Interactive facilities for collaborative feature modeling on the Web

Rafael Bidarra Eelco van den Berg Willem F. Bronsvoort

Faculty of Information Technology and Systems
Delft University of Technology
Mekelweg 4, NL-2628 CD Delft, The Netherlands
http://www.cg.its.tudelft.nl

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

Abstract

Collaborative modeling systems are distributed multiple-user systems that are both concurrent and synchronized, aimed at supporting engineering teams in coordinating their modeling activities. During a collaborative modeling session, several users are connected to each other in order to perform activities, such as design, manufacturing planning or evaluation, together, using some common product data. An interesting research challenge is to develop a collaborative modeling system that offers all facilities of advanced modeling systems to its users.

This paper focuses on the interactive modeling facilities required by such collaborative modeling systems. Previous work in the area of collaborative modeling is surveyed, and several techniques for interaction with feature models are presented, ranging from display of sophisticated feature model images to interactive selection facilities, which have been implemented in the web-based collaborative feature modeling system webSPIFF. It has a client-server architecture, with an advanced feature modeling system as a basis of the server, providing feature validation, multiple views and sophisticated visualization facilities.

The architecture of webSPIFF, the functionality of the server and the clients, their communication mechanisms, and the distribution of model data is described. In particular, maintenance and synchronization of model data at the clients, and techniques for their effective utilization for enhancing user interaction and collaboration are described. It is shown that a good compromise between interactivity and network load has been achieved, and that indeed advanced modeling with a collaborative system is feasible.

Keywords

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

1. INTRODUCTION

In the last decade, research efforts in the areas of solid and feature modeling substantially contributed to the improvement of computer-aided design (CAD) systems. A broad range of advanced modeling facilities is now becoming available in high-end commercial systems, amplified by continuous enhancements in interactive and visualization capabilities, and profiting from the availability of faster and more powerful hardware. Still, these improvements have their counterpart in the increasing size and complexity of such systems. At the same time, a number of research prototypes are pushing the edge to even more advanced modeling facilities. For example, embodiment of richer semantics in feature models and validity maintenance of such models [Bidarra and Bronsvoort 2000], and physically-based modeling techniques [Kagan *et al.* 1999] are among the current research issues.

A common characteristic of most current CAD systems is that they run on powerful workstations or personal computers. Interaction with the system is usually only possible if the user is directly working at the CAD station, although remote interaction is sometimes possible through a high-bandwidth local area network. This situation is no longer satisfactory, as nowadays more and more engineers, often at different locations, are getting involved in the development of products. It would be preferable if a user could remotely browse and manipulate a model, via Internet, as if he were working directly at a powerful CAD station. A web-based system would be ideal for this, as it would facilitate access to all sorts of product information in a uniform, simple and familiar framework.

Even more attractive would be the support of collaborative modeling sessions, in which several geographically distributed members of a development team could work together on the design of a product. Typically, in such collaborative sessions, different participants would be provided with their own, application-specific views on the product model according to the analyses or activities required, e.g. detailed design, manufacturing planning or assembly planning [de Kraker *et al.* 1997; Hoffmann and Joan-Arinyo 1998]. In addition, each session participant, as in normal development teams, should be given his own competence and specific session privileges by the system.

So far, only a small number of tools have been developed that somehow support collaborative design activities. For example, tools for collaborative model annotation and visualization via Internet are now becoming available, providing concepts such as shared cameras and telepointers [Autodesk 2000; Parametric 2000; Kaon 2001]. However, such tools are primarily focused on inspection, e.g. using simple polygon mesh models, and do not support real modeling activities. In other words, they are valuable assistants for teamwork, but no real CAD systems. Some more recent research is focusing on the possibility of enhancing existing CAD systems with collaborative facilities; see Section 2. To the best of our knowledge, the only commercial system currently offering some collaborative modeling facilities is OneSpace [CoCreate 2000]. However, its modeling capabilities are severely constrained by the modeler at the server, SolidDesigner, and by the model format into which it converts all shared models.

The idea of collaborative modeling combines very well with the increasingly popular concept of Application Service Providers (ASP), in which clients remotely access, via Internet, specialized applications running on a server, being billed exclusively for the service time they spend logged on at the ASP server. Such an approach has been identified as a very promising and affordable alternative for distributed CAD teams [Comerford 2000]. The first rudimentary commercial CAD ASP has recently been launched by [CollabWare 2000].

In order to satisfy all requirements outlined above, it is an interesting research challenge to develop a modeling system that offers all facilities of advanced feature modeling systems to its users, while at the same time providing them with the necessary coordination mechanisms that guarantee an effective collaboration. With this goal in mind, a new web-based, collaborative feature modeling system has been developed at Delft University of Technology. A complete description of its client-server architecture and functionality can be found in [Bidarra *et al.* 2001], including solutions to the critical concurrency and synchronization problems that characterize collaborative design environments.

This paper concentrates on the facilities for interaction with feature models of this prototype system. In particular, it describes the models that are maintained by the clients, and how these models can be effectively used for visualization and user interaction. The paper is organized as follows: first, the main research issues of collaborative modeling are briefly surveyed (Section 2); second, an overview of the web-based, collaborative modeling system is given (Section 3); third, several facilities for interactive visualization of the product model, in particular of its features, are described (Section 4); fourth, techniques are presented for interactively selecting and using feature entities in the specification of modeling operations (Section 5); finally, some results and conclusions are presented (Section 6).

2. PREVIOUS WORK

Collaborative systems can be defined as distributed multiple-user systems that are both concurrent and synchronized. *Concurrency* involves management of different processes trying to simultaneously access and manipulate the same data. *Synchronization* involves propagating evolving data among users of a distributed application, in order to keep their data consistent.

These concepts being in general already rather demanding, their difficulty becomes particularly apparent within a collaborative modeling framework, where the amount of model data that has to be synchronized is typically very large, the concurrent modeling actions taking place may be very complex, and the requirements for real time display and user interaction are so acute. This section briefly surveys collaborative modeling, highlighting the key aspects put forward by recent research, and summarizing the lessons learned from a few prototype systems proposed so far.

2.1. Client-server architecture

The requirements for concurrency and synchronization in a collaborative modeling context lead almost inevitably to the adoption of a *client-server* architecture, in which the server provides the participants in a collaborative modeling session with the indispensable communication, coordination and data consistency tools, in addition to the necessary basic modeling facilities. For a recent survey on client-server architectures, see [Lewandowski 1998].

A recurrent problem in client-server systems lies in the conflict between limiting the complexity of the client application and minimizing the network load. In a collaborative modeling context, client complexity is mainly determined by the modeling and interactive facilities implemented at the client, whereas network load is mainly a function of the kind and size of the model data being transferred to/from the clients.

A whole range of compromise solutions can be devised between the two extremes, so-called *thin clients* and *fat clients*. A pure thin-client architecture typically keeps all modeling functionality at the server, which sends an image of its user interface to be displayed at the client. Clicking on the image generates an event, containing the screen coordinates of the interface location the user clicked on. This event is sent to the server, which associates it with an action on a particular widget. Eventually, this action is processed, and an updated image of the resulting user interface is sent back to the client, where it is displayed. This approach requires a continuous information stream between server and clients, and is therefore very expensive in terms of network traffic. The response time would be intolerably high for many model specification actions, thus making it very ineffective to remotely participate in a modeling session.

On the other extreme, a pure fat client offers full local modeling and interaction facilities, maintaining its own local model. Communication with the server is then often required in order to synchronize locally modified model data with the other clients. In a collaborative environment where clients can concurrently modify local model data, preventing data inconsistencies between different clients becomes a crucial problem. In addition, fat clients, to be effective, place on the platform running them the heavy computing power requirements of typical CAD stations.

2.2. Current research prototype systems

Several collaborative modeling prototype systems have recently been described in literature. Some of these systems will be shortly surveyed here, and their shortcomings identified.

CollIDE [Nam and Wright 1998] is a plug-in for the Alias modeling system, enhancing it with some collaborative functionality. Users of CollIDE have private workspaces, where model data can be adjusted independently from other users. In addition, a shared workspace exists containing a global model, which is synchronized between all users participating in a collaborative modeling session. Users can simply copy model data between the private and shared workspaces, in order to create and adjust certain model data locally, and add it to the model in the shared workspace. The architecture of CollIDE poses severe restrictions to crucial collaborative modeling issues. In particular, no special measures have been taken to reduce the amount of data sent between the participants of a collaborative modeling session, resulting in delayed synchronization of the shared workspace and of the users' displays. Also, since each user operates on a separate instance of the modeling system, able to perform modeling operations by itself, concurrency has to be handled by the users themselves in order to keep shared model data consistent. A positive point in CollIDE is, however, that it provides its users with most interactive and visualization facilities of the native system Alias, although this comes at the cost of a very fat client.

The ARCADE system [Stork and Jasnoch 1997] defines a *refine-while-discussing* method, where geographically distributed users can work together on a design, interacting with each other in real-time. Every participant uses a separate instance of the ARCADE modeling system, and all ARCADE instances are connected to a session manager via Internet. A message-based approach was chosen, where every change of the product model is converted into a short textual message, which is sent to all other instances of ARCADE through the session manager. ARCADE provides a collaborative environment in which the network load is kept low. This was done by including all modeling functionality in the distributed ARCADE instances, which exchange only textual messages, rather than large sets of polygons. A drawback of this approach, however, is that the user application becomes rather complex, thereby requiring much computational power. In addition, ARCADE provides a primitive concurrency control mechanism, where only one user can edit a particular part at a time.

CSM, the Collaborative Solid Modeling system proposed by [Chan *et al.* 1999] is a web-based collaborative modeling system. Within its client-server architecture, the server contains a global model, while every client owns a local copy of this model. When a user has locally modified the model, it is propagated to all other users through the server. Concurrency is managed in two ways: (i) the model can be locked, using token passing, restricting it from being accessed by other users as long as some user is performing a modeling operation; and (ii) functionality can be locked, preventing certain functions from being used by particular users. Clearly, such methods provide a very strict concurrency handling policy. In fact, they turn the clients into several independent modeling systems, just using the same product model alternately. In a truly collaborative modeling system, one expects a higher level of coordination support.

NetFEATURE [Lee *et al.* 1999] claims to be a collaborative, web-based, feature modeling system. A server provides basic functions on a central product model, including creation and deletion of features. On the clients, a local model is available, containing a boundary representation of the product, derived from the server-side central model. The local model is reportedly used for real-time display, navigation and interaction, although very little has been described by the authors on how this operations take place. For more advanced operations, the server must be accessed. Updating the local model is done incrementally, which required a rather heavy naming scheme. This scheme severely reduces the modeling functionality of the system, degrading it to a history-based geometric modeling system, instead of a genuine feature modeling system. Furthermore, NetFEATURE uses, just like CSM, very strict concurrency handling methods, thus seriously limiting genuine collaborative modeling.

2.3. Conclusions

Collaborative modeling systems can support engineering teams in coordinating their modeling activities. Instead of an *iterative* process, sending product data back and forth among several team members, designing becomes an *interactive* process, in which several engineers are simultaneously involved to agree on design issues. Collaborative modeling systems typically have a client-server architecture, differing in the distribution of functionality and data between clients and server.

Concurrency control is still a crucial issue in current collaborative environments. If a user is allowed to change a model entity, while another user is already changing the same entity, problems can easily arise concerning consistency. To avoid this situation, a strict concurrency control mechanism can limit access for other users. It depends on the application, whether all entities of the design should be locked or just some of them. If possible, users should be allowed to simultaneously modify different parts of the design, but this could lead to much more complicated concurrency control mechanisms. Also, one should always bear in mind that designing is a constructive activity. Users can therefore be given some responsibility for establishing a good collaboration.

Current systems also often fall short in adequately handling synchronization of model data among distributed clients. Timely updating data over a network is difficult, since there is a certain delay between the moment data is sent and the moment it is received at another node of the network; during this time interval, the latter might try to manipulate data that is not up-to-date. Mechanisms to detect such conflicts should be available, and recovery mechanisms provided. Good locking can also help to avoid such situations, but sometimes it may hinder users' flexibility.

For a collaborative CAD system to be successful, it should combine a good level of interactivity with the sort of powerful visualization typically provided by conventional CAD systems. Users will not be able to design properly if they have to wait a considerable time after every operation. But increasing interactivity by just porting more and more data and functionality to the clients is not a good solution either, as synchronization problems would then turn critical. Furthermore, fat clients are typically platform-dependent applications that require more complex installation and maintenance procedures, and are therefore less practical in a web-based context.

In conclusion, a good compromise solution to the difficulties summarized above can better be achieved with a web-based, client-server architecture. The next section introduces webSPIFF, a prototype system that follows this approach.

3. ARCHITECTURE OF webSPIFF

webSPIFF has a client-server architecture, consisting of several components; see Figure 1. On the server side, two main components can be identified: the SPIFF modeling system, providing all feature modeling functionality; and

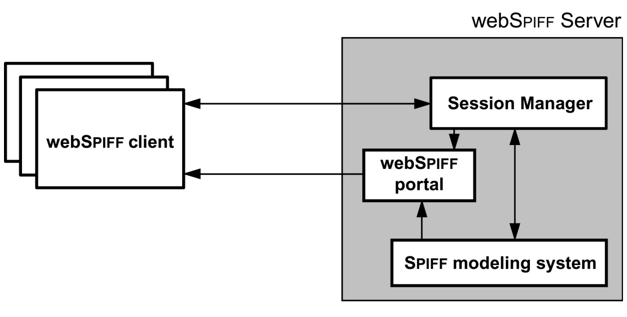


Figure 1. Architecture of webSPIFF

the Session Manager, providing functionality to start, join, leave and close a modeling session, and to manage all communication between SPIFF and the clients. The webSPIFF portal component provides the initial access to a webSPIFF session for new clients, and includes a web server where model data is made available for download by the clients.

The clients perform operations locally as much as possible, e.g. regarding visualization of, and interaction with, the feature model, and only high-level semantic messages, e.g. specifying modeling operations, as well as a limited amount of model information necessary for updating the client data, are sent over the network.

The server coordinates the collaborative session, maintains a central product model, and provides all functionality that cannot, or should not, be implemented on the client. In particular, as soon as real feature modeling computations are required, such as required by modeling operations, by conversion between feature views and by 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 advantage of this architecture is that there is only one central product model in the system, thus avoiding inconsistency between multiple versions of the same model.

3.1. The server

As a basis for the server, the SPIFF system developed at Delft University of Technology was chosen, which offers several advanced modeling facilities. First, it offers *multiple views* on a product model, each view consisting of a feature model with features specific for the application corresponding to the view. The current version of webSPIFF provides two such views: one for design and another for manufacturing planning of parts. In the design view, the feature model consists of both additive (e.g. protrusions) and subtractive (e.g. slots and holes) features. In the manufacturing planning view, the feature model consists of only subtractive features. All views on a product model are kept consistent by feature conversion [de Kraker *et al.* 1997]. Second, it offers *feature validity maintenance* functionality. This can guarantee that only valid feature models, i.e. models that satisfy all specified requirements, are created by a user [Bidarra and Bronsvoort 2000]. Third, it offers *sophisticated feature model visualization* techniques, which visualize much more specific feature information than most other systems do. For example, feature faces that are not on the boundary of the resulting object, such as closure faces of a through slot, can be visualized too [Bronsvoort *et al.* 2001]. All these facilities are computationally expensive, and require an advanced product model, including a cellular model with information on all features in all views [Bidarra *et al.* 1998].

The Session Manager stores information about an ongoing session and its participants. It manages all information streams between webSPIFF clients and the SPIFF modeling system. Since several session participants can send modeling operations and queries to the webSPIFF server at the same time, concurrency must be handled at the Session Manager. It is also the task of the Session Manager to synchronize session participants, by sending them the updated data structures, after a modeling or camera operation has been processed. The Session Manager has been implemented using the Java programming language [Sun Microsystems 2000].

3.2. The clients

The clients of webSPIFF make use of standard web browsers. When a new client connects to the webSPIFF portal, a Java applet is loaded, implementing a simple graphical user interface (GUI), from which a connection with the Session Manager is set up. Different clients can connect from various locations, local through a network or remote via Internet, in order to start or join a modeling session.

Once connected to the server, the user can join an ongoing collaborative session, or start a new one, by specifying the product model he wants to work on. Also, the desired view on the model has to be specified. Information on the feature model of that view is retrieved from the server, and used to build the client's GUI, through which the user can start active participation in the modeling session.

The bottom line is obviously that clients should be able to specify modeling operations in terms of features and their entities; for example, a feature, to be added to a model, should be attachable to entities of features already in the model (e.g. faces and datums). After a feature modeling operation, with all its operands, has been fully specified, the user can confirm the operation. The operation is then sent to the server, where it is checked for validity and scheduled for execution. Notice that this can result in an update of the product model on the server, and thus also of the feature model in the view of each session participant.

In addition to the above functionality, several visualization and interactive facilities of the SPIFF system have also been ported to the clients. All these facilities of webSPIFF clients make use of so-called *camera* windows, i.e. separate windows in which a graphical representation of the product model is shown. Visualization facilities include displaying of sophisticated feature model images, rendering of a 3D model that supports interactive modification of camera viewing parameters (e.g. rotation and zoom operations) and a collaborative, shared camera. These will be dealt with in Section 4. webSPIFF cameras also provide facilities for interactive specification of modeling operations, e.g. assisting the user in selecting features or feature entities by having them picked on a sophisticated image of the model. The interactive functionality of webSPIFF cameras is described in detail in Section 5.

3.3. Distribution of model data

As explained above, only one central product model is maintained at the server. This feature model includes all canonical shapes, representing individual features in a specific view, and the cellular model. To support visualization and interaction facilities, however, the webSPIFF clients need to locally dispose of some model data. This data is derived by the server from its central model, but it does not make up a real feature model. webSPIFF clients need just enough model information in order to be able to autonomously interact with the feature model, i.e. without continuously requesting feedback from the server.

Model data at the clients can be classified into the following three categories:

Textual data

This data is used for specific sets of model information, mostly in list form. The most important are:

- *List of feature classes:* contains the names of all feature classes available in a given view. It is used to fill a GUI list widget when adding a new feature instance, and is requested from the server at client initialization time. This list does not need to be refreshed during a modeling session.
- *List of feature instances:* contains the names of all feature instances in a given view of the model. It is used to fill a GUI list widget when editing or removing an existing feature instance. This list is set upon initialization of the client, and is refreshed after each modeling operation.
- *List of parameter values:* contains the values of all parameters of a given feature instance, in a predefined order. It is used to fill various GUI entry widgets when editing the selected feature instance, and is always queried before a feature editing operation.

Graphical data

This comprises the *sophisticated feature model images*, rendered at the SPIFF server, and displayed in camera windows at the clients. These images provide very powerful visualizations of a feature model, as explained in Subsection 4.1. Many visualization options can be specified. A separate image is needed for each camera, and it must be updated every time the model or the camera settings are changed. The images are stored in the web server to be downloaded by the respective clients.

Geometric data

This comprises two kinds of models: the visualization model and the selection model.

- *visualization model:* represents the global shape of the product model. The visualization model is generated by SPIFF, and it is used at the clients for interactively changing the camera viewing parameters, as explained in Subsection 4.2. All cameras on a particular client use the same local visualization model, but each camera displays it with its own viewing parameters.

- *selection model:* is a collection of three-dimensional objects representing the canonical shapes of all features in a given view of the model. It is also generated by SPIFF, with the purpose of supporting the interactive selection of feature faces on a client's camera, during the specification of a modeling operation, as explained in Section 5. Again, the selection model is identical for all cameras on a client, each applying its own viewing parameters.

Client model data is never modified directly by the clients themselves. Instead, after a modeling operation has been sent to the server and executed, updated model data is sent back to the client. In addition, if the central product model has been changed, appropriate updated model data is sent to all other session participants as well. This consists of, possibly several, new model images, a new visualization model, and an incremental update of the selection model (containing only new and/or modified feature canonical shapes).

Similarly, when a client modifies any camera settings, the corresponding camera operation is sent to the webSPIFF server, which generates a new sophisticated feature model image. Since the feature model remains unaffected by camera operations, the server only needs to send the new sophisticated feature model image back to the client that requested the camera update.

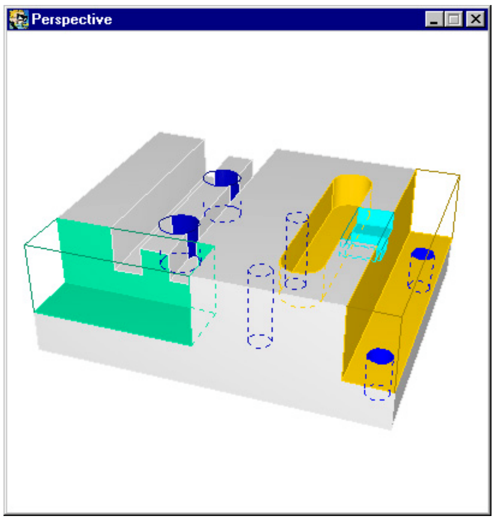
Temporary local inconsistencies can still occur at a client, for example after the execution of a modeling operation. Since sending information from the server to all clients takes some time, for a short period model information on the clients is not up-to-date. Avoiding conflicts arising from this transitory mismatch is one of the main tasks of the Session Manager.

4. PRODUCT MODEL VISUALIZATION

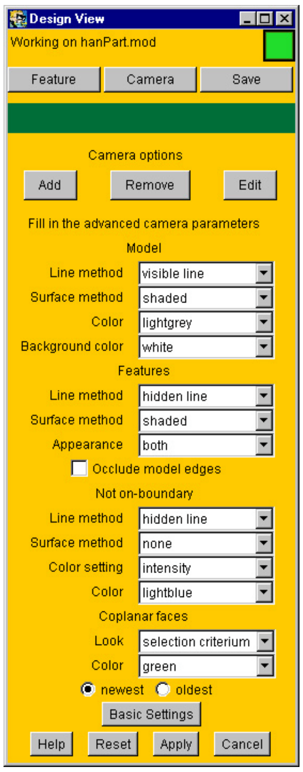
As anticipated in Subsection 3.2, webSPIFF provides clients with three ways of visualising a product model, all making use of *camera windows*. Each client may create as many cameras as desired. First, a sophisticated feature model image can be displayed. Second, a model can be rendered that supports interactive modification of camera viewing parameters, e.g. rotation and zoom operations. Third, shared cameras may be created that support collaborative, synchronized visualization of the model among several users.

4.1. Sophisticated feature model images

The most powerful visualisation technique generates *sophisticated feature model images*, which can very effectively support the user during the modeling process. These camera images provide not only a plain visualisation of the resulting final shape of the product model. Several advanced visualisation techniques are available that allow the user to customise the images to a variety of



(a) camera window displaying a sophisticated feature model image, highlighting the closure edges for some selected features



Working on hanPart.mod

Fill in the advanced camera parameters

Model

Line method	visible line
Surface method	shaded
Color	lightgrey
Background color	white

Features

Line method	hidden line
Surface method	shaded
Appearance	both

☐ Occlude model edges

Not on-boundary

Line method	hidden line
Surface method	none
Color setting	intensity
Color	lightblue

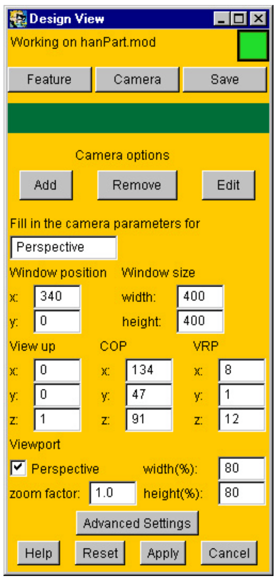
Coplanar faces

Look	selection criterium
Color	green

[✓ newest] oldest

Basic Settings

(b) model visualization parameters



Working on hanPart.mod

Fill in the camera parameters for

Perspective

Window position: x: 340 y: 0

Window size: width: 400 height: 400

	View up	COP	VRP
x	0	134	8
y	0	47	1
z	1	91	12

Viewport

☑ Perspective width(%): 80

zoom factor: 1.0 height(%): 80

Advanced Settings

(c) viewing parameters

Figure 2. webSPIFF camera panels

needs [Bronsvoort *et al.* 2001]. Sometimes, a user wants to have a closer look at a particular feature in a model, e.g. because he wants to fine-tune its parameters. Using different visualisation techniques for a selected feature and the rest of the model, extra insight into the selected feature is offered, e.g. on its shape and location in the model. For example, the selected feature may be visualised with shaded faces, and the rest of the model as a wire frame or with visible lines only. As already mentioned in Subsection 3.1, also additional feature information, such as closure faces of subtractive features, can be visualised. The facilities for rendering such images make extensive use of the ACIS Modeling Kernel [Spatial 2000], and are therefore not available on the clients. Instead, the images are rendered by the SPIFF modeling system, and sent by the Session Manager to the clients, where they are displayed in a camera; see Figure 2(a).

In addition to the settings for the above-mentioned techniques, several viewing parameters (such as center of projection, view reference point, projection type, etc.) can be set per camera. The camera panels of the webSPIFF GUI offer the clients functionality to specify and modify any camera parameter by means of menus, control boxes and checkboxes; see Figure 2(b) and (c). Interactive specification of the viewing parameters will be elaborated in Subsection 4.2.

The sophisticated image displayed in a camera has to be updated whenever the client modifies any of its camera parameters. Similarly, the image no longer reflects the current state of the model when any session participant has modified the model. In both cases, the image is regenerated on the server and resent to the client(s). Both GIF and JPG image formats provide satisfactory results; see Figure 2(a) for a sophisticated image example. Sending an image from the server to a client is therefore very cheap, both in terms of network load (approximately 10 Kbytes) and display time at the client.

4.2. Interactive visualization model

As described in the previous subsection, changing the viewing parameters of a camera can be done by specifying values for them using the camera panels. This is convenient, for example, when the user wishes to position the viewing camera at an exact location. Often, however, it is much more practical to be able to position and orient the viewing camera in 3D space in an interactive way, using the mouse, as usual in most CAD systems. As the mouse moves, the viewing parameters are modified continuously according to the mouse events generated, creating a smooth animation. Rendering a sophisticated feature model image at the server, and sending it back to a client, takes quite some time, which makes it impossible to update the sophisticated image in real time: the time elapsed between arrival of two successive images at the client would simply be too long, hindering smooth interaction.

The requirement for graphical interaction led to the introduction of a *visualization model*, which is a polygon mesh, generated at the server in VRML format [Ames *et al.* 1997], and sent to the client, where it is loaded into a Java3D scene. Unlike the cellular model on the server, it contains no information about features, except possibly different color attributes for faces originating from different features. Figure 3 shows a camera window with an image of a visualization model.

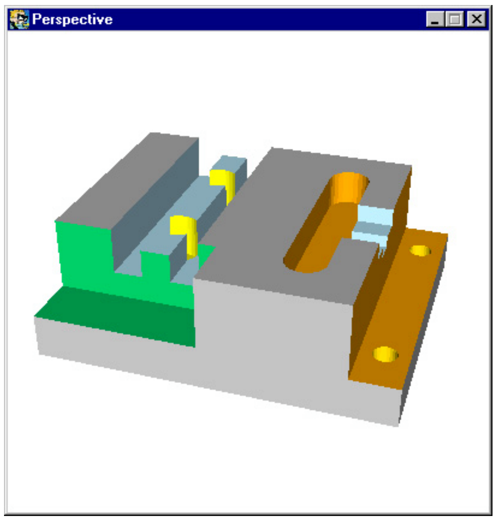
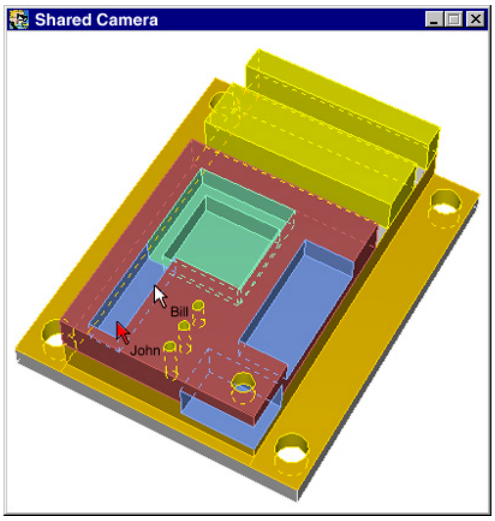


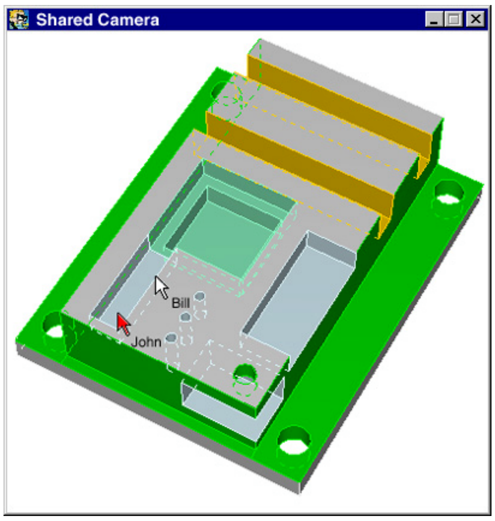
Figure 3. Camera window displaying the visualization model of the part used in Figure 2

By maintaining the visualization model at the clients, viewing cameras can be interactively oriented and positioned in virtual space as follows. As default, a sophisticated feature model image is shown in a camera window, while the visualization model is hidden. When a mouse button is pressed on the camera, the sophisticated feature model image disappears, and the visualization model is displayed instead. The user can then interactively adjust the viewing parameters, until the camera has the desired position and orientation. After being confirmed by the user, the new camera parameters are sent to the server, which in turn generates a new sophisticated feature model image according to the new parameters. This is then delivered back to the client, where it is displayed in the camera window, hiding the visualization model again.

The visualization model only needs to be regenerated by the server and updated at the clients whenever any user modifies the product model. Sending the VRML file to the clients is reasonably cheap in terms of network load (in the order of 100 Kbytes for a moderately complex feature model). Taking into account these file sizes, it can be questioned whether compressing them before transmission to the clients would further improve system throughput, due to the overhead introduced by the compression and decompression algorithms. It would probably be more effective to use techniques for incremental or progressive transmission of the VRML data; see, for example, [Gueziec *et al.* 1999].



(a) design view



(b) manufacturing planning view

Figure 4. Shared cameras at two users with different views on the same product model

4.3. Shared cameras

In a collaborative modeling environment, synchronous communication channels play an important role. Among the various techniques available for supporting collaborative modeling work, one of the most effective is the use of *shared cameras*. In a shared camera, several distributed users share the same viewing parameters on the visualized product model. These parameters are permanently synchronized, so that everytime one user modifies any of them, the shared cameras of all other users are automatically updated.

In webSPIFF, shared cameras were very easily implemented, because every client already disposes locally of its own visualization model, introduced in the previous subsection. The only requirement here is the propagation to all users of the changing viewing parameters, so that these can be adjusted for the local shared camera. This is very effectively handled using the Remote Method Invocation (RMI) facilities of Java, via the Session Manager, which receives from any client the modified parameters, and forwards them to the remaining users.

In addition to the above, webSPIFF provides each user of a shared camera with a personalized *telepointer*. The telepointers of all participants in a shared camera session are also constantly updated in all shared cameras. In this way, e.g. when discussing on some local geometric detail (typically using some phone conferencing facility), participants in a shared camera can always precisely trace back where each interlocutor is pointing at.

Another advantage of this use of shared cameras within the multiple-view feature modeling framework of webSPIFF is that what each participant sees rendered in his shared camera is (the visualization model of) his own view specific feature model. Figure 4 illustrated this, showing two shared cameras in the same session. The user in (a) is working on the design view, and hence his shared camera visualizes the design view of the product model; the user in (b), however, is working on the manufacturing planning view, and sees therefore the feature model of that view in his shared camera. Although each one 'sees' only feature instances that are meaningful within his own view on the product, the presence of telepointers facilitates focusing their dialogue on a mutual region of interest for their discussion.

5. INTERACTIVE SELECTION OF FEATURE ENTITIES

As explained in Subsection 3.2, an essential characteristic of the SPIFF system is that its modeling operations are specified in terms of features and feature faces. The interface of webSPIFF clients provides a panel for the specification of feature modeling operations, presenting the required menus filled with appropriate names (e.g. of all features, or of all faces of a particular feature). The user can then browse through these names to specify the operands of modeling operations, as he might do when working directly at a CAD station running the SPIFF system.

However, graphical interaction is very useful, not only for visualization of the product model, as described in Section 4, but also for assisting the user in selecting model entities, specifically for modeling operations. In fact, it is often much more convenient to graphically select those entities directly on an image of the visualized product model than from a menu.

For this, the *selection model* was introduced at the web clients. It consists of a set of feature canonical shapes, each of which comprises a number of uniquely named entities, in particular the feature faces. Each canonical shape is generated at the server into a separated file in VRML format, and loaded into the Java3D scene at the client.

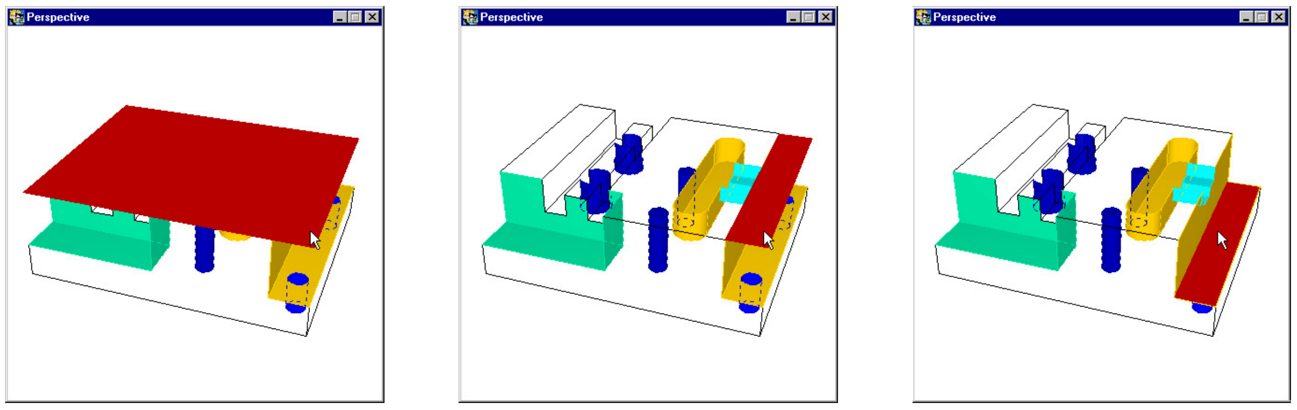


Figure 5. Selection of the step bottom face using the selection model

The canonical shapes in the selection model are never fully displayed simultaneously. Instead, they are kept invisible in the camera, until the user selects a point with the mouse. At that moment, a conceptual ray is determined from the position of the selected point and the viewing parameters used to generate the image. The feature faces intersected by the ray are subsequently highlighted for possible selection, until the user confirms the selection of one of them; see Figure 5 for an example. Notice that in this way also feature faces can be selected that are (partly or totally) not on the boundary of the resulting object, as shown in Figure 5.(b).

When the camera viewing parameters have been modified, the canonical shapes of the selection model do not need to be updated at the client: the only thing needed is to visualize them according to the new viewing transformation, similarly to what is done with the visualization model. A canonical shape only needs to be regenerated by the server, and reloaded by the client, when the parameters or the position of the corresponding feature are modified, as a result of some modeling operation. Sending VRML files of the canonical shapes to the clients is again cheap in terms of network load (in the order of 5 Kbytes per canonical shape).

6. CONCLUSIONS

This paper discussed a number of user interaction facilities suitable for web-based, collaborative feature modeling. These have been implemented in the new collaborative modeling system webSPIFF, which has a client-server architecture. The webSPIFF server runs on a HP B180L Visualise workstation. So far, webSPIFF clients running on Unix, Windows and Linux platforms have successfully participated in collaborative sessions. The only requirement at the client side is a Java/Java3D-enabled web browser. The webSPIFF portal has a demo version available on Internet for users to experiment with, at www.webSPIFF.org.

webSPIFF provides a powerful framework for investigating many issues involved in collaborative feature modeling systems, including synchronization, concurrency and user interaction aspects. The proposed distribution of functionality between the server and the clients has resulted in a well-balanced system. On the one hand, the full functionality of an advanced feature modeling system is offered by the server. On the other hand, all desirable interactive modeling functionality is offered by the clients, ranging from display of sophisticated images of feature models to interactive selection facilities. The Java-based client application is quite simple, and a good compromise between interactivity on the clients and network load has been achieved.

As Internet technology rapidly improves, faster and better collaboration becomes possible. It can therefore be expected that, although the development of collaborative modeling systems is still at its early stages, such systems will soon play an important role in the product development process.

7. REFERENCES

Ames, A., Nadeau, D. and Moreland, J. (1997) The VRML 2.0 Sourcebook. Second Edition, John Wiley & Sons, New York

Autodesk (2000) AutoCAD 2000 Online. Autodesk Inc., San Rafael, CA, USA. *http://www.autodesk.com*

Bidarra, R., van den Berg, E. and Bronsvoort, W.F. (2001) Collaborative modeling with features. TBP in: CD-ROM Proceedings of the ASME 2001 Design Engineering Technical Conferences, Pittsburgh, PA, USA, ASME, New York

Bidarra, R. and Bronsvoort, W.F. (1999) Validity maintenance of semantic feature models. Proceedings of Solid Modeling '99 – Fifth Symposium on Solid Modeling and Applications, Bronsvoort, W.F. and Anderson, D.C (Eds.), ACM Press, NY, pp. 85–96

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

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): 301–313

Bronsvoort, W.F., Bidarra, R. and Noort, A. (2001) Feature model visualization. *Submitted for publication*

Chan, S., Wong, M. and Ng, V. (1999) Collaborative solid modeling on the WWW. Proceedings of the 1999 ACM Symposium on Applied Computing, San Antonio, CA, pp. 598–602

CoCreate (2000) Shared engineering. http://www.cocreate.com/onespace/documentation/whitepapers/shared_eng.pdf

CollabWare (2000) An introduction to GS–Design Beta. https://www.prodeveloper.net/downloads/whitepaper.pdf

Comerford, R. (2000) Software, piecewise. *IEEE Spectrum*, **37**(2): 60–61

Gueziec A., Taubin, G., Horn, B. and Lazarus, F. (1999) A framework for streaming geometry in VRML. *IEEE Computer Graphics and Applications*, **19**(2): 68–78

Hoffmann, C.M. and Joan-Arinyo, R. (1998) CAD and the product master model. *Computer-Aided Design* **30**(11): 905–918

Kagan, P., Fischer, A. and Bar-Yoseph, P.Z. (1999) Integrated mechanically-based CAE System. Proceedings of Solid Modeling '99 – Fifth Symposium on Solid Modeling and Applications, Bronsvoort, W.F. and Anderson, D.C (Eds.), ACM Press, NY, pp. 23–30. Also in: *Computer-Aided Design*, **32**(8/9): 539–552

Kaon (2001) HyperSpace-3DForum. Kaon Interactive Inc., Cambridge, MA, USA. *http://www.kaon.com*

de Kraker, K.J., Dohmen, M. and Bronsvoort, W.F. (1997) Maintaining multiple views in feature modeling. Proceedings of Solid Modeling '97 – Fourth Symposium on Solid Modeling and Applications, Hoffmann, C.M. and Bronsvoort, W.F. (Eds.), ACM Press, NY, pp. 123– 130

Lee J.Y., Kim, H., Han, S.B. and Park, S.B. (1999) Network-centric feature-based modeling. Proceedings of Pacific Graphics '99, Kim, M.-S. and Seidel, H.-P. (Eds.), IEEE Computer Society, CA, pp. 280–289

Lewandowski, S. (1998) Frameworks for component-based client/server computing. *ACM Computing Surveys*, **30**(1): 3–27

Nam, T.J. and Wright, D.K. (1998) CollIDE: A shared 3D workspace for CAD. Proceedings of the 1998 Conference on Network Entities, Leeds. *http://interaction.brunel.ac.uk/~dtpgtjn/neties98/nam.pdf*

Parametric (2000) Pro/ENGINEER 2001*i*. Parametric Technologies Corporation, Waltham, MA, USA. *http://www. ptc.com*

Stork, A. and Jasnoch, U. (1997) A collaborative engineering environment. Proceedings of TeamCAD '97 Workshop on Collaborative Design, Atlanta, GA, pp. 25–33

Spatial (2000) ACIS 3D Modeling Kernel, Version 6.2. Spatial Technology Inc., Boulder, CO, USA. *http://www.spatial. com*

Sun Microsystems (2000) The Sun Java™ Technology Homepage. *http://java.sun.com*