

## A FEATURE INTERFACE MODEL TOWARDS DISTRIBUTED SOLID MODELING

Xiaofeng Mi, Min Tang, Jinxiang Dong

State Key Laboratory of CAD & CG, Zhejiang University, P.R.China

*Feature is a fundamental design unit in modern solid modeling systems, and this makes a well defined feature interface a matter of concernment in the component-based distributed modeling environment. This paper introduces a feature-based modeling service framework and proposes an interface model of feature-based modeling service toward such a modeling environment, including the support of remote feature attachment and semantic maintenance. A prototype-based feature interface definition strategy using procedural attaching and declarative validation mechanism is discussed in details and how the interface model solves the problems in distributed environment, such as locality-transparent model reusing, is also explained.*

**Keywords:** Solid modeling, procedural, constraint-based and parametric modeling, feature-based modeling, feature interface, distributed design, computer-aided design.

### 1. Introduction

Feature modeling has gradually become the technology in parameterized CAD system since Pratt and Wilson used form feature as the geometry description of design element in 1985 (Pratt and Wilson 1985). While the traditional feature modeling systems are usually based on single application and only a few of them take distributed design into account.

#### 1.1. Why feature interface

Consider the problem of the feature attachment request for a cross-shaped screw cap shown in Figure 1 with the decomposition hierarchy tree. In the distributed environment, the sub-features' information of the cross-shaped screw cap is not resident in its local node A, but in node B and C. The interactive graph of the remote modeling request is shown in Figure 2.

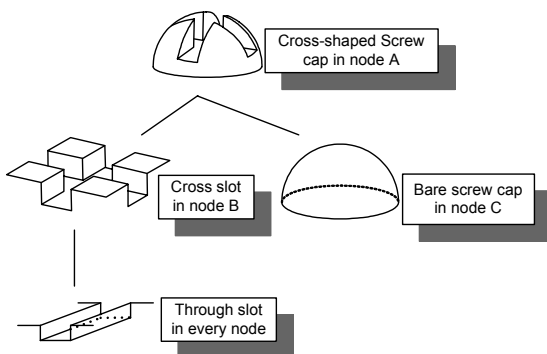


Fig. 1 Decomposition hierarchy tree of cross-shaped screw cap.

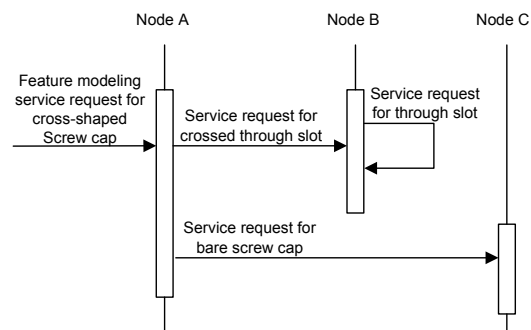


Fig. 2 Service request interactive graph for cross-shaped screw cap.

Design of a system that supports feature-based distributed modeling is a challenging work. Such a system must support the mechanisms for the functions listed below:

- 1) To locate the feature information in the distributed feature library.
- 2) Attachment and validation of third party user-defined features or compound procedurally user-defined features, remote as they might be.
- 3) Both 1 and 2 must be done with location-transparency to the users and third-party programmers.

A natural way to solution is to keep the interface of feature attachment simple and to handle the attachment and validation through the feature interface. Since current component technologies have already provided the interface-based distributed application framework, so the problem has come down to how to abstract the interface of feature attachment and in what a manner to assembly the features through the interfaces they expose to implement the distribute modeling.

## **1.2. Related Work**

A fully competent while simple feature interface should provide the ability to create, delete and manipulate the features, to validate the features by validation constraints or rules and to enable users to define their own validation constraints and rules (Shah and Mantyla, 1995). The difficulty is the variability of the feature type, especially user-defined feature.

Most implementations of modern feature-based modeling systems have already presented a well defined model-oriented feature interface.

An typical procedural implementation of features is that of Hoffmann and Joan-Arinyo's (1998). Their representation creates the user-defined feature prototype as a design process and validity checking of instantiation and attachment mechanism is provided. Bidarra and Bronsvort proposed a declarative scheme for the specification of interface (Bidarra et al., 1998). They give a full specification of feature semantics (Bidarra and Bronsvort, 2000), comprising validity issues at geometric, topologic and functional levels.

In both Hoffmann's procedural and Bidarra et al.'s declarative representation, a clear and simple feature interface is provided, which encapsulates implementation details of the class by means of so-called interface parameters. Such design is a typical paradigm of object-oriented system design and which is suitable for distribute applications.

However, a practically useful interface is not that of model-oriented, actual service must be available, i.e. implementation-oriented feature interface must also be provided. DOME (de Kraker et al., 1996), proposed by Francis P. et al. is such a distributed application architecture based on module interface. In that paper, interface means encapsulated services of a module. In addition, an interface wrapper is used to allow preexisting software programs to communicate with other modules in the objected-based modeling and evaluation discussed.

Liu's research (Liu, 2000) applies component technology into the attachment of feature. The architecture proposed by him wraps each type of feature as a MFIC that meets some predefined interface specification.

Our current research is a distributed version of Liu's system (Liu, 2000), expect that the building of most user-defined feature interface can not only be done by programming, but also by means of graphical user interface, instead of interpretative language in Hoffmann and Bidarra's researches, and thus more user-friendly.

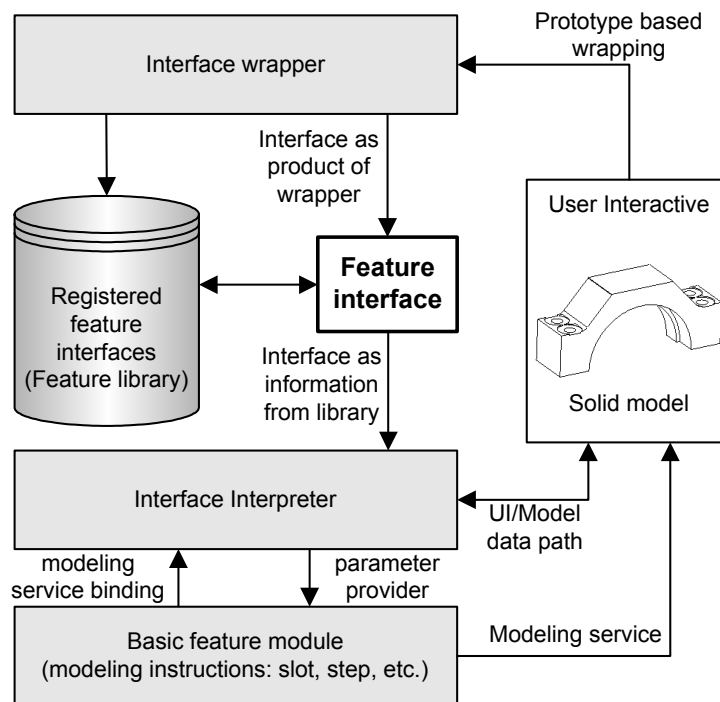
The remainder of the paper is organized as follows. First, an interpretation and then the formal definition of feature interface are given. Then a detailed discuss on how to construct a feature interface, including basic or compound feature, is presented. Next, how this kind of interface benefits distributed feature modeling and validation is discussed, together with the modeling work flow. Finally, we give an illustrative example and draw some conclusions.

## 2. What is Feature Interface

A feature is an information unit representing a region of interest within a product. As the definition of form features, they are the representations of shape aspects of a physical product that usually have generic shape and engineering semantic. Most researches use feature class or feature class frame (Shah and Mantyla, 1995; Bidarra et al., 1998; Laakko, 1993; Laakko and Mantyla, 1993; Laakko and Mantyla, ??), as a template for all instances of a given feature type, to describe all feature properties above by means of feature definition frame.

Feature interface is the exposed view of feature through which the modeling system handles the corresponding feature operations, including attaching and validation. This exposed view is generally consists of specifications on the interactions between the user and feature.

A feature interface should provide the program module with sufficient information to implement its service and can be chiefly divided into three parts, i.e. attachment input specification, shape specification and conceptual information, each representing one aspect of feature's properties, will be described in detail soon.



**Fig. 3 Framework overview.**

Note that feature interface is not the executable hard-code which presents as the programming interface in the CORBA or COM component model. The later, called by the feature interface interpreter, will be discussed further in section 3. The feature interface, in some sense, is actually the feature information service and more like a protocol between the interface interpreter and the interface wrapper. The interface wrapper is resident on every node of the distributed modeling environment and helps to register new feature types into the distributed feature library. Detailed discussion of it will be present in section 3. The overview of interface-based modeling environment and the role of feature interface in such architecture can be seen from Figure 3.

## 2.1. Components of Feature Interface

Feature interface can be divided into four parts, i.e. input specification, output specification, operation instruction and conceptual engineering information.

Input specification specifies feature attachment input and constraints on these input elements. Normally, we define them as follows.

**Definition 2.1.1** *Reference elements set*,  $D_{IN}$ , is the geometry elements referenced when attaching feature, including datum plane, datum vector and datum mark.

**Definition 2.1.2** *Algebra parameter set*, denoted by  $P_{IN}$ , is the set of feature parameters

**Definition 2.1.3** *Orientation constraints*,  $C_O$ , is the subset of  $C \times D_{IN} \times P_{IN} \times B$ ; it specifies the constraints  $D_{IN}$  and  $P_{IN}$  should meet, where  $D_{IN}$  and  $P_{IN}$  is defined as definition 2.1.1 and 2.1.2 and  $B = \{ \text{true, false} \}$  and  $C$  denotes the geometry constraints class,  $C = \{ \text{parallel, biparallel, perpendicular, co\_planar, pt\_on\_plane, co\_pos} \}$ . Possible constraint types between given two datum elements are listed in Figure 4.

**Definition 2.1.4** *Algebra constraints*,  $C_A$ , is a set of evaluation expressions with the operands element from set  $E_D \cup P_{IN} \cup R$ , where  $R$  denotes the real number set,  $E_D$  is a subset of triple  $F \times D_{IN} \times D_{IN}$  and  $F = \{ \text{angle, dist} \}$ , which means the angle or distance between datum elements. Operator of  $C_A$  can be every algebra operator supported.

	datum face	datum vector	datum point
datum face	parallel biparallel perpendicular coplanar	parallel biparallel perpendicular	pt_on_plane
datum vector	parallel biparallel perpendicular	parallel biparallel	none
datum point	pt_on_plane	none	co_pos

Fig. 4 Possible constraint types between datum elements in GS-CCAD.

**Definition 2.1.5** *Input constraints set*,  $C_{IN}$ , is defined upon  $E_{IN}$  and  $P_{IN}$  and  $C_{IN} = C_O \cup C_A$ , where  $C_O$  and  $C_A$  mean orientation constraints set and algebra constraints set, respectively.

*Output specification*, or *shape specification* of the feature, designates the feature's shape. This part of feature interface serves as the validation specification whose responsibility is to inform the interface interpreter how the attached feature should look like and what constraints should the elements of the shape meet. In GS-CCAD, four data structures are used to describe the shape specification, i.e. prototype face table, prototype edge table, prototype adjacency graph and feature-element constraint list. Detailed definition listed as follows.

**Definition 2.1.6** *Prototype face table*,  $F_{OUT}$ , is the expected faces generated by the attachment of the corresponding feature, consisting of face name, whether can intersect with other geometry elements and whether optional.

**Definition 2.1.7** *Prototype edge table*,  $E_{OUT}$ , is the expected edges generated by the attachment of the corresponding feature, consisting of edge name, whether intersectable with other geometry elements, whether optional and whether is convex edges, which, together with the prototype adjacency graph is used in the topological and geometrical validation of generated form feature.

**Definition 2.1.8** *Prototype adjacency graph*,  $G$ , is a subset of triple,  $F'_{OUT} \times E'_{OUT} \times F'_{OUT}$ , where

$F'_{OUT} = F_{OUT}[FaceKey]$ ,  $E'_{OUT} = E_{OUT}[EdgeKey]$ , representing the output faces and edges respectively. Each entry of  $F'_{OUT} \times E'_{OUT} \times F'_{OUT}$ ,  $(f_1, e, f_2)$ , means  $f_1$  and  $f_2$  is adjacent on edge  $e$ . Together with  $A_{OUT} = E_{OUT}[Attri]$ , the projection of the convex properties on the prototype edge table, it serves as the attributed adjacency graph of the feature.

**Definition 2.1.9** *Output constraints*,  $C_{OUT}$ , is the constraints defined on  $E_{IN}$ ,  $P_{IN}$ ,  $F'_{OUT}$  and  $E'_{OUT}$ , specifying the constraints the result geometry elements of the feature modeling should meets.  $C_{OUT} = C_{OOUT} \cup C_{AOUT}$ , where  $C_{OOUT}$  is the orientation constraints.

$$C_{OOUT} \subset C \times G_{ALL} \times P \times B;$$

$$G_{ALL} = F'_{OUT} \cup E'_{OUT} \cup D_{IN};$$

$$P = E_{ALL} \cup P_{IN} \cup R$$

where  $R$  is the real number set.  $P_{IN}$  is given in definition 2.1.2,  $E_{ALL}$  is a subset of triple  $F \times G_{ALL} \times G_{ALL}$  and  $F$  is given in definition 2.1.4.  $C_{AOUT}$  is the algebra constraints and is a list of evaluation expressions with operands  $P$  and all supported operator.

As the interface towards the distributed feature modeling system, the key component of the feature interface lies in the procedural definition of features. In GS-CCAD, it is embodied by feature modeling instruction sequence.

**Definition 2.1.10** A feature modeling instruction of feature  $F$ ,  $\iota \in I_{Fi} \times d_{IN} \times p_{IN} \times f'_{OUT} \times e'_{OUT}$ , where  $I_{Fi}$  denotes a sub-feature interface calling entry of  $F_i$  in the procedural modeling of  $F$ .  $d_{IN}$ ,  $p_{IN}$ ,  $f'_{OUT}$  and  $e'_{OUT}$  are instance sets of feature  $F_i$ 's  $D_{IN}$  set,  $P_{IN}$  set,  $F'_{OUT}$  set and  $E'_{OUT}$  set, the former two designating input of  $F_i$  and the remainder represents the expected results. Feature modeling instruction sequence  $\{\iota_i\}$  is a ordered sequence of feature modeling instructions and meets:

$$d_{IN_{Fi}} \subset inst(D_{IN_{Fi}}) \cup driv\left(\bigcup_{j=1}^{i-1} (d_{IN_{Fj}} \cup p_{IN_{Fj}} \cup f'_{OUT_{Fj}} \cup e'_{OUT_{Fj}})\right)$$

$$p_{IN_{Fi}} \subset inst(P_{IN_{Fi}}) \cup driv\left(\bigcup_{j=1}^{i-1} (d_{IN_{Fj}} \cup p_{IN_{Fj}} \cup f'_{OUT_{Fj}} \cup e'_{OUT_{Fj}})\right)$$

where  $driv(G)$  denotes all the geometry and algebra elements that can be derived form  $G$ . Thus the formulas above specify that the input of the modeling of  $i$ th sub-feature is either interactively input or can be derived from the modeling result of the previous  $i-1$  modeling actions.

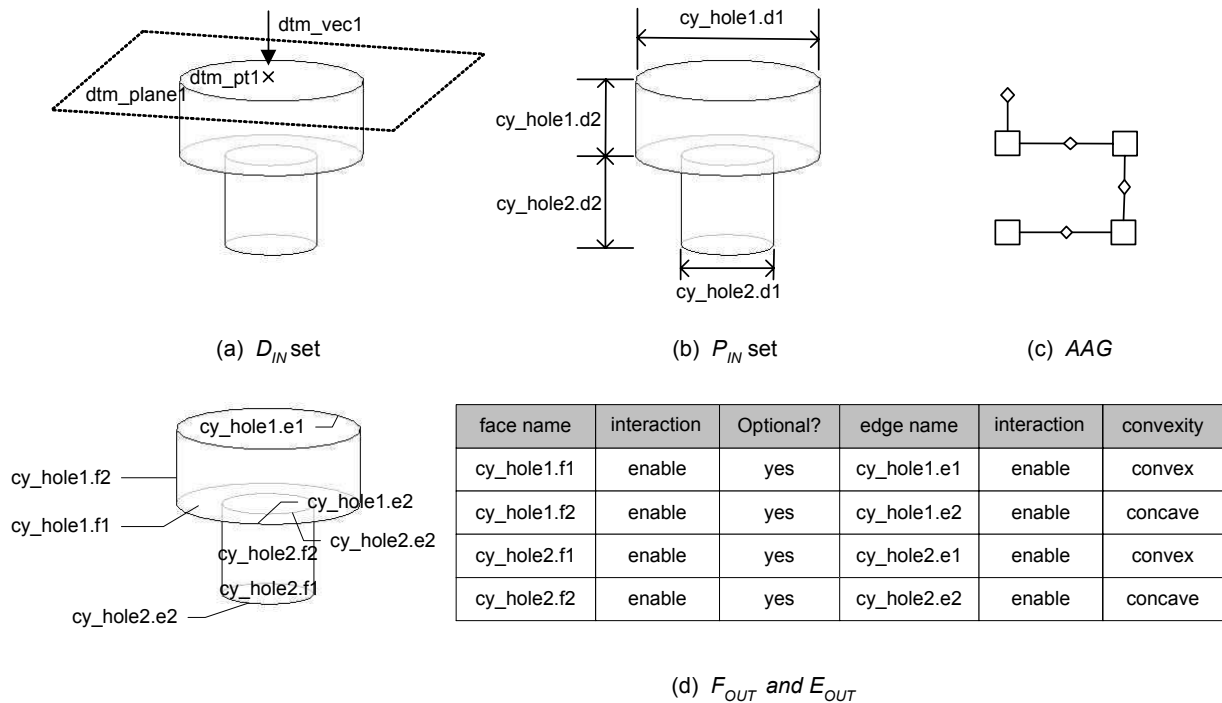
## 2.2. Definition of Feature Interface

Virtually, in GS-CCAD, feature interface provides all the retrieval interface of information on modeling and validation for interface interpreter.

**Definition 2.2.1** Feature interface is a nontuple,  $I_F = \{D_{IN}, P_{IN}, C_{IN}, F_{OUT}, E_{OUT}, G, C_{OUT}, \Gamma, \Delta\}$ , a set of user or system defined, modeling-system oriented, feature-relative information, which give a formal modeling interface for its interpreter to attachment, modification and validation of feature.

$D_{IN}$ ,  $P_{IN}$ ,  $C_{IN}$ ,  $F_{OUT}$ ,  $E_{OUT}$ ,  $G$ ,  $C_{OUT}$ , etc. are given by definition 2.1.1 through definition 2.2.9.

$\Gamma = \{\iota_i\}$ , denoting the modeling instruction sequence.



**Fig. 5 Input and output of stepped-hole.**

$\Delta$  is a set of formalized, accessible by interface interpreter description of feature's conceptual information. In GS-CCAD, this includes feature name, path in the feature library, parent interface and function information among which, name and parent provides necessary support for adding new interface into feature library in the distributed environment.

In GS-CCAD system, interface definition framework of the feature “stepped-hole” is shown in Figure 5.

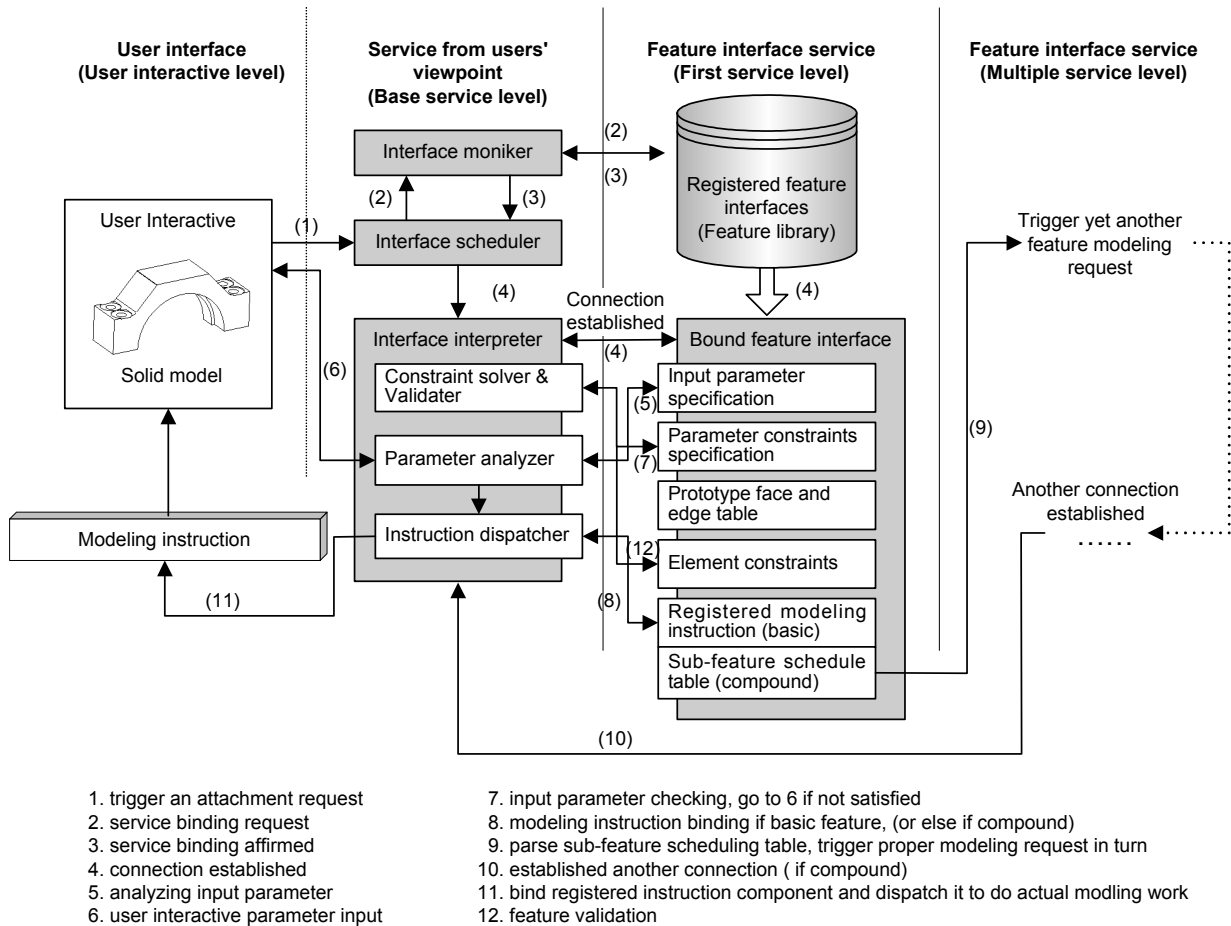
### 3. Interface-Based Distributed Modeling

After the interface of feature is properly defined, with the aid of the component technique, we can assembly the features in a proper structured modeling environment, though distributed as it may be.

#### 3.1. Framework and Workflow

As a fully qualified feature class, both attaching and validation scheme are needed. While in the distributed environment, these steps must be done with the locality transparency. Concurrent distributed object oriented programming techniques, such as CORBA from OMG and COM from Microsoft, have provide such framework support, and the feature interface definition discussed in previous sections has made the framework well extendable to a distributed version.

In the distributed modeling environment, given a feature interface key, an interface moniker is used to find the server node containing modeling service of that feature class. Then a connection is established between the modeling client and modeling server, the server may request service to yet another server, but all is done transparently including the locality of the server node (though it may be a local server!).



**Fig. 6 Interface-based feature modeling, overall framework and workflow for feature attachment and validation.**

Figure 6 shows the system framework and workflow based on the interaction between the feature interface interpreter and the feature interface discussed above. The workflow the modeling systems executes when user triggers a feature modeling request is also given with the label and the explanation in the bottom of the figure.

From Figure 6 we can see that, such system architecture simplifies the user's viewpoint towards feature modeling service. The backing up feature library, a distributed database of registered feature interfaces, is totally transparent to the end user. The local modeling system can thus naturally use the feature registered in any node in the modeling environment as long as proper connection between local interface interpreter and feature interface is established. The interoperation across the connection between the interpreter and interface can be accomplished via a daemon service resident both in client and server nodes. As for the Microsoft Windows platform, the active directory service interfaces (ADSI) technique is a good implementation tool.

Note that the modeling instruction, bound by the instruction dispatcher and serving as a component object, may not be a local component: the registry information from the feature interface just inform the dispatcher where to find the component and it is the dispatcher, under the aid of the component framework, that finds the proper modeling instruction in the distributed environment and dispatch it to do actual modeling work.

### 3.2. Modeling and Validation Component

Modeling instruction, which has already been mentioned many times in previous sections, exists in the distributed environment as a registered component object. Each of these components must contain certain implementations of modeling methods, regardless of the framework platform. That is, the local modeler has a modeling proxy object while the remote server runs a modeling stub, just as the most implementations of concurrent component platforms, COM or CORBA, do.

Unlike the modeling instruction component, validation may be accomplished via the constraint validation part of the interface interpreter. In a registered feature interface, there is always certain validation information available through the connection between the standard interface interpreter and registered interface. In a CORBA-based framework, a service control manager like component should be implemented and run in each node in the modeling environment to maintain the connection and help information transfers, while in the Microsoft windows platform, ADSI, a system service using COM object representation, is a good option to construct and manipulate the distributed feature interface registry repository

### 3.3 Compare with Traditional Feature Definition

From the attaching and validation of the features through their interface, we can see that our feature interface model is application-oriented and concerns mostly in the architecture of the distributed modeling system. However, the traditional feature definition schemes, such as the ISO 10303 (ISO, 1994; Owen, 1997; Kemmerer, 1999) standard (STEP), are mainly proposed to resolve the data incompatibility and data conversion problems.

Standardized representations of applications appear within STEP in the form of Application Protocols (AP).

Application protocols define the set of information models required for the information exchange and archiving of data for a given application domain. These APs are rigorously defined through a series of steps that include activity analysis, domain modeling and incorporation into the overall STEP information structure. With a standard representation for an application domain, data incompatibilities are eliminated. Information sharing through electronic exchange and archiving are now common practice.

However, the interface model presented in this paper mostly address the problem encountered when some modeling node in the distributed modeling environment want to use the feature model resident in another modeling node. The problem, such as how to attach a hard-coded remote feature model, is resolved by the introduction of modeling instruction in our feature interface model.

### 4. Example of Interface Wrapping

As a final inspection of our framework toward the interface-based modeling, in addition to the stepped blinded hole exemplified already, we would study one more case, the cross-shaped screw cap which is simple while well illustrative and with the decomposition tree already shown in Figure 1, to make our framework more understandable.

When the user interactively selects the feature instances, i.e. the cross slot and the bare screw cap, and trigger an compound feature interface definition command, the interface wrapper module thus assembly the feature interface following the steps described in section 3. Then passes the result to the user and enable the user modifying the interface specification as they will.

Initially, the sub-features cross slot and bare screw cap have the following input specification respectively.

```
Sub-feature cross slot:  
  cs.vecl of type datum plane
```

```

cs.vec2 of type datum plane
cs.etr_vec of type datum plane
cs.loc_pt of type datum mark
input constraints:
  { perp, cs.etr_vec, cs.vec1, true }
  { perp, cs.etr_vec, cs.vec2, true }
  { perp, cs.etr_vec, cs.vec2, true }
Bare screw cap:
  bsc.vec of type datum plane
  bsc.pt of type datum mark
input constraints: none

```

When wrapping the new feature interface of cross-shaped screw cap, additional constraints into the constraint set are listed as below.

```

{ parallel, bsc.vec, cs.etr_vec, true }
{ parallel, cs.etr_vec, bsc.pt-cs.loc_pt, true }
{ dist, bsc.pt, cs.loc_pt, = bsc.height }

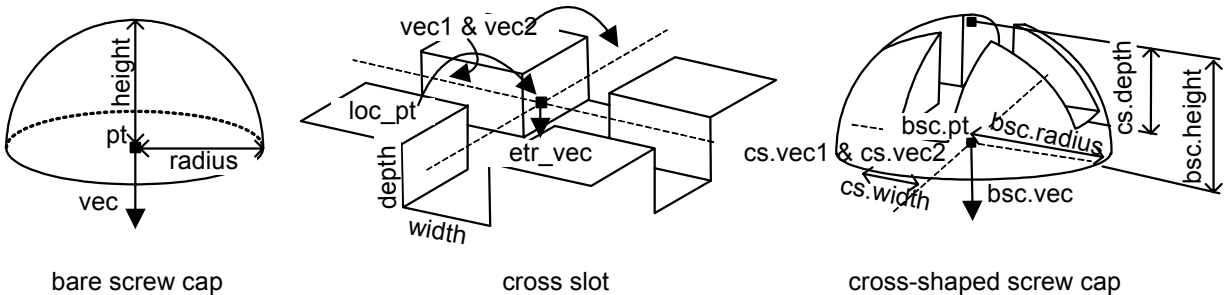
```

The eliminated input set of parameters includes cs.etr\_vec and cs.loc\_pt, with the following evaluation schemes.

```

eliminated inputs:
cs.etr_vec, evaluated by
  { para, bsc.vec, cs.etr_vec, true }
cs.loc_pt, evaluated by
  { para, cs.etr_vec, bsc.pt - cs.loc_pt, true }
  { dist, bsc.pt, cs.loc_pt, = bsc.height }

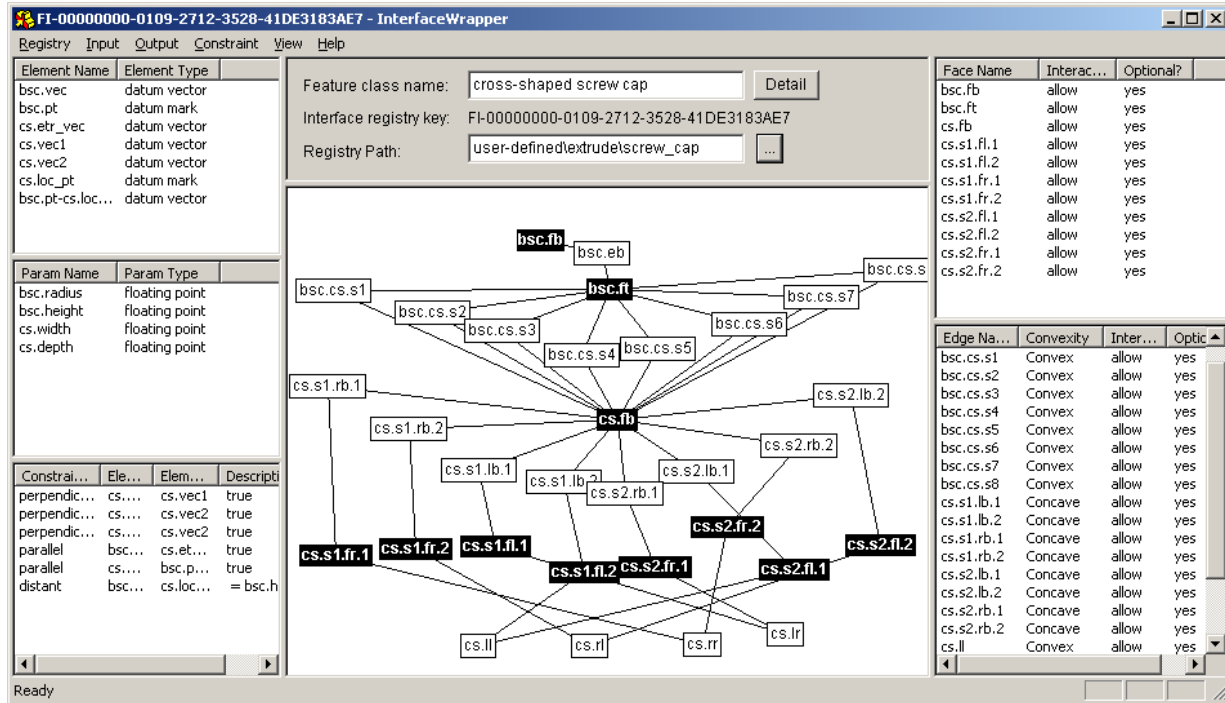
```



**Fig. 7 Source and result feature parameters.**

The result parameters are shown in Figure 7, together with the original features and parameters, a snapshot of the user interactive edit interface is shown in Figure 8.

On attaching the feature, the modeling system first find the interface of the feature via the interface moniker, then attachment scheduling table of the cross-shaped screw cap is inspected and the bare screw cap is firstly attached. Attachment scheduling table of cross slot is then inspected and two through slots, perpendicular with each other, is attached in turn.



**Fig. 8** A snapshot during the user interactive feature interface wrapping

## 5. Conclusion

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties (Szyperski, 1998).

Since we have done a lot of work on a traditional parametric feature-based CAD system, GS-CAD, the prototype of the interface-based feature modeling system is based on it. Our work, employing the Microsoft COM as the framework development tool, has solved the problem of remote feature reusability and has the following further advantages over the traditional architecture.

*Wider service sharing.* All modeling service exposed by the feature interfaces in the entire distributed modeling environment is available to the user.

*Transparency of feature interface.* As a basic feature of component-based application, the location of feature interface is transparent to the user. The user simply utilize the existing modeling service through feature interfaces in the distributed environment, and need not care where to find the attachment module of the given feature interface.

*Extendibility.* Every feature class, including the registered modeling instruction is self-dependent. That is, it has its particular function and semantic. The modeling system can only access its service via the interface exposed. If the modeling ability of the system is to be enhanced, one can register new feature interface to the feature library.

*Transparently distributed modeling.* Since the modeling instruction, the core function unit, is organized as distributed features, then the modeling work may automatically be distributed.

*Compatible with the future web service architecture.* Web service is a direction of the future software framework; the feature interface architecture proposed in this paper is well following the schema of the framework and bound to be easily transported to the web-based service framework.

There are also various problems yet to resolve. Most severe of which is that the modeling ability of the feature is restricted by the feature interface. One reason is that directory obtaining feature

parameters may inform the modeling unit more than assembly a user interface from the interface. For example, the traditional manner to get orientation parameter is make it possible to access directly the topological element of the solid model such as a face or an edge, while the orientation parameters of feature interface is a set of pure-geometry information. One solution is to enhance the type set of  $E_F$ , but it is anyway not a way home.

## 6. References

Bidarra R., Idri A, Noort A. and Bronsvort W.F., 1998, "Declarative user-defined feature classes", CD-ROM proceedings of the 1998 ASME Design Engineering Technical Conferences, 13-16 September, Atlanta, GA, USA, New York: ASME.

Bidarra R. and Bronsvort W.F., 2000, "Semantic feature modeling", *Computer-Aided Design*, Vol. 32, pp.201-225.

de Kraker, K.J., Dohmen M. and Bronsvort W.F., 1996, "Feature validation and conversion", In *CAD Tools for Products*, ed. D. Roller and P. Brunet. Springer, Berlin.

Hoffmann C.M. and Joan-Arinyo R., 1998, "On User-Defined Features", *Computer-Aided Design*, Vol. 30, No. 5, pp.331-343.

ISO, 1994, "ISO 10303:1994 - Industrial Automation Systems and Integration - Product Data Representation and Exchange", (The ISO web site is at <http://www.iso.ch/cate/cat.html> - search on 10303 to find a list of parts of the standard).

Kemmerer, S.J. (ed.), 1999, "STEP: The Grand Experience", NIST Special Publication SP 939, US Government Printing Office, Washington, DC 20402, USA.

Laakko T., 1993, "Incremental Feature Modelling: Methodology for Integrating Features and Solid models", Dissertation for the degree of Doctor of Technology at Helsinki University of Technology, Helsinki University of Technology, FIN-02150 Espoo, Finland

Laakko T. and Mäntylä M., 1993, "A Feature definition language for bridging solids and features", *Proceedings of Second Symposium on Solid Modelling and applications*, ed. J. Rossignac, J. Turner and G.Allen. ACM Press. Montreal, Canada, pp. 333-341

Laakko T. and Mäntylä M., "Feature-Based Modeling of Product Families", Helsinki University of Technology, FIN-02150 Espoo, Finland.

Liu X., 2000, "CFAXA: Component framework for feature-based design and process planning", *Computer-Aided Design*, 32, pp. 397-408.

Owen, J., 1997, "STEP: An Introduction", 2<sup>nd</sup> Edition, Information Geometers, Winchester, UK.

Pratt, M.J. and Wilson P.R., 1985, "Requirements for Support of Form Features in a Solid Modelling System – Final Report", CAM-I Report R-85-ASPP-01, Computer Aided Manufacturing-International Inc., Arlington.

Shah J.J. and Mäntylä M., 1995, "Parametric and Feature-based CAD/CAM; Concepts, Techniques and Applications", John Wiley & Sons.

Szyperski C., 1998, "Component Software, Beyond Object-Oriented Programming", ACM Press, Addison-Wesley, NJ.