Web-based direct manipulation of feature models

Rafael Bidarra André van Bunnik Willem F. Bronsvoort

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, NL-2628 CD Delft, The Netherlands
http://graphics.tudelft.nl

(R.Bidarra/W.F.Bronsvoort)@its.tudelft.nl

Abstract

Providing advanced 3D interactive facilities to users of a client-server collaborative modeling system presents a great challenge when thin clients are involved, mainly due to their lack of both a proper CAD model and the adequate modeling and solving functionality. This paper presents a new approach that provides a convenient representation of feature model data suitable for direct manipulation of feature models at such clients. It combines all advantages of a thin client approach with the sort of 3D direct manipulation facilities usually only found in powerful standalone CAD systems.

Keywords

feature modeling, graphical interaction, web-based modeling, collaborative modeling

1. INTRODUCTION

Current CAD systems, holding a sizeable modeling kernel which maintains an unabridged CAD model, provide many advanced interactive facilities for model manipulation. Requirement for this, however, is that they run on powerful, typically standalone, workstations.

Current demands for supporting design collaboration, on the other hand, require an efficient networked environment in which geographically distributed members of a development team can work together on the design of a part. In an ideal collaborative modeling framework, several team members should be able to remotely browse and manipulate a model, via Internet, *as if* they were working directly at a powerful CAD station. A web-based system, for example, would greatly facilitate this, by providing access to all sorts of product information in a uniform, simple and familiar framework. The above mentioned characteristics of current CAD systems prevent them from matching these demands.

A number of commercial tools that are now emerging provide some limited support for collaborative design activities. For example, tools for collaborative model annotation and visualization are now becoming available, providing concepts such as interactive 3D visualization, shared cameras and telepointers. However, such tools are primarily focused on visualization and inspection, basically using polygon mesh models, and do not support real modeling activities. In other words, they are valuable assistants for teamwork, but no real modeling systems.

Meanwhile, new prototype systems are being developed which directly concentrate on collaborative modeling facilities. In such systems, mostly following a client-server architecture, a crucial role is played by their complex concurrency and synchronization mechanisms.

Current commercial client-server modeling systems which offer some real collaborative modeling facilities as, for example, OneSpace [CoCreate03] and IX SPeeD [ImpactXoft03], use considerably fat clients, requiring heavy data synchronization among clients, and are severely constrained by the model format into which they convert shared CAD models.

Thin client web-based approaches, in contrast, are gaining particular attractiveness, one of the main reasons being that they usually provide a more efficient solution to data synchronization problems by using a single, server-based central model. In addition, directly loading the client application via Internet avoids complex installation and maintenance procedures, and therefore increases portability. Typically, in such systems, a development team member should be allowed to graphically specify a modeling operation, appreciate its consequences and, upon approval, issue it for execution at the server.

There are two important characteristics of thin clients in such systems which make user interactivity with the model particularly challenging.

First, they lack a real modeling kernel, and cannot therefore locally execute actual modeling operations. Instead, because such operations are executed only at the

server, this is required, after each operation, (i) to export to the clients the necessary model data (indispensable for visualization and user interaction), and (ii) to guarantee that such data is always kept up-to-date.

Second, these thin clients lack a comprehensive constraint solver. As a result, it is in general not possible to locally anticipate all consequences a given operation may have in the whole model. For example, when several features are related through geometric constraints, displacing one of them will typically affect a few others, but the overall result can only be precisely determined by means of a constraint solver.

Summarizing, a careful choice of client model data is required in order to provide thin clients with proper user interaction mechanisms.

webSPIFF, a web-based, collaborative feature modeling prototype system developed at Delft University of Technology, offers such a thin client framework. A complete description of its client-server architecture and functionality can be found in [Bidarra02]. In particular, the reader is referred to this reference for all aspects related to its consistency management, data synchronization and consistency maintenance facilities. In Section 3 only a short overview of the system is provided.

Model data used so far by the webSPIFF clients described in [Bidarra02] was mainly aimed at providing its users with (i) interactive visualization of the model, (ii) interactive selection of feature faces during operation specification, and (iii) textual information on each feature's parameter values. The main limitation of such data is that it can only be modified by means of updates received from the server, never directly by the client itself.

The goal of this project is twofold: to extend the model data at the webSPIFF clients (i) with a feature representation suitable for direct manipulation (Section 3) and (ii) with advanced interaction mechanisms supporting such feature direct manipulation (Section 4).

2. INTERACTIVE FACILITIES IN 3D WEB SYSTEMS

In this section interactive facilities offered by a few web-based 3D systems are briefly surveyed. The reader is referred to the Web3D Repository at the Web3D Consortium site [Web3D03] for an overview of other similar systems.

Kaon

The Kaon Composer [Kaon03] is a Java applet aimed at supporting virtual product presentations via Internet. It uses Kaon's Master Model native format to provide interactive visualization for zooming, panning and rotating a 3D mesh model directly in the web browser. Pre-defined regions of the model can be made sensitive to actions, as for example displaying attached annotations or triggering an animation. In addition, queries on dimensions can be also interactively performed by clicking and dragging on the model.

RealityWave

RealityWave [RealityWave03] developed VizStream Platform, a client-server technology aimed at supporting collaborative browsing and visualization of 3D models by loading a simple viewer in a web browser. In addition to the same functionality mentioned above for Kaon, VizStream provides also inspection facilities as, for example, clipping the 3D model by means of an interactively adjustable clipping-plane. The user can also attach markup to regions of his choice on the visualized model. Finally, the possibility of selecting which (and how) components of the model are visualized is also provided.

Art of Illusion

Art of Illusion [ArtOfIllusion03] is an open source studio application integrating modeling and rendering functionality. Although it is not strictly speaking web-based, we include it here as, to the best of our knowledge, it is the first fully Java-implemented 3D modeling system available. Being a moderately small application, it offers advanced direct manipulation and complex modeling operations (including face lifting and Boolean operations), comparable to those found in many commercial programs.

3. CLIENT REPRESENTATIONS FOR FEATURE MODELS

The webSPIFF server has two main components: the SPIFF modeling system and the Session Manager. The SPIFF modeling system provides all feature modeling functionality, including multiple views on a part, advanced visualization [Bronsvoort02] and validity maintenance of feature models [Bidarra00]. It maintains a central product model, which includes a cellular model for the geometric representation of a part, and canonical shapes representing the individual features in each view. The Session Manager provides functionality to start, join, leave and close a collaborative session, to coordinate the session, and to manage all communication between SPIFF and the clients. In particular, the Session Manager collects all operations requested by the various clients, and schedules them for execution at the SPIFF system.

webSPIFF clients operate locally as much as possible, e.g. regarding visualization of, and interaction with, their feature model, and only high-level messages, e.g. for specifying modeling operations, as well as a limited amount of model data necessary for updating the client information, are sent over the network. As soon as real feature model computations are required, such as for executing modeling operations, conversion between feature views and feature validity maintenance, they are executed at the webSPIFF server, on the central product model, and their results are eventually exported back to the clients. An important characteristic of this architecture is that by using a central product model, inconsistencies

are avoided among multiple versions of the model data at different clients.

Both the clients and the Session Manager of webSPIFF were implemented in Java, using its Remote Method Invocation (RMI) facilities for communication, and Java3D for model visualization.

In order to support user interaction during collaborative modeling sessions, each client receives from the server the necessary model data. This data has to be carefully derived from the feature model, in order to satisfy two somehow conflicting goals: (i) it should contain all aspects of the feature model which are relevant for direct manipulation purposes; (ii) it should be compact enough to be quickly updated in all clients whenever the model is modified at the server.

In this section, a combination of model data is presented that fulfils these goals. In particular, the new notion of *feature skeletons* is presented, and examples are given of how they represent the relations among feature instances in a model. The interactive facilities provided by feature skeletons will be dealt with in Section 4.

3.1. Graphical data

Graphical data consists of feature model images that are rendered at the webSPIFF server in GIF format, and displayed in camera windows at the clients. These images provide very powerful visualizations of a feature model [Bronsvoort02]. Many visualization options can be specified. For example, selected features may be visualized with shaded faces, and the rest of the model as a wire frame or with visible lines only. Also, additional feature information, such as closure faces of holes, can be visualized. A separate image is needed for each camera, and it must be updated every time the model or the camera settings are changed.

These model images provide the camera background on top of which other visualization and interaction techniques are available at the webSPIFF clients.

3.2. Geometric data

webSPIFF clients dispose of two representations of the model geometry: the visualization model and the selection model. Each one has a specific purpose in the camera windows at the clients [Bidarra01]. Both the visualization model and the selection model are generated by the webSPIFF server in VRML format and loaded by a client into its camera's scene graph.

The *visualization model* represents the global shape of the product model. It is used at the clients for interactively modifying the camera viewing parameters (e.g. rotating and zooming). All cameras on a particular client use the same local visualization model, but each camera displays it with its own viewing parameters.

The *selection model* is a collection of objects representing the canonical shapes of all features in a given view of the product. Its purpose is to support interactive selection of feature faces on a feature model image, during the specification of a modeling operation.

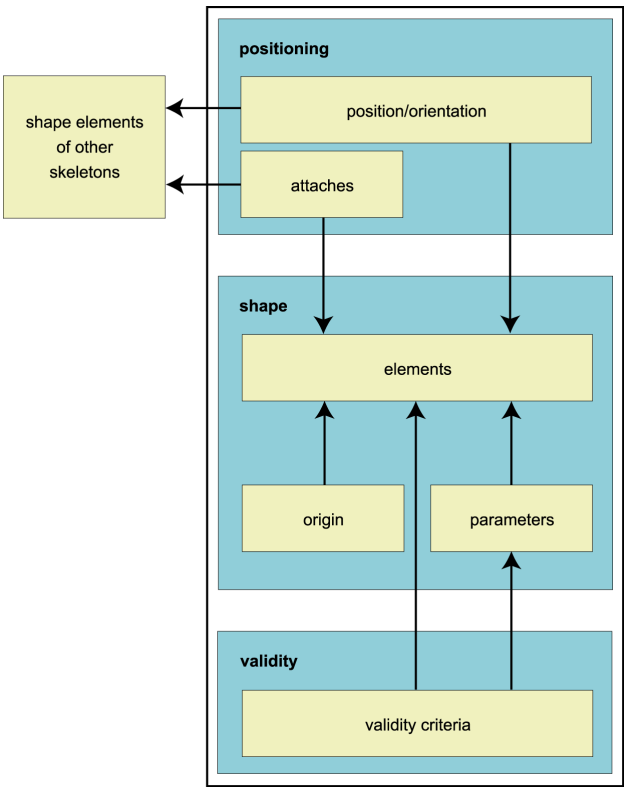


Figure 1 - Generic structure of a feature skeleton

Again, the selection model is identical for all cameras on a client, each applying its own viewing parameters.

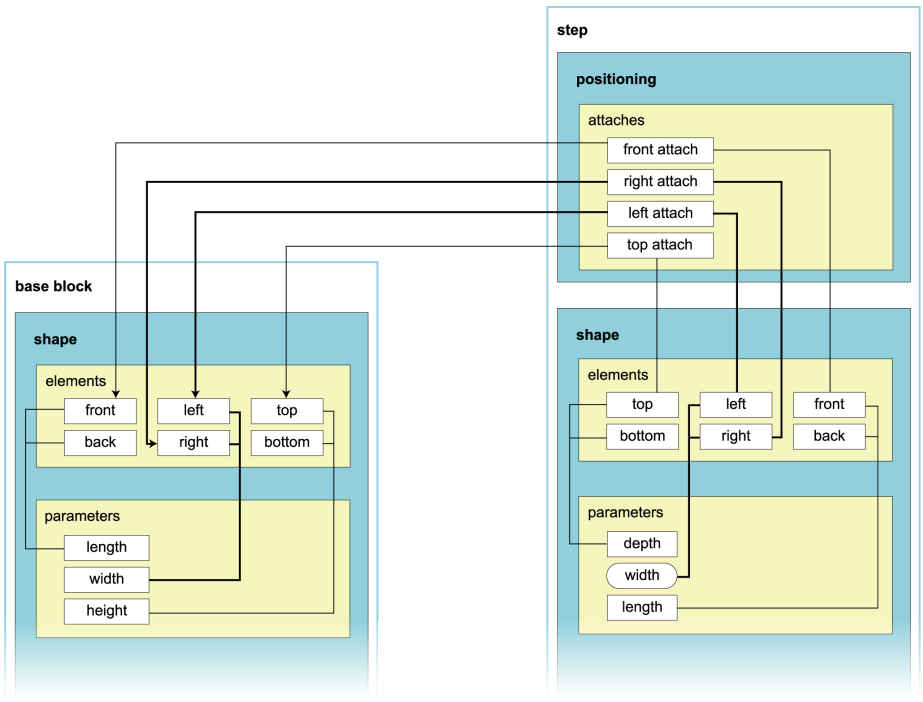
3.3. Feature skeletons

A feature skeleton is a parametric representation of a feature instance which is linked to a simplified geometric model of its shape, in such a way that the latter can be modified by interactively manipulating the former.

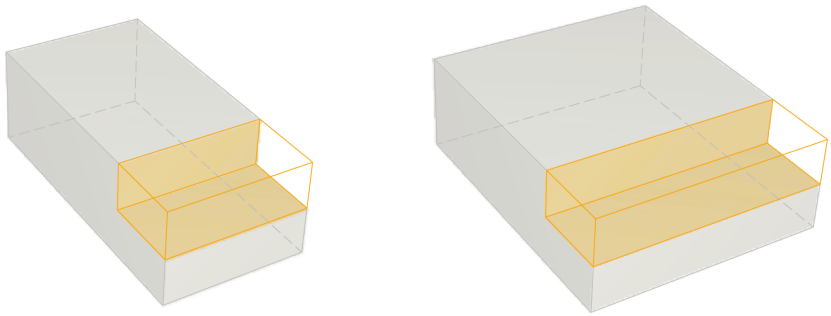
Since skeletons are meant to represent feature instances at the client, the structure of skeletons bears resemblance with the generic structure of a feature class, as described in [Bidarra00]. This structure is read from the server during client initialization, for each class in the feature library, after which the client is able to instantiate the skeleton of every feature instance in a feature model.

All skeletons consist of three main components: a *shape* component, a *positioning* component and a *validity* component; see Figure 1. The shape component describes the feature shape in terms of (i) a number of so-called *shape elements* (e.g. the axis reference and the top, bottom and side faces of a cylinder shape), (ii) a number of *parameters* (e.g. the radius and the height of a cylinder shape), and (iii) an *origin*, specified as the intersection point of some shape elements (e.g., for the cylinder shape, the intersection of the axis and the top face).

Each shape parameter conveys a relation between two shape elements. For example, the height of a cylinder shape expresses the distance between its top and bottom faces. Although several feature classes may be based on



(a) partial skeletons of a model with a block and a step features



(b) before the operation

(c) after the operation

Figure 2 - Propagation of dimension modifications between dependent features

the same shape type (e.g. block or cylinder), each feature class uses its shape in a different way. This is also reflected in the corresponding feature skeletons, and specifically, in the way the skeleton parameters may be adjusted by the user. For example, the skeletons of a blind hole and of a through hole both have a similar cylinder shape component. However, the blind hole skeleton provides two adjustable parameters, radius and depth, whereas for the through hole skeleton only the radius parameter is adjustable, its actual height being derived from the attaches of the through hole.

In short, skeleton parameters may be either adjustable or derived, and these settings are specified in each feature class, together with its own attach and positioning scheme, which will now be described.

The positioning component of a skeleton describes the geometric relations of a feature with the rest of the model. Such relations represent the attach and geometric constraints used at the server to hierarchically structure the actual feature model. An attach constraint is a kind of coplanar geometric constraint which takes into account the nature of the two features it relates in order to determine the orientation of the attach. Examples of geometric constraints are *distance-face-face* and *angle-face-face* constraints between two planar faces. Because the clients do not dispose of a geometric constraint solver, recording such relations amounts to permanently maintaining the relative position, orientation and dimensions of each feature in terms of the features to which it is explicitly related. Basically, a skeleton achieves this by relating the parameters and the origin of its shape to elements of other features by means of geometric transformations.

Among other things, this information is crucial to know which other features a given feature depends on, and thus, to allow for tracking the correct propagation of changes

when any of those features is modified. For example, Figure 2 presents a model with a rectangular step attached to a base block, together with the respective skeletons. When the block width parameter (relating its left and right faces) is increased, the step width should be increased as well, because its left and right attaches refer to those block faces. As will be explained in the next section, when a user is modifying a feature, its dependent features are also highlighted in the camera, so that possible modifications in their derived parameters become apparent to him as well.

The validity criteria referred to in Figure 1 reflect the validation constraints specified by a feature class for each of its instances. An example of these are dimension constraints, which prescribe a specific range for the value of a given feature parameter. Such criteria can be profitably used during direct manipulation of the model, to prevent the user from performing feature modifications which would turn the model invalid. It should be noted, however, that not all advanced validity criteria specified in feature classes can be maintained and assessed remotely at the clients. This is, for example, the case of most topology-related validity criteria involved in feature interaction management, which can only be properly maintained on the central model at the server [Bidarra00].

Summarizing, feature skeletons provide a compact parametric representation of the features in a model, and their relations. As a result of their integration with the feature geometry stored in the selection model, webSPIFF clients are able to support direct manipulation of the feature model, as will be discussed in the next section.

4. INTERACTIVE FEATURE EDITING FACILITIES

Feature skeletons, as described in the previous section, contain all elements required to provide interactive feature editing facilities to webSPIFF clients. To achieve this, a number of methods has been implemented at the clients that visualize a feature skeleton, including all its adjustable parameters, and allow for their interactive modification. As explained in Section 1, the main goal of this functionality is to achieve that webSPIFF users specify their modeling operations in an interactive manner and be given as much insight as possible into their result, without recurring to the server.

Direct manipulation on a skeleton is performed by means of handles, each of which is associated to exactly one parameter, so that dragging a handler adjusts the value of the corresponding parameter.

There are three sorts of handles: (i) for attaches and references, (ii) for positioning and (iii) for shape parameters. Handles for attaches and references are aimed at selecting the shape elements of other skeletons to be used in attaches and positioning references. These handles describe thus how the feature skeleton relates to those of the other features in the feature model. Handles for positioning are aimed at setting the value for positioning parameters. These handles determine the distances and angles used in skeleton position and orientation parameters. Handles for shape parameters are aimed at setting the values for the adjustable parameters of a feature skeleton.

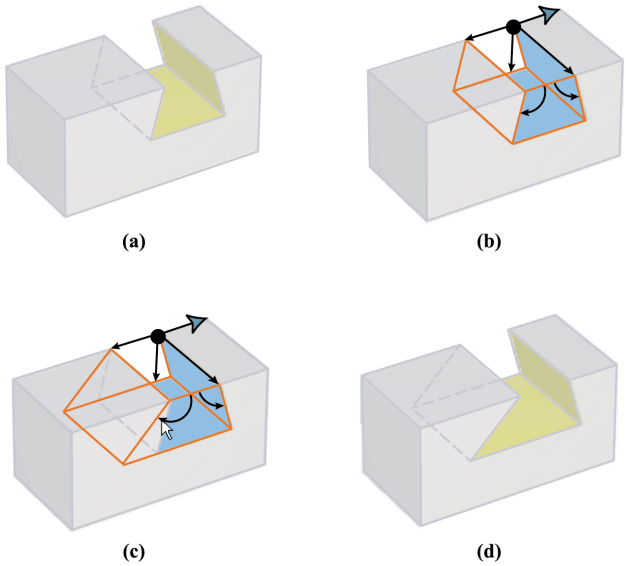


Figure 3 – Modifying a feature at the client

The basic procedure to interactively modify a feature in the model is straightforward. First, the user selects the feature by clicking it on a camera, showing a model image. If more than one feature is located behind the selected camera position, then clicking on the same position will scroll through all intersected features. The selected feature is highlighted by displaying its canonical shape (from the selection model) and, on top of it, the wireframe representing its skeleton. This wireframe contains the different sorts of handles mentioned above, which can be used to modify the feature. Dragging a handle results in the immediate modification of the skeleton. As the shaded image does not change, it is clearly visible what is the effect of changing the parameter whose handle is being dragged. When the user has finished manipulating the feature, the specified modify operation, containing the new parameters, is sent to the server, where it is executed on the central model. As a result, new model data is generated and sent back to the client, where it is visualized.

The example in Figure 3 illustrates this with a model containing a block and a trapezoidal slot, for the modification of one of the angle parameters of the slot (Figure 3.a). First, the slot is selected and its skeleton is displayed, together with its handles (Figure 3.b). Subsequently, the handle of the angle parameter between the left and top faces is dragged to change its value. The original feature model depicted in the model image remains unchanged, so that the difference between the original angle and the new angle shown in the skeleton is clearly perceptible (Figure 3.c). Finally, the operation is submitted for execution at the server, after which the

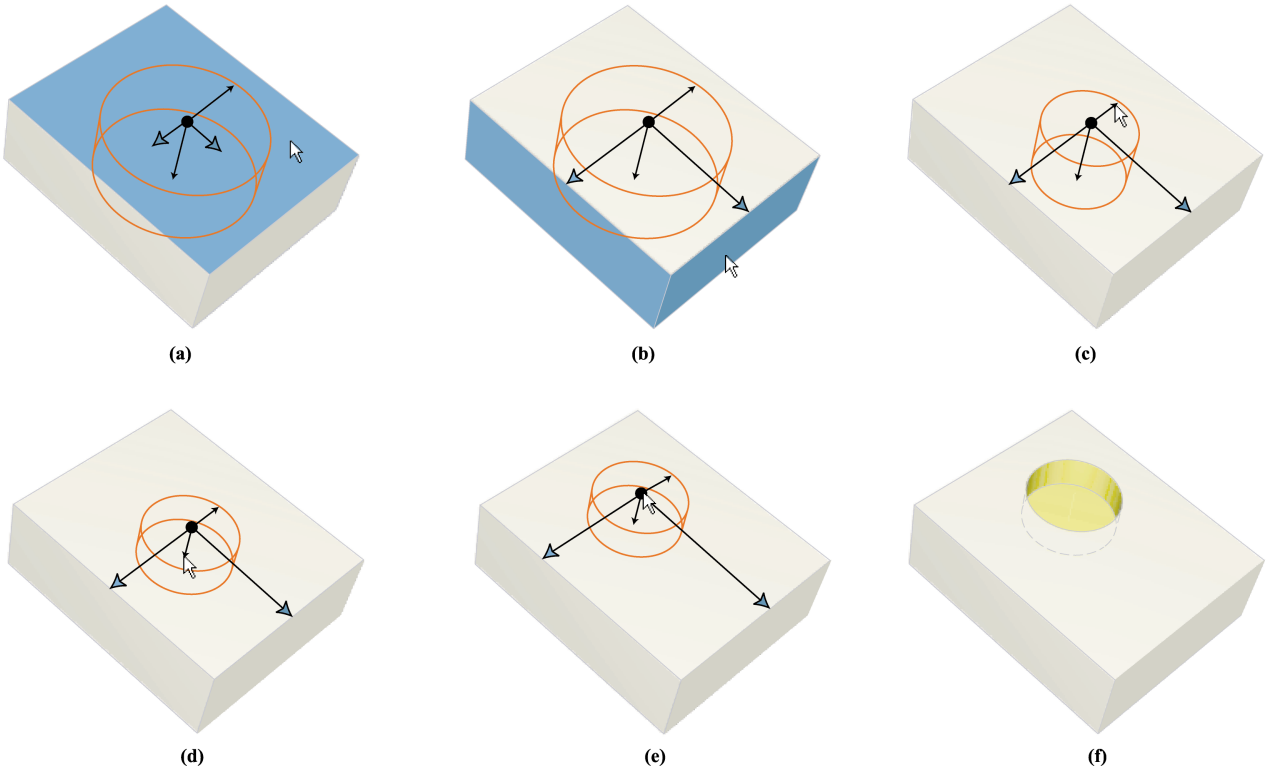


Figure 4 - Adding a new feature to the model

resulting model is displayed at the client's camera (Figure 3.d).

Adding a new feature to the current feature model is in many regards analogous to modifying an existing one. It only requires an initial step, in order to first choose the feature class of the new feature instance and to select its attach faces. Once this is done, the skeleton of the new instance is displayed, with its parameters set to their default values. The user is then required to interactively select the references required for positioning the new feature, after which its shape parameters can be adjusted as desired.

Figure 4 provides a simple example of this, illustrating how a blind hole can be attached to a base block. As soon as the desired attach face is selected on the block, the skeleton of the blind hole is displayed, exhibiting its default shape parameter values (Figure 4.a). Subsequently, the user selects the required pair of (non-parallel) reference faces relative to which the blind hole origin is positioned (Figure 4.b), in this case, the front and left face of the block. The parameters of the hole can then be adjusted by dragging the corresponding handles: dragging the radius handle closer to the origin decreases the radius of the hole (Figure 4.c), dragging the depth handle closer to the origin decreases its depth (Figure 4.d). By dragging the blind hole origin to another position on the attach face, the entire skeleton is displaced, and the parameters representing the distance to the reference faces are modified accordingly (Figure 4.e). When the user is satisfied with the settings of the blind hole, he can submit the operation for execution at the server, after which a new model image of the resulting model is displayed at his camera (Figure 4.f).

5. CONCLUSIONS

A novel approach has been presented that enables thin clients in a client-server collaborative modeling environment to provide their users with direct manipulation facilities on a feature model. These are made possible by the use of feature skeletons, a compact parametric representation of the features in a model and their relations, which are maintained at the system's thin clients. Feature skeletons can be visualized and manipulated in real-time, allowing users to locally specify a modeling operation in an interactive manner, and giving them insight into its results, prior to recurring to the modeling server for its actual execution.

Future work in this project will have to clarify whether it is always possible to determine in advance which validity criteria can be handled at the client and which do not.

6. REFERENCES

[ArtOfIllusion03] Eastman, P. Art of Illusion. www.artofillusion.org, May 2003.

[Bidarra00] Bidarra, R. and Bronsvoort, W.F. Semantic feature modelling. *Computer-Aided Design*, **32**(3): 201–225, 2000.

[Bidarra01] Bidarra, R., van den Berg, E. and Bronsvoort, W.F. Interactive facilities for collaborative feature modeling on the web. Proceedings of 10th EPCG, Madeira, J., Marques, J.S., Dias, M.S., and Jorge, J.A. (Eds.), pp. 43-52, October 2001.

[Bidarra02] Bidarra, R., van den Berg, E. and Bronsvoort, W.F. A Collaborative feature modeling system. *Journal of Computing and Information Science in Engineering* **2**(3): 192-198, 2002.

[Bronsvoort02] Bronsvoort, W.F., Bidarra, R. and Noort, A. Feature model visualization. *Computer Graphics Forum* **21**(4): 661-673, 2002.

[CoCreate03] CoCreate Software Inc., Fort Collins, CO, USA. www.cocreate.com, April 2003.

[ImpactXoft03] ImpactXoft, San Jose, CA, USA, www.impactxoft.com, January 2003.

[Kaon03] Kaon Interactive Inc. Maynard, MA, USA, www.kaon.com, April 2003.

[RealityWave03] RealityWave Inc. Cambridge, MA, USA, www.realitywave.com, April 2003.

[Web3D03] The Web3D Repository. www.web3d.org/vrml/vrml.htm, May 2003.