [3.17, 3.20, 3.21, 3.24, 3.26]. In the process planning stage, features are also related to non-geometric entities, such as machines, cutting tools, and machining processes [3.22]. The methods of using non-geometric relations to validate product models have not been developed.

A product model has to be constructed or analysed iteratively using engineering knowledge from different aspects of expertise to fulfil requirements, such as functional or manufacturing requirements. In addition, lifecycle stages are inter-related and mutually constraining. Any modification in one stage may provoke a chain of subsequent modifications to entities of the same or other stages. This propagation of changes requires the management of inherent relations within and among these stages. In other words, a product model must have a sound mechanism to check its validity. Compared to the strict validity maintenance mechanisms of B-rep or CSG, current feature-based modelling schemes are weak in this aspect.

Laakko and Mantyla [3.14], and Rossignac [3.27] suggested that a feature's validity should be defined in terms of the referenced geometric entities and of their existence, shape, and relations to other geometric elements of the model. A feature model is valid if the geometric and algebraic constraints specified on features are satisfied. However, with the introduction of *associative features* [3.28], the validity of features must be checked in more complex scenarios. The associative feature concept expands feature definitions of specific application-related shapes into a set of well-constrained geometric entities. By using an object-oriented approach, a feature type can be modelled in a declarative manner that basically consists of the properties and behaviours. Feature properties define the geometric entities whose behaviours define the related constraints and logics in functioning methods throughout the lifecycle of any feature instance. With the built-in object polymorphism capability, a systematic modelling scheme for a generic and abstractive parent feature class, with levels of specification as per application domain requirements, can be developed. Such a generic feature definition scheme unifies many traditionally defined, application-oriented feature definitions and supports XML representation and fine grain database repository. Under the

associative feature concept, where the associative constraints across multiple phases of applications of a product lifecycle, complicated engineering features (patterns) and engineering intent can be implemented. An example associative feature, cooling channel pattern in plastic injection mould design, was given in [3.28]. An initial sketch-based conceptual pattern in the early mould design stage is implemented and its downstream cooling *hole* features are derived from the pattern; and then the related assembly interfacing features and associated standard components at the manufacturing and assembly stages are associatively generated and managed via a well-defined feature class model.

Feature validity is concerned with a feature's internal semantic characteristic properties, logics, constraints and attributes. This validity aspect is largely categorised as the constraint satisfaction problem, which has been partly addressed to a wide extent.

Feature consistency refers to the tally relations between related features or more abstracted semantic entities. Feature consistency is related to the semantic relations. The consistency requirement can have different types. Some researchers suggest that feature consistency means that the feature concerned is agreeable to the engineering intent [3.29]. In their publications, engineering intent must be transformed into a set of geometric, algebraic or preliminary semantic constraints, such as the boundary or interaction constraints [3.15]. However, during the transformation process, engineering intent may be lost because it has not been modelled explicitly so far. Others emphasise that non-geometric constraints, such as a dependency constraint, specified on the features have to be satisfied. For example, the presence of features, or the values of feature parameters, may be determined by functional requirements [3.18]. For another example, different machining sequences may influence the presence, form, volume, and validity of machining features. Hence, the presence of a machining feature is coupled with a machining process. Currently the representation, checking and maintenance methods of inter-feature non-geometric constraints are immature. Few researchers have touched on the feature consistency aspect although they are equally important for product modelling. A more detailed literature review by the authors is available [3.30]. This work introduces a solution framework that entails major class definitions, association structures, as well as integration and reasoning mechanisms based on a unified feature concept.

3.3 Unified Feature

Unified feature is a feature class definition that can generically represent the common properties as well as the required methods throughout product lifecycle stages. A unified feature is defined as a set of constrained associations among a group of geometric and non-geometric entities. The commonalities of application features, such as conceptual design features, detailed design features, and process planning features, are defined in the unified feature class as generic fields and methods. A brief publication can be found in [3.31]. Table 3.2 gives the major fields and methods defined in the unified feature class.

Figure 3.1 gives the generic definition using a UML diagram [3.32]. The UML symbols used in the figure are explained here. Rectangles represent classes, such as

Table 3.2. Major fields and methods of the unified feature class

	Name		Description
Fields	Attributes	Association attributes	Identities of the associated objects, such as functions and behaviours in a conceptual design, machines and cutters in a process plan, other features, *etc.*
		Self-describing attributes	Material, surface finish, belonging application, *etc.*
	Parameters		Variables used as input to geometry creation methods
	Constraints	Geometric constraints	Identities of geometric constraints that the feature's topological entities participate in
		Algebraic constraints	Identities of algebraic constraints that the feature's self-describing attributes or parameters participate in
		Rule-based constraints	Identities of rules that the feature or its self-describing attributes, parameters, numerical constraints participate in
	Geometric references		Topological entities
Methods	Geometry construction	*createGeometry()*	Generate the feature geometry
	Interface to geometric model	*getCell()*	Find out the feature's member topological entities
		setCell()	Assign a topological entity as the feature's identity
		insertGeometry()	Notify the geometric model to insert the feature geometry
		deleteGeometry()	Notify the geometric model to delete the feature geometry
	Interface to expert system	*getFact(), setFact()*	Retrieve or create the corresponding facts
		getRule(), setRule()	Retrieve or assign the corresponding rules
		checkRule()	Check whether the related rules are satisfied or not
	Interface to relation manager	*addToJTMS()*	Add the feature or its self-describing attributes, parameters to the relation manager as nodes
		validityChecking()	Call the relation manager for feature validation
	Interface to database	*saveFeature(), retrieveFeature()*	Store a feature in or retrieve a feature from the database

the *UnifiedFeature* class. Dashed and directed lines represent dependency relations. The lines are directed from the depending class to the class it depends on. Solid and directed lines with triangular open arrowheads represent generalisation relationships, pointing to the more general class that defines basic properties. Solid and directed lines with open diamonds represent aggregation relationships, pointing from the

parts to the whole, aggregated object. Composition (indicated by a filled diamond) is a variation of simple aggregation relationship. It describes strong ownership and coincident lifetime between the parts and the whole. The ranges aside the origin and target of an aggregation (or composition) arrow indicate how many parts can or must be in a whole. For example, a unified feature may include none or many other unified features. A circle attached to a class represents an interface (such as the *IAttribute*) realised by (undirected lines) the class. Other classes can use this interface, *e.g.* the *UnifiedFeature* class uses the *IAttribute* interface.

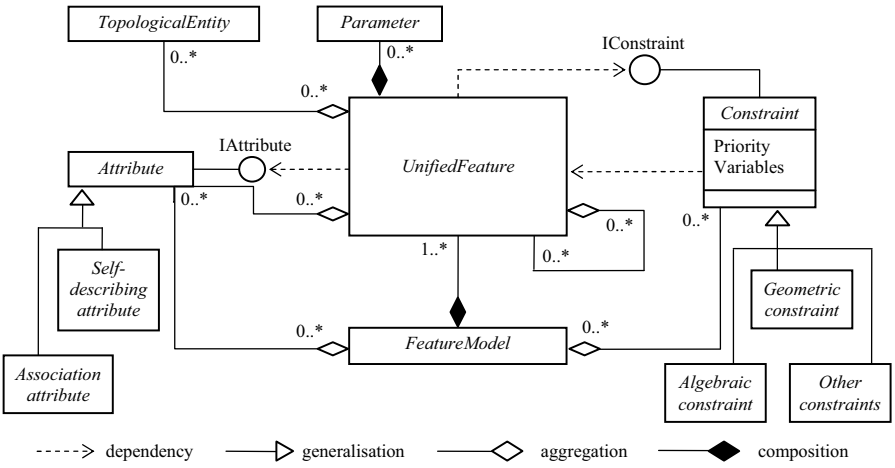

Figure 3.1. Unified feature

3.3.1 Fields

The unified feature class has four main kinds of fields.

(1) *Non-geometric attributes* represent feature properties that are attached to the feature or to the feature's geometric entities. They do not directly describe a feature's shape. Attributes are further classified into self-describing attributes and association attributes. Self-describing attributes represent properties that are special to a particular feature class. Examples of self-describing attributes are material type, surface finish, and feature nature (adding or removing material). Association attributes are references to the entities associated to this feature, such as other features, corresponding facts in the expert system, *etc*. In addition, association attributes are used to refer to non-geometric entities. For example, they refer to functions and behaviours in the conceptual design stage, or machine tools and machining operations in the process planning stage.

(2) *Geometric parameters* describe a feature's geometric shape, dimension, position, and orientation, such as the origin position and length, width, height of a block feature. Geometric parameters are used as input to the geometry creation methods provided by the geometric modelling kernel.

(3) *Constraints* can be classified according to the elements they constrain: (a) *intra-feature constraints* restrict the field values in a feature. For example, a

pocket's width equals to its length or a blind-hole's bottom face must be on the part boundary; (b) *inter-feature constraints* specify relations between two or more features; and (c) *semantic constraints* can also be specified between a feature and other entities. For example, a process planning rule is used as the constraint to specify whether a cutter can be used to create a feature with the specified shape, dimension, tolerance and surface finish. Constraints can also be classified according to their types, *i.e.* (a) *algebraic constraints*; (b) *geometric constraints*; and (c) *rule-based constraints*, which are used to restrict a feature's presence or the values of feature properties directly based on engineering rules. Constraints are prioritised.

(4) *Geometric references* are pointers to topological entities in a geometric model. Since features are used to describe specific relations between topological entities, a feature's geometry is not necessarily volumetric, connected, or two-manifold.

3.3.2 Methods

Interfacing functions, which deal with geometric modeller, knowledge engineering module, relation manager and database, are defined in the unified feature class.

(1) *Creating and editing feature geometry*. In the proposed scheme, conceptual design and detailed design features are created from predefined and parameterised geometric templates. The values of these parameters are specified to generate feature geometry. In the process planning stage with a design feature model as input, a process planning application analyses all machining faces for suitable process planning features. The properties of these faces are then used to determine the parameters of process planning features. Feature parameters are used to create product geometry with the help of functions provided by a geometric modeller. Because the definition of geometry is application specific, the way geometry is created is delegated to the specific application features. Feature geometries can be 2D faces or 3D solids in the developed scheme. The geometries of different dimensional features are represented uniformly in a non-manifold geometric model (Chapter 3.5). When an application feature is created, its geometry is inserted into the geometric model. When a feature is changed, it notifies the geometric model of modifications. In both cases, the geometric model will update itself accordingly.

(2) *Supporting knowledge embedment* [3.33]. A fact table corresponding to a set of associated features is created as a subset supporting a knowledge base. When an application feature is created, a corresponding fact is generated and inserted into the corresponding fact table and then accessible from the knowledge base. The fact of a feature describes the feature's identity, its parameters and self-describing attributes. The fact generation and insertion methods are defined in the unified feature class. When a feature is altered, it notifies the knowledge base. Matching rules (if any) are then fired.

(3) *Supporting data associations and validity maintenance*. In a single stage, when an application feature is created, a corresponding node is generated and inserted into a relation manager. The relation manager is responsible for managing the dependency relations among entities. The constraints, which are responsible for the feature's presence or controlling the values of feature parameters or self-describing attributes, are also inserted into the relation manager and are associated to

the corresponding feature node. The node generation, insertion and association methods are commonly defined for different application features. When a feature is modified, it calls the relation manager for change propagation. Related constraints are validated. To support inter-stage data sharing, associations and change propagation, application features as well as their inter-relations are stored in a common database. The methods of storing features into the database are defined in the unified feature class.

Two points about the above proposed unified feature definitions are worth noting. First, traditionally, numerical constraints are used to represent engineering intent. As an extension, the unified feature definition also defines associations to knowledge base, geometric model and other non-geometric entities in order to represent and maintain engineering intent. Second, from the viewpoint of software engineering, data sharing is difficult because one application does not know the data structures of other applications.

Hence, applications cannot manipulate the data created by other applications. With the unified feature definition, the issue of sharing feature data among applications is considerably improved. An application feature may have its specific properties, which are not included in the unified feature definition. However, with both application features defined as sub-classes of the unified feature class, an application understands the generic part of feature objects of other applications. These generic data is then used to reconstruct unified feature objects (Figure 3.2). In the proposed scheme, each application stores the data in a central relational database. An application can access the database to retrieve the data that is authorised.

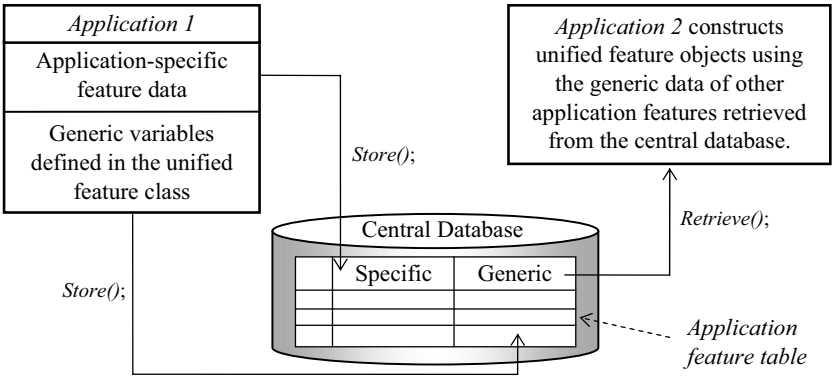

Figure 3.2. Data access methods via generic fields of application features

3.4 Entity Associations

The prime purpose of the unified feature-based product modelling scheme is to maintain the validity, consistency, and integrity of product models. Traditional CAx systems have limitations in serving this purpose. Two major problems are (1) engineering intent is not well represented and managed, and (2) inter-stage, non-

geometric relations are not well maintained. The unified feature-based product modelling scheme tackles these two problems via establishing and maintaining geometric and non-geometric data associations, within a single or across different stages. For example, in the conceptual design stage, the geometry of a feature is usually not fully defined. The resulted entities could be, for instance, only surface shapes, abstract mechanism concepts, or parameterised volumes without assigning detailed properties. An abstract conceptual design feature has its concrete counterparts in the detailed design feature model. Because a conceptual design feature represents a primitive design function that is usually realised through the interactions between a few components, it is likely that an individual conceptual design feature is transformed into several features belonging to different components in the detailed design stage. On the other hand, one detailed design feature may also participate in the realisations of several conceptual design features. Such *feature object dependency* associations are one kind of non-geometric associations between features as discussed in [3.34]. Feature attributes, parameters, or constraints specified in the conceptual design feature model are transformed into attributes, parameters, or constraints for corresponding detailed design features. For example, a parameter of a conceptual design feature may be transformed into a constraint between two detailed design features of different components. A conceptual design constraint could be related to several constraints in the detailed design feature model. Such *feature property dependency* associations are another kind of non-geometric associations across features of different stages [3.34]. These associations are generalised as constraint-based associations and sharing associations (Figure 3.3). Constraint-based associations are established on the basis of intra- or inter-stage, numerical or rule-based constraints. Sharing associations are established based on the unified cellular model.

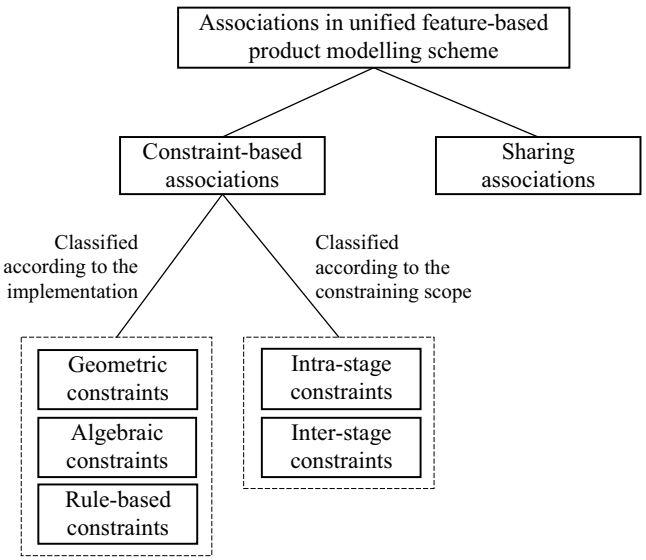

Figure 3.3. Associations in the unified feature-based product modelling scheme

3.4.1 Implementing the Constraint-based Associations

Together with a rule-based expert system and a numerical constraint solver, a justification-based truth maintenance system (JTMS) is used to implement the constraint-based associations as introduced in [3.34]. A JTMS dependency network consists of a series of related nodes that represent the belief status of entities. *Assumption* nodes are believed without any supporting justifications. *Simple nodes* are only believed if they have valid justifications. An assumption node can be converted into a simple node, which then needs to be supported by justifications. A justification consists of antecedent nodes and consequent nodes. A node is said to be justified by a supporting justification if all antecedents of the justification are justified.

Whenever a constraint-based association is generated, the corresponding JTMS nodes and justifications are inserted into a JTMS dependency network. After the insertion process, each node records three items: (1) a reference to its direct supporting justification; (2) references to the justifications that use this node as antecedent (for later change propagation); and (3) its current belief status. Whenever a modification to the JTMS dependency network occurs, such as adding or retracting assumptions, modifying nodes or adding justifications, the JTMS dependency network is searched for the affected nodes as well as the related justifications. If it is a rule-based constraint to provide the justification, the system refers to the knowledge base to validate the modification. If it is a numerical constraint to provide the justification, the system refers to the numerical constraint solver to validate the modification. These checking and change propagating processes are automated. The result is a new status of each affected JTMS node or a rejection of the modification on the basis of contradicting beliefs. The data structures and algorithms of JTMS are generic. Therefore, it handles geometric and non-geometric constraints uniformly.

A relational database is used for all applications to store and publish their data. An application can access and enquire the database for data published by other applications. When an inter-stage constraint-based association is established, this association and the involved data are stored in the database. When an application modifies its model, it must check the database for relevant inter-stage associations. If such associations exist, a validity checking process is triggered. The applications involved are responsible for maintaining the consistency (between associated stages) while the database is a medium for storing the repository data, inter-stage associations, and propagating changes. Figure 3.4 illustrates the constraint-based associations between the conceptual and the detailed design feature models. The constraint-based associations between the detailed design and the process planning feature models are established in a similar way.

3.4.2 Implementing the Sharing Associations

Two methods are developed for sharing associations using a unified cellular model implemented in a database.

(1) *Generating a new application feature.* Each application feature class has its geometry creation and manipulation functions. When a creation function is invoked, the feature geometry is created and inserted into the application's runtime cellular

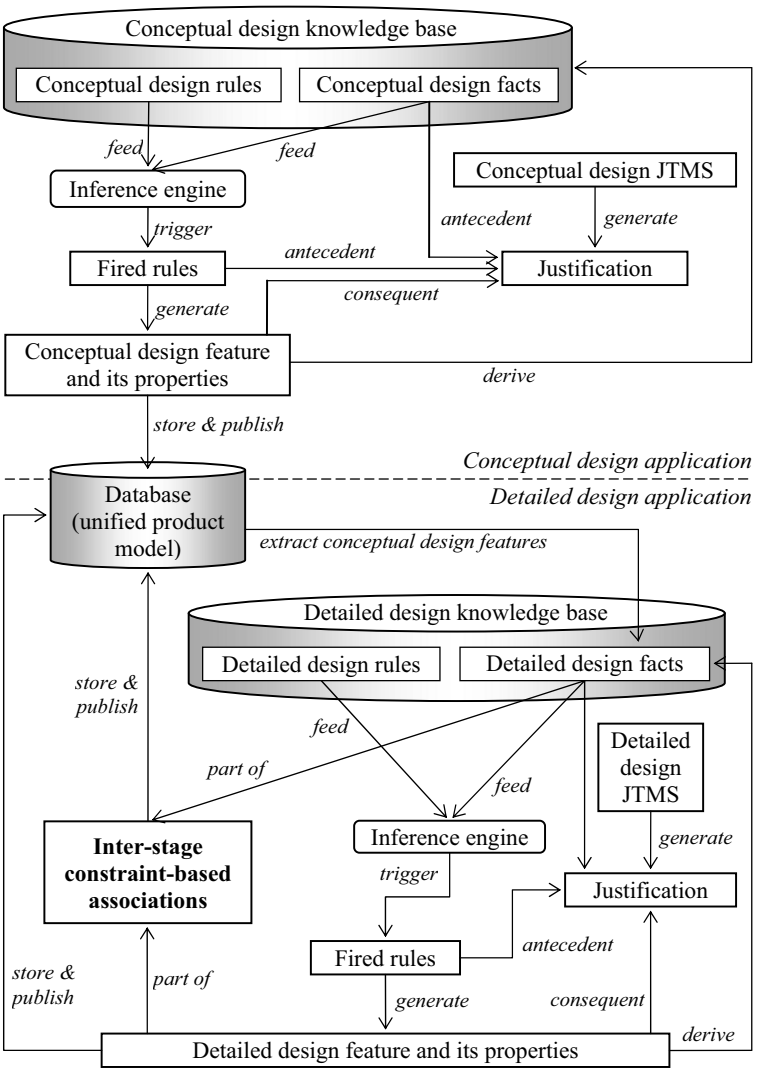

Figure 3.4. Constraint-based associations for conceptual and detailed design stages

model. The topological entities created are associated with the feature through the owning feature attributes and the feature's runtime geometric references. The feature geometry is also inserted into the unified cellular model. If any cell in the unified cellular model is affected by this new feature, *e.g.* overlapping, the owning features of the affected cells are marked for validity checking.

(2) *Modifying an application feature*. When an application feature is modified, in addition to updating the application's runtime cellular model, the application also notifies the unified cellular model about the modifications. The unified cellular model is updated and the affected cells are marked as been modified. The owning

features of the affected cells are then validated by the corresponding applications. The sharing association mechanism enables different application features to be associated with the same geometric or topological entities and hence supports achieving inter-stage geometric consistency.

3.4.3 Evaluation of Validity and Integrity of Unified Feature Model

This subsection introduces a set of criteria, which is used to evaluate the validity and integrity of a unified feature-based product model. The general requirement for a valid product information model is that each application model (corresponding to a particular stage) must be valid and also consistent with other associated application models. The detailed evaluation criteria are classified into feature, intra-stage, and inter-stage levels.

A feature is valid if (i) the feature geometry refers to valid topological entities; (ii) the values of feature parameters are consistent with the product's geometric model; (iii) all constraints on the feature are satisfied; and (iv) any feature property, if included in the JTMS dependency network, has a "believed" status, *i.e.* its supporting justifications are valid.

A product model is valid if (i) all features in the model are valid; (ii) in its knowledge base, the antecedent conditions of all fired rules, which are the justifications for the generated features (or feature properties), are satisfied; (iii) all constraint-based associations between consequent facts and respective features (or feature properties) hold; and (iv) cellular entities, which are referenced by the geometric references of all the existing features, exist and have the correct status (material or void, on the boundary or not on the boundary) according to the feature sequences in their owning feature lists.

Two product models (corresponding to different lifecycle stages) are consistent if (i) sharing associations between their corresponding application features hold; and (ii) constraint-based associations between their corresponding application features or feature properties hold. In particular: (a) each critical feature in the conceptual design is linked to features in the detailed design via valid constraint-based associations; (b) each feature property or inter-feature constraint in the conceptual design has its valid counterparts (may not be one to one relations) in the detailed design; (c) each detailed design feature to be machined is linked to process planning features via valid constraint-based associations; and (d) all the design specifications (such as tolerances and surface finishes) are satisfied by the finish process planning features.

3.4.4 Algorithms for Change Propagation

If users (designers or process planners) modify the product model, the modifications must be checked to make sure that the consistency of the whole product information model is maintained. As indicated in previous sections, a dependency network is established using constraint-based associations and sharing associations. It is implemented through a JTMS and a common database. The purpose of the dependency network is for the propagation of modifications and determining the influence scope of a modification.

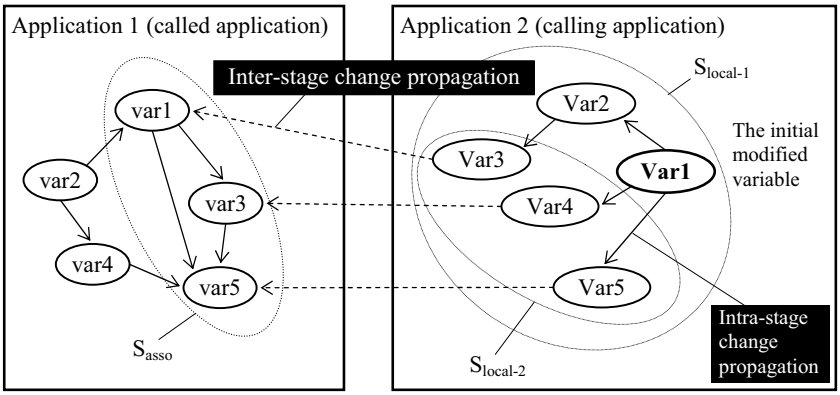

Figure 3.5. Propagation chain of intra- and inter-stage changes

The propagation and checking process is divided into two major generic routines [3.35]: local checking within a specific model and global checking across different models (see Figure 3.5). Assume variable *x* in an application is changed (as the initial modification). In the figure, arrows are directed from driving to driven variables while "var" represents variables. The change propagation algorithms are developed with reference to constraint-based and sharing associations as well as their corresponding implementations in the unified feature-based product modelling scheme. The algorithm is in iterative manner and starts from a local application domain first; the local change impact is evaluated using a JTMS and a common database to establish inter-stage non-geometric associations. An algorithm for change propagation within a lifecycle stage is presented as follows.

PROCEDURE *Check_Local(x)*
/* checking the intra-stage associations */

(1) Backup the value of the initial modified variable *x*. Put *x* into a local set (set_1, which records modified variables v_i that need to be checked for intra-stage associations). For each v_i in set_1, search the JTMS dependency network for variables that associate to v_i using JTMS attributes (antecedent or consequent). The variables, which are antecedents of v_i, are driving variables. The variables, which are consequents of v_i, are driven variables.

(2) Check the constraints between each v_i and its driving or driven variables one by one:

– If the new value of v_i violates the constraints between v_i and any of its related variables:

→ If the related variable is a driven variable

- If the value of the driven variable is fixed by the constraint, *i.e.* without alternative values, then the modification is rejected and run *Abandon()*.
- If the driven variable has alternative values, search one for which the constraint is satisfied:

- If the constraint can be satisfied (and the value of the driven variable is changed), make a backup of the old value and put the driven variable into set_1.
- If no alternative value satisfies the constraint, then *Abandon()*.

→ If the related variable is a driving variable, then *Abandon()*.

– If no constraint has been violated or if some constraints has been violated but can be re-satisfied, the modification is locally accepted.

The *Abandon()* used in the above algorithm is given here:

PROCEDURE *Abandon()*
/* retracting all changes temporarily made */

(1) All modifications made in the calling and called applications are revoked using backup values.
(2) In the database, the data of the called application, whose values are temporarily changed, are set back to their original values.

Next, further check is carried out for v_i in the database. If v_i appears in any inter-stage associations in the database, move v_i from set_1 to set_2 (which records variables that need to be checked for inter-stage associations). Run *Check_Global()*.

PROCEDURE *Check_Global()*
/* checking the inter-stage associations */

(1) For each member of set_2, add all associated features or feature properties in the database to set_3 (which records associated variables in other applications). An initial modification in an application may invoke many modifications in other applications. The members of set_3 are checked (in the next two steps) one by one until set_3 becomes empty.
(2) The values of members of set_3 are temporarily changed in the database using the constraints recorded in the calling application.
(3) For each member of set_3, execute *Check_Local()* in the called application until the modification is found to be locally accepted or rejected. The corresponding message (about whether the modification is locally accepted or rejected in the called application) is sent back to the calling application that initiates the initial modification.
   – If all modifications are accepted, the initial modification in the calling application is globally accepted and committed.
   – If any of the modifications are rejected, the initial modification in the calling application is rejected, then *Abandon()*.

After the iteratively looped change propagation procedures, finally, the algorithm for change propagation finishes if there is no new changes triggered. Then further high-level attributes properties of related objects, logical facts, statuses of indicators and controls are updated accordingly. Eventually, the consistency of product models has been evaluated. This algorithm can be triggered again and again whenever there is a major change decision is to be committed.

3.5 Multiple View Consistency

3.5.1 Cellular Model

Traditional geometric modelling systems use boundary representation (B-rep) or constructive solid geometry (CSG) models for geometry representation [3.36]. They have limitations with respect to the requirements of the unified feature-based product modelling scheme.

Only the final product geometry is stored and managed. Intermediate geometries, which do not belong to the final boundary, are usually not stored. This limitation makes feature modifications difficult. It also results in a persistent naming problem.

CAx systems have different requirements on representing product geometry. A hybrid geometric modelling environment that can accommodate the associative wireframe, surface, and solid models coherently is a natural outcome of the unified feature-based product modelling scheme. CAx systems need to represent the same product geometry in different ways. On one hand, geometry may be represented in different abstraction levels. For instance, a hole can be represented as a central line (plus a radius), a cylindrical face, or a cylinder in different contexts. On the other hand, product geometry may be represented in different ways. For instance, two adjacent faces in one application may be represented as a single face in another application. In addition, it is important for the unified feature-based product modelling scheme that higher level application features can use lower level topological entities to propagate modifications and control the information consistency. Relationships or constraints in higher levels (*e.g.* feature level) may also be specified using lower level (*e.g.* topological entity level) relations.

Some solutions were proposed to solve these problems. Bidarra *et al*. [3.37, 3.38] proposed to use a cellular model to represent intermediate product geometry as well as to support links between different views. However, their methods are confined to 3D features only. Other researchers [3.39–3.42] proposed to use the multi-dimensional non-manifold topology (MD-NMT) to meet the geometric modelling requirements of different applications. However, they did not fully apply the MD-NMT to the feature-based modelling processes. It can be seen that a multi-dimensional geometric modelling environment, which is capable of propagating geometric modifications across feature models, does not exist.

3.5.2 Using Cellular Topology in Feature-based Solid Modelling

The goal of using a cellular topology is to keep a complete description of all the input geometric entities without removing them after set operations on volumes (unite, intersect, and subtract), regardless of whether they appear in the final boundary or not [3.43]. The cellular model uses three mechanisms to fulfil this goal:

1. *Attribute mechanism.* There are two kinds of attributes used in a cellular model: (a) *cell nature* – a cell is either additive or negative depending on whether it corresponds to materials of the product (or the topological entities on the part boundary) or not; and (b) *owner* – each cell records its owing features because a cell may belong to several features due to feature

interactions. The sequence of the owning features is kept to determine the cell nature.

2. *Decomposition mechanism.* Two 3D cells do not overlap volumetrically. Whenever two cells overlap, new cells for the overlap are generated with the merged owner list.
3. *Topology construction mechanism.* In a cellular topology-based, non-manifold boundary representation, an operation on volumes does not remove any input geometry. The cellular model constructs topology, or generates new faces, edges, and vertexes, before classifying the topological entities as "in", "out" or "on" the boundary. All topological entities are marked and filtered for displaying according to the type of the operation.

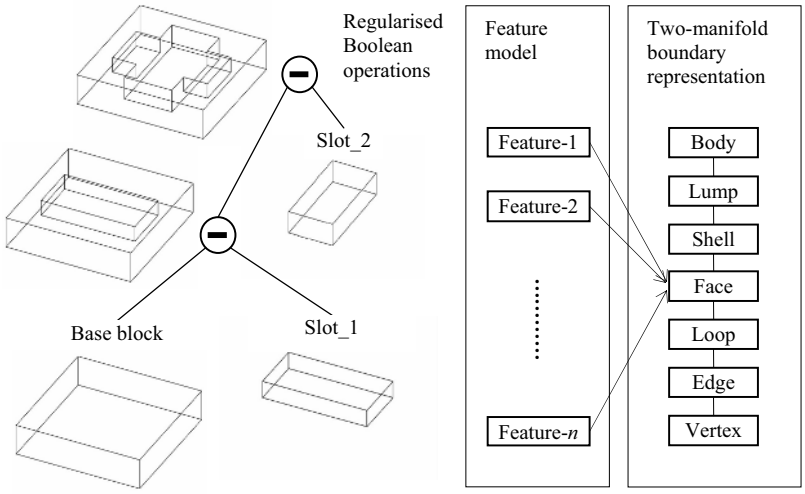

Figure 3.6. Traditional feature model based on a two-manifold boundary representation

Figure 3.6 describes a feature model based on a two-manifold boundary representation. Traditional boundary model does not store intermediate geometries. In other words, according to the types of Boolean operators, "useless" geometries are discarded before the part topology is reconstructed. For example, in Figure 3.6, when inserting the slot_1 feature, its top face is not stored. During the later modelling process, the intersections (due to feature interactions) further split and remove feature geometry from the boundary model. It is hence difficult to relate the feature to its corresponding topological entities in the final boundary model. In the constraint-based, parametric design processes, this limitation makes the feature model history-based. This limitation is also the major reason for the persistent naming problem [3.15].

Alternatively, in cellular topology based non-manifold boundary representations, operations on volumes do not discard any input geometry. Part geometries are represented using cellular topologies. Figure 3.7 shows the decomposed cells of the simple example according to the cellular topology along with the feature modelling process.

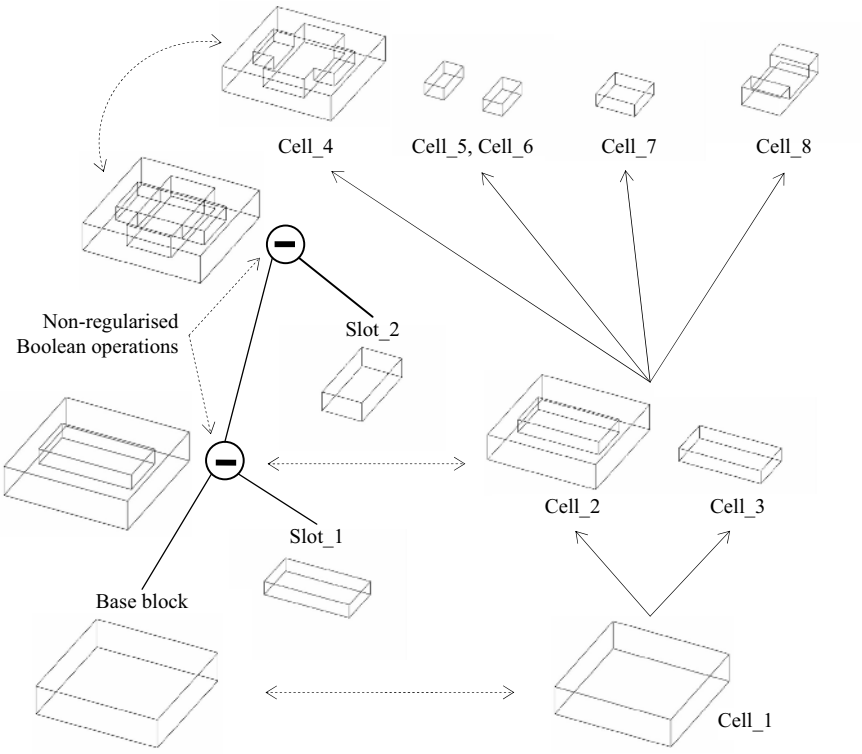

Figure 3.7. Feature model based on the cellular topology

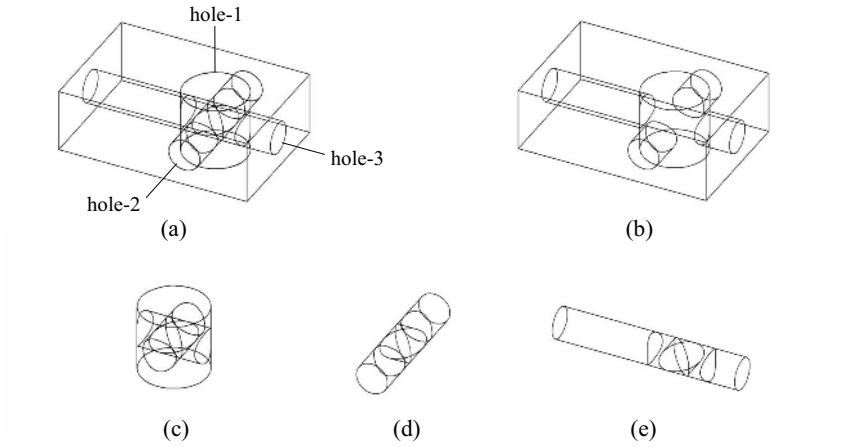

Figure 3.8. Cellular geometry with different cylindrical features: (a) a block with three interacting holes, (b) the cell, which belongs only to the block feature, (c) hole-1 feature, (d) hole-2 feature, and (e) hole-3 feature

The use of cellular topologies simplifies the representation of feature geometry as combinations of cells. When a feature is initialised, it is a single cell that carries only this feature's identifier. Whenever feature interactions occur, new cells that belong to the intersections are generated. The newly generated cells carry a merged owning feature list.

The geometry of a cell can describe any shape. For example, Figure 3.8(a) shows a block workpiece with three intersecting holes. Figure 3.8(b) shows the cell, which carries only the block as its owning feature. Figures from 3.8(c) to (e) show the cellular representations of the three hole features. In this way, all feature geometry, geometric relations between features as well as relations between a feature and its corresponding topological entities are stored in the product model persistently. The persistent naming problem is avoided. It is also possible to modify features based on the dependency relations, not on the construction history because the influence scope of a geometric modification is confined by the inter-cell relations. However, to meet the requirements of the unified feature-based product modelling scheme, this 3D-cell-based multiple-view feature modelling approach needs to be extended. Some case studies are given in [3.43].

3.5.3 Extended Use of Cellular Model

Distinct applications covered by the unified feature-based product modelling scheme have their particular geometry representation requirements. (1) During conceptual design, a designer is concerned about functions and behaviours. Only critical geometries and their relations are specified at that time. These critical geometries may only be represented as abstracted lines, faces, curves, or surfaces. Solid models, detailed topologies and geometries are not specified in this stage. (2) In the detailed design stage, the product geometries or layouts are further materialised. Two-manifold solid model representation is usually preferred. (3) In the process planning stage, features are usually defined as material removal or accessing volumes related to machining operations. Fixtures are also conceptualised in this stage. For these types of features, solid representation with surface manipulation support is more appropriate because, other than the machined volumes, fixture design uses sub-area patches of the part, *e.g.* locating or clamping areas. (4) Similar requirements are applicable to the assembly design stage. In particular, the sub-areas of the part or assembly for interfacing or grasping are concerned.

The geometrical representations discussed above relate to each other. They represent different aspects or abstraction levels of a product. To meet these diverse geometry representation requirements, the current cellular topology-based feature modelling method needs to be extended to support not only 3D solid features, but also non-solid features. A multi-dimensional cellular model, named the *unified cellular model*, is proposed here to integrate all these representations, manage their relations and hence support the multiple-view feature-based modelling processes. The geometric model of each application is a particular aspect (a sub-model) of the unified cellular model. The traditional usage of the cellular topology in multiple-view feature modelling is extended in three aspects: (1) 2D and 3D features are supported uniformly; (2) the unified cellular model is used to share geometric data as well as to propagate geometric modifications (creating new cells, modifying or

deleting cells) among views through the cells' owner attributes; and (3) relationships in the cell level are generalised. These relation types can be used as building blocks to establish higher level feature relations.

The unified cellular model ensures the geometric consistency between the application feature models.

1. The geometries of a detailed design and the corresponding process planning model may have different topologies. However, both models correspond to the same final product geometry. In other words, these two application cellular models must correspond to the same B-rep solid model, which represents the final part geometry. This geometric consistency is realised through mapping 2D or 3D application features to the corresponding cells in the shared unified cellular model.
2. Two features may represent the same item at two abstraction levels, *e.g.* a central line or a cylindrical face of a hole. The consistency is maintained through specifying geometric or topological constraints on the related cells in the unified cellular model.

3.5.4 Characteristics of the Unified Cellular Model

A unified cellular model *UCM* includes all geometries from different applications [3.43]. It consists of a set of cells:

$$UC_i{:}\,UCM = \left(\bigcup_{i=1}^{q} UC_i^0\right) \cup \left(\bigcup_{j=1}^{r} UC_j^1\right) \cup \left(\bigcup_{k=1}^{s} UC_k^2\right) \cup \left(\bigcup_{l=1}^{t} UC_l^3\right) \tag{3.1}$$

In the expression, UC^0, UC^1, UC^2, and UC^3 represent zero-dimensional (0D), one-dimensional (1D), two-dimensional (2D), and three-dimensional (3D) cells, respectively. Similarly, q, r, s, and t are the numbers of 0D, 1D, 2D, and 3D cells, respectively, in the unified cellular model.

Each cell (except 0D cells) is bounded by a set of cells of a dimensionality lowered by one. On the other hand, a cell may exist independently without bounding any higher dimensional cell. The point sets of any two cells (of the same or different dimensionalities) do not overlap: $UC_i^a \cap UC_j^b = \phi$ ($0 \le a < b \le 3$ or $(a = b) \wedge (i \ne j)$). In addition, a cell does not include its boundary, except for 0D cells.

The cellular model obeys the Euler–Poincare formula for non-manifold geometric models [3.41, 3.43]. Each application feature model uses the unified cellular model. The relations among these models are described in [3.43]. An application feature model *AFM* consists of a set of application features AF_i and other non-geometric entities NGE_j:

$$AFM = \bigcup_{i=1}^{m} AF_i \cup \bigcup_{j=1}^{n} NGE_j \tag{3.2}$$

where m and n are the numbers of application features and non-geometric entities in this application feature model.

An application cellular model ACM is created at runtime, which consists of a set of application cells AC_i: $ACM = \bigcup_{i=1}^{u} AC_i$, where u is the number of application cells in this application cellular model. Each application feature refers to a set of application cells. An application cell may belong to several application features, *i.e.* it records several features in its owning feature list. The geometries of an application feature correspond to 1D, 2D, or 3D cells.

An application cell can be mapped to one or more cells in the unified cellular model. On one hand, for a particular application, one cell in the unified cellular model is mapped to at most one application cell. On the other hand, each cell in the unified cellular model is mapped to at least one application cell (and therefore at least one application feature). This mapping is realised through the owner attribute mechanism.

The rule for determining cell nature applies to the unified cellular model, *i.e.* the nature of the latest feature in the owner list determines the nature of the cell. Please refer to [3.43] for more details. All applications use this unified multi-dimensional non-manifold cellular model. The geometry of each application feature model is a particular aspect of the unified cellular model.

In Figure 3.9, the design of a cooling system of a plastic injection mould is used as an example to illustrate the idea. In the conceptual design stage, the cooling system is represented as cooling circuits for cooling effect analysis while in the detailed design or process planning stage, the cooling system is in 3D for manufacturability analysis and process planning. The cooling circuits and the cooling channels are representations of the same cooling system in different abstraction levels. The geometries of these two feature models are kept consistent through the unified cellular model.

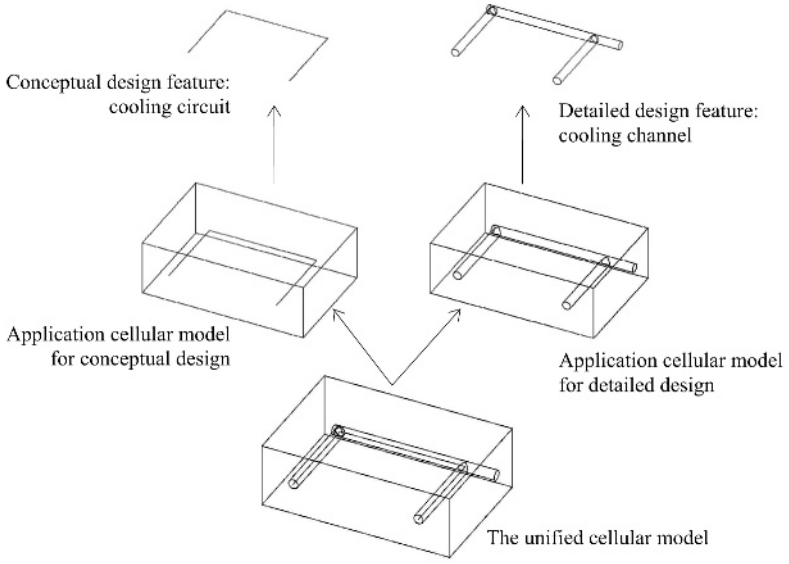

Figure 3.9. Link conceptual and detailed designs using unified cellular model

3.5.5 Two-dimensional Features and Their Characteristics

The idea in this sub-section is that the unified cellular modelling scheme represents 1D, 2D, and 3D cells uniformly (they are referred to as edge, face, and solid cells, respectively hereafter). Currently, the prototype system only handles face and solid cells (corresponding to 2D and 3D features, respectively). 1D features are mentioned here but more research will be done in the future. Examples of 2D application features include: (1) conceptual design features, which represent functional areas in the product; the geometries of this kind of features are usually abstracted as pairs of interacting faces; these faces correspond to partial faces in the detailed design; (2) assembly features, which represent grasping or mating areas of parts in assembly processes; and (3) locating or clamping features, which represent locating or clamping faces during a machining operation.

Similar to solid cells, the major advantage of using face cells instead of geometric faces is that operations on face cells do not remove faces (or parts of faces) even if they do not belong to the final boundary. For example, the non-regularised operations on faces and decomposition mechanism upon overlapping detection are available to face cells. A 2D feature is represented as a group of associated face cells with engineering semantics. For example, a locating feature is defined as a pair of faces associated with the constraints on accessibility, machining accuracy, non-interference, and minimising setup changes. The geometry of a 2D feature is one or more surfaces. Two characteristics of 2D features exist. (1) A 2D feature has a nature attribute (additive or negative) that can be changed by feature interactions. A change of cell nature (from additive to negative or *vice versa*) requires the corresponding features to be validated. For example, a clamping feature represents a local area on a part that is used for clamping. When a clamping feature is altered, its face cells may be split with the natures of some of the resulting face cells inverted. This may jeopardise the clamping feature's stability (sufficient area for clamping). Similar situations are encountered for functional, assembly, and locating features. (2) Face cells corresponding to functional, assembly, locating, and clamping features have the same surface definitions as existing face cell(s). Hence, to simplify the implementation, it is assumed that newly inserted face cells and existing solid cells do not intersect. However, this is not valid for some CAE analysis applications, in which middle faces are commonly used.

When a 2D feature is generated, the corresponding face cell is also generated and inserted into the application and the unified cellular models. Figure 3.10 illustrates a simple example: the integration of a detailed design and a process planning model on the basis of a unified cellular model.

The designed part is a block with a blind hole. The hole has a distance specification with face F2 as datum and a perpendicularity specification with face F1 as datum. The corresponding process planning model begins with a blank feature (larger than the part to allow machining). Other process planning features are (1) a surface-milling feature (due to the perpendicularity specification); (2) a clamping feature (for surface-milling); (3) a drilling feature and a boring feature (to meet the surface finish requirement of the hole); and (4) a locating feature and a supporting feature for the drilling and boring operations. These 2D or 3D features are associated in the unified cellular model.

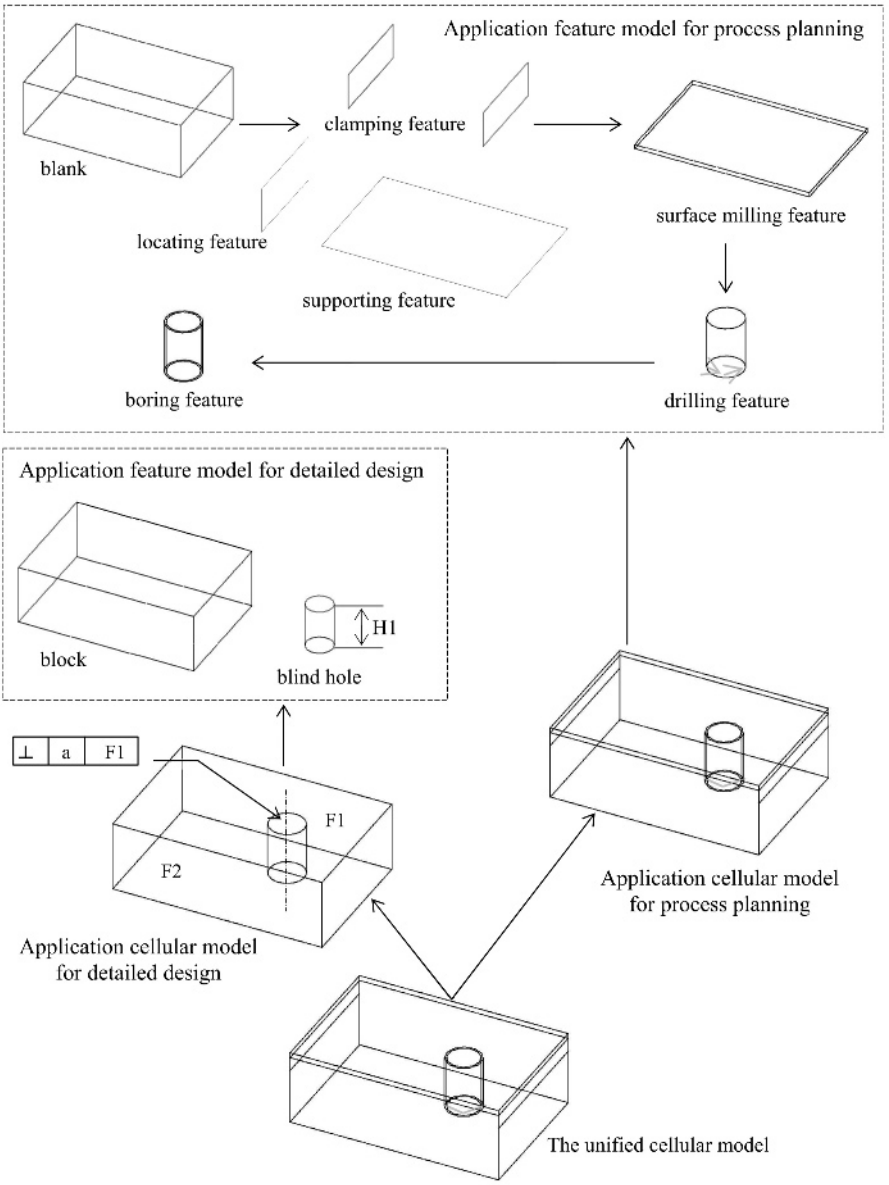

Figure 3.10. Integrating detailed design and process planning feature models

3.5.6 Relation Hierarchy in the Unified Cellular Model

Relations can be established on the cell level, the feature geometry level, and the feature semantic level, respectively. Higher-level relations are established on the basis of two lower-level relations.

The lowest level of relations is between two cells, which cover four cases:

1. The bounding relations among cells
2. The bounding cells that inherit the owner attributes of the bounded cell
3. Two solid cells that are adjacent if they are bounded by one or more common face cells (two face cells are adjacent if they are bounded by one or more common edge cells)
4. Two adjacent edge or face cells that may be part of the same curve or surface

The second level relations are topological relations between the geometries of application features. Note that a feature's dimensionality can be diverse depending on the application nature. Three possible topological relations between two application features are identified here:

1. *Overlap*: After cellular splitting, two *n*-dimensional features are said to *overlap* each other if they use the same *n*-dimensional cell(s). An *n*- and a (*n*–1)-dimensional features are also said to *overlap* each other if they use the same (*n*–1)-dimensional cell(s).
2. *Adjacent*: Two different *n*-dimensional features are defined as *adjacent* ones if they share (*n*–1)-dimensional cell(s) but do not overlap.
3. In a 3D feature, *adjoining area* refers to one or more faces (represented by the face cells), which are mathematically connected and defined on the same surface.

For two 3D features *A* and *B*, feature *A* is said to be *completely adjacent* to feature *B,* if feature *A*'s adjoining area is fully enclosed by any of feature *B*'s adjoining area. In plastic injection mould design, completely adjacent relations can be used to represent maps from the plastic part to core or cavity inserts as well as electrode geometry. Such maps are commonly encountered in die casting, forging tooling, and fixture design as well. Again, for more details, refer to [3.43]. Other examples are: (i) a single face in the detailed design corresponding to several functional faces in the conceptual design; and (ii) a face in the process planning model corresponding to one or more faces in the detailed design.

Higher level relations are semantic relations between application features. Relation types in this level are application specific. Examples are:

1. *Splitting*. Figures 3.11(a) to (c) show a base block with a *hole* feature; and the *hole* feature is further split by a vertical *through-slot* feature. A similar situation for 2D features is shown in Figures 3.11(d) and (e), in which the original clamping feature is split by a newly inserted *through-slot* feature. The middle face cell of the clamping feature becomes negative. The clamping feature must hence be checked for stability. This kind of relation between two interacting features is defined as a splitting relation [3.38]. Using the above-mentioned two lower levels of relations, the splitting relation can be described as: (i) the nature of the second feature is negative; (ii) the two features overlap; and (iii) the insertion of the second feature splits the original single cell (additive or negative) of the first feature into several (at least three) cells, where the nature of at least one of the middle cell(s) is negative.

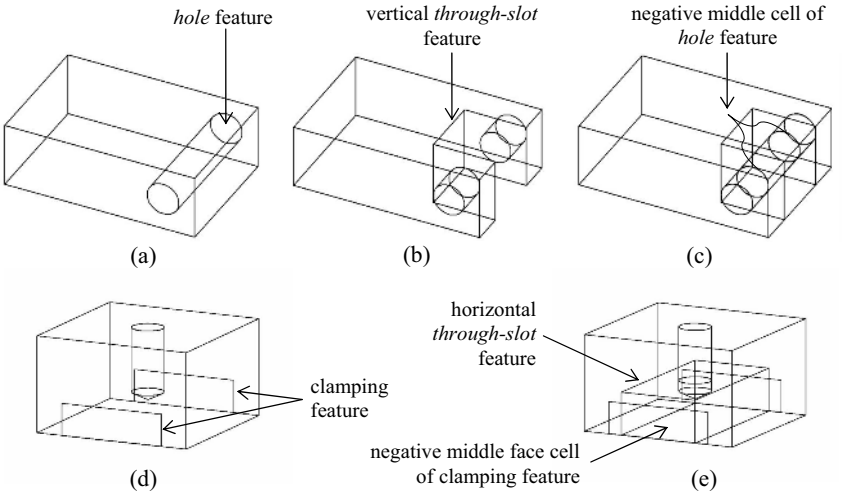

Figure 3.11. Splitting relation: (a) insert the first feature; (b) the second feature splits the first feature; (c) the middle cell of the first *hole* feature becomes negative; (d) two additive face cells of the clamping feature are inserted; and (e) the middle face cell of the *clamping* feature becomes negative

2. *Transmutation.* Figure 3.12 shows a base block with a *blind-hole* feature and a vertical *through-slot* feature. The relation between these two features is defined as a *transmutation* relation in [3.38]. Using the above-mentioned two lower levels of relations, the transmutation relation can be described as: (i) the nature of the two features is negative, but the nature of one of the bounding cells of the first feature, which represents the bottom face of a blind hole, is additive; (ii) the two features overlap; and (iii) the insertion of the second feature splits the original single 3D negative cell into two 3D negative cells. The previous additive bounding 2D cell becomes negative.
3. *Non-interference*. This relation specifies that two features cannot overlap with or are adjacent to each other. This constraint is satisfied if no cell in the unified cellular model has both of these two features in its owner list. This constraint is commonly used in product design or manufacturing activities. For example, a process planning feature cannot interfere with the corresponding clamping features.

3.6 Conclusions

Unified feature theory is a significant contribution to feature level collaboration in future virtual enterprises. In the proposed scheme, unified features provide an intermediate information layer to bridge the gap between engineering knowledge and product geometry. Unified features are also used to maintain geometric and non-geometric relations across product models. The feasibility of the proposed unified feature modelling scheme is demonstrated with a prototype system and case studies.

With the unified feature definition, application feature definitions, the unified cellular model, dependency network, and the change propagation algorithm, the proposed unified feature-based product modelling scheme is able to integrate the conceptual design, detailed design, and process planning applications. For detailed case studies, please refer to [3.35].

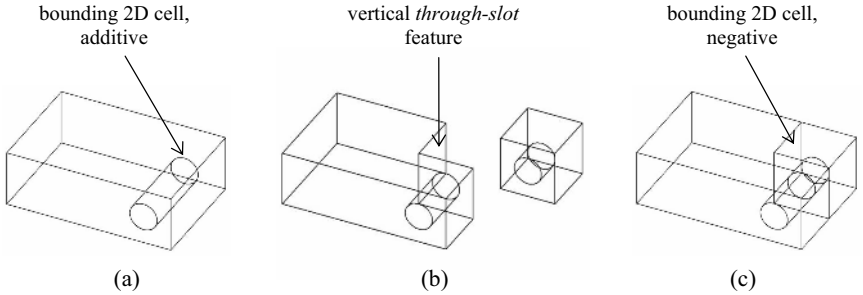

Figure 3.12. Transmutation relation: (a) insert the first feature, a *blind-hole*; (b) the second feature changes the *blind-hole* into a *through-hole*; (c) the bounding 2D cell is changed to negative due to feature interaction

References

[3.1] Ma, Y.-S. and Fuh, J., 2008, "Editorial: product lifecycle modelling, analysis and management," *Computers in Industry*, **59**(2–3), pp. 107–109.

[3.2] Varady, T., Martin, R.R. and Cox, J., 1997, "Reverse engineering of geometric models – an introduction," *Computer-Aided Design*, **29**(4), pp. 255–268.

[3.3] Benko, P., Martin, R.R. and Varady, T., 2001, "Algorithms for reverse engineering boundary representation models," *Computer-Aided Design*, **33**(11), pp. 839–851.

[3.4] Starly, B., Lau, A., Sun, W., Lau, W. and Bradbury, T., 2005, "Direct slicing of STEP based NURBS models for layered manufacturing," *Computer-Aided Design*, **37**(4), pp. 387–397.

[3.5] Kramer, T.R., Huang, H., Messina, E., Proctor, F.M. and Scott, H., 2001, "A feature-based inspection and machining system," *Computer-Aided Design*, **33**(9), pp. 653–669.

[3.6] Rezayat, M., 1996, "Midsurface abstraction from 3D solid models: general theory and applications," *Computer-Aided Design*, **28**(11), pp. 905–915.

[3.7] Ma, Y.-S., Britton, G., Tor, S.B., Jin, L.-Y., Chen, G. and Tang, S.-H., 2004, "Design of a feature-object-based mechanical assembly library," *Computer-Aided Design & Applications*, **1**(1–4), pp. 397–404.

[3.8] International Organisation for Standardization (ISO), 2000, *Industrial Automation Systems and Integration: Product Data Representation and Exchange: Integrated Generic Resource: Part 42 – Geometric and Topological Representation*, Geneva, Switzerland.

[3.9] Shah, J.J. and Rogers, M.T., 1993, "Assembly modelling as an extension of feature-based design," *Research in Engineering Design*, **5**(3–4), pp. 218–237.

[3.10] van Holland, W. and Bronsvoort, W.F., 2000, "Assembly features in modelling and planning," *Robotics and Computer Integrated Manufacturing*, **16**(4), pp. 277–294.

[3.11] Karinthi, R.R. and Nau, D., 1992, "An algebraic approach to feature interactions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **14**(4), pp. 469–484.

[3.12] Hounsell, M.S. and Case, K., 1999, "Feature-based interaction: an identification and classification methodology," *Proceedings of the Institution of Mechanical Engineers, Part B – Journal of Engineering Manufacture*, **213**(4), pp. 369–380.

[3.13] Vandenbrande, J.H. and Requicha, A.A.G., 1993, "Spatial reasoning for the automatic recognition of machinable features in solid models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **15**(12), pp. 1269–1285.

[3.14] Laakko, T. and Mantyla, M., 1993, "Feature modelling by incremental feature recognition," *Computer-Aided Design*, **25**(8), pp. 479–492.

[3.15] Bidarra, R. and Bronsvoort, W.F., 2000, "Semantic feature modelling," *Computer-Aided Design*, **32**(3), pp. 201–225.

[3.16] Schulte, M., Weber, C. and Stark, R., 1993, "Functional features for design in mechanical engineering," *Computers in Industry*, **23**(1–2), pp. 15–24.

[3.17] Feng, C.-X., Huang, C.-C., Kusiak, A. and Li, P.-G., 1996, "Representation of functions and features in detail design," *Computer-Aided Design*, **28**(12), pp. 961–971.

[3.18] Anderl, R. and Mendgen, R., 1996, "Modelling with constraints: theoretical foundation and application," *Computer-Aided Design*, **28**(3), pp. 301–313.

[3.19] Kim, C.-S. and O'Grady, P.J., 1996, "A representation formalism for feature-based design," *Computer-Aided Design*, **28**(6–7), pp. 451–460.

[3.20] Mukherjee, A. and Liu, C.R., 1997, "Conceptual design, manufacturability evaluation and preliminary process planning using function-form relationships in stamped metal parts," *Robotics & Computer-Integrated Manufacturing*, **13**(3), pp. 253–270.

[3.21] Whitney, D.E., Mantripragada, R., Adams, J.D. and Rhee, S.J., 1999, "Designing assemblies," *Research in Engineering Design*, **11**(4), pp. 229–253.

[3.22] Khoshnevis, B., Sormaz, D.N. and Park, J.Y., 1999, "An integrated process planning system using feature reasoning and space search-based optimisation," *IIE Transactions*, **31**(7), pp. 597–616.

[3.23] Stage, R., Roberts, C. and Henderson, M., 1999, "Generating resource based flexible form manufacturing features through objective driven clustering," *Computer-Aided Design*, **31**(2), pp. 119–130.

[3.24] Brunetti, G. and Golob, B., 2000, "A feature-based approach towards an integrated product model including conceptual design information," *Computer-Aided Design*, **32**(14), pp. 877–887.

[3.25] Park, S.C., 2003, "Knowledge capturing methodology in process planning," *Computer-Aided Design*, **35**(12), pp. 1109–1117.

[3.26] Brunetti, G. and Grimm, S., 2005, "Feature ontologies for the explicit representation of shape semantics," *International Journal of Computer Applications in Technology*, **23**(2/3/4), pp. 192–202.

[3.27] Rossignac, J.R., 1990, "Issues on feature-based editing and interrogation of solid models," *Computers & Graphics*, **14**(2), pp. 149–172.

[3.28] Ma, Y.-S. and Tong, T., 2003, "Associative feature modelling for concurrent engineering integration," *Computers in Industry*, **51**(1), pp. 51–71.

[3.29] Martino, T.D., Falcidieno, B., Giannini, F., Hassinger, S. and Ovtcharova, J., 1994, "Feature-based modelling by integrating design and recognition approaches," *Computer-Aided Design*, **26**(8), pp. 646–653.

[3.30] Ma, Y.-S., Chen, G. and Thimm, G., 2008, "Paradigm shift: unified and associative feature-based concurrent and collaborative engineering," *Journal of Intelligent Manufacturing*, **19**(6), pp. 625–641.

[3.31] Chen, G., Ma, Y.-S., Thimm, G. and Tang, S.-H., 2004, "Unified feature modelling scheme for the integration of CAD and CAx," *Computer-Aided Design & Applications*, **1**(1–4), pp. 595–601.

[3.32] Booch, G., Rumbaugh, J. and Jacobson, I., 1999, *The Unified Modelling Language User Guide*, Addison Wesley.

[3.33] Chen, G., Ma, Y.-S., Thimm, G. and Tang, S.-H., 2005, "Knowledge-based reasoning in a unified feature modelling scheme," *Computer-Aided Design & Applications*, **2**(1–4), pp. 173–182.

[3.34] Chen, G., Ma, Y.-S., Thimm, G. and Tang, S.-H., 2006, "Associations in a unified feature modelling scheme," *Transactions of the ASME*, *Journal of Computing and Information Science in Engineering*, **6**(6), pp. 114–126.

[3.35] Chen, G., Ma, Y.-S. and Thimm, G., 2008, "Change propagation algorithm in a unified feature modelling scheme," *Computers in Industry*, **59**(2–3), pp. 110–118.

[3.36] Hoffman, C.M., 1989, *Geometric and Solid Modelling: An Introduction*, Morgan Kaufmann, San Francisco.

[3.37] Bidarra, R., Madeira, J., Neels, W.J. and Bronsvoort, W.F., 2005, "Efficiency of boundary evaluation for a cellular model," *Computer-Aided Design*, **37**(12), pp. 1266–1284.

[3.38] Bidarra, R., de Kraker, K.J. and Bronsvoort, W.F., 1998, "Representation and management of feature information in a cellular model," *Computer-Aided Design*, **30**(4), pp. 301–313.

[3.39] Lee, S.H., 2005, "A CAD-CAE integration approach using feature-based multi-resolution and multi-abstraction modeller techniques," *Computer-Aided Design*, **37**(9), pp. 941–955.

[3.40] Sriram, R.D., Wong, A. and He, L.-X., 1995, "GNOMES: an object-oriented non-manifold geometric engine," *Computer-Aided Design*, **27**(11), pp. 853–868.

[3.41] Masuda, H., 1993, "Topological operators and Boolean operations for complex-based non-manifold geometric models," *Computer-Aided Design*, **25**(2), pp. 119–129.

[3.42] Crocker, G.A. and Reinke, W.F., 1991, "An editable non-manifold boundary representation," *IEEE Computer Graphics & Applications*, **11**(2), pp. 39–51.

[3.43] Chen, G., Ma, Y.-S., Thimm, G. and Tang, S.-H., 2006, "Using cellular topology in a unified feature modelling scheme," *Computer-Aided Design & Applications*, **3**(1–4), pp. 89–98.