Web-based collaborative modelling with SPIFF

Eelco van den Berg

Delft, September 2000

TU Delft
Delft University of Technology
Web-based collaborative modelling with SPIFF

Eelco van den Berg

Supervisors:
Dr. W.F. Bronsvoort
Dr.eng. R. Bidarra

Delft University of Technology
Faculty of Information Technology and Systems
Chair of Computer Graphics and CAD/CAM
Abstract

The opportunities offered by Internet technologies have resulted in numerous new types of applications, among which collaborative systems. Collaborative systems are distributed multiple-user systems that are both concurrent and synchronised. During a collaborative session, several users are connected to each other in order to perform activities, such as design or evaluation, together.

Several techniques for interaction with feature models, ranging from display of sophisticated feature model images to interactive selection facilities, have been implemented in webSPIFF, a new web-based, collaborative modelling system. webSPIFF is based on the multiple-view feature modelling system SPIFF.

In particular, attention has been given to the maintenance of model data at the clients, and their effective utilisation for enhancing user interaction and collaboration. All real feature modelling computations are executed on a central feature model, maintained at the server computer. A session manager on the server acts as an intermediate between the SPIFF modelling system and the web clients that remotely operate the modelling system.

Distribution of functionality between the server and the clients has resulted in a well-balanced web-based, collaborative feature modelling system. On the one hand, the server offers the full functionality of the original feature modelling system. On the other hand, all desirable interactive modelling functionality is offered by the clients. A good compromise between interactivity on the clients and network load has been achieved.
Preface

This thesis contains a description of webSPIFF, a new collaborative, web-based feature modelling system. I have developed webSPIFF as the final part of my graduation project. The graduation project was performed at the group Computer Graphics and CAD/CAM of the Faculty of Information Technology and Systems of Delft University of Technology.

The new modelling system is based on the existing feature modelling system SPIFF, developed by the group. The purpose of this project was to allow SPIFF to be concurrently used by different users at different geographical locations, using different platforms, via Internet.

I wish to thank Wim Bronsvoort and Rafa Bidarra for skilfully guiding me through the project. I must also thank Alex Noort, who was always prepared to help me tackle SPIFF-related problems, and whose input has been of great value.

Contents

Abstract i

Preface ii

Contents iii

1. Introduction 1

2. Collaborative modelling systems 3
   2.1 Collaborative modelling systems 3
   2.2 Existing collaborative modelling systems 3
   2.3 Conclusions 8

3. The SPIFF modelling system 10
   3.1 Feature modelling 10
   3.2 Structure of the SPIFF modelling system 10
   3.3 Multiple views and validity maintenance 10
   3.4 Models 12
   3.5 Cameras 14

4. Architecture of webSPIFF 16
   4.1 Alternatives for architecture 16
   4.2 Components of webSPIFF 17
   4.3 Models 18
   4.4 Data communication 20

5. The server 22
   5.1 Extensions to the SPIFF modelling system 22
   5.2 The Session Manager 22
   5.3 Data structures 24
   5.4 Concurrency handling 25

6. The Web Clients 29
   6.1 Structure 29
   6.2 User interface 29
   6.3 Data representation 31
   6.4 The Communications Manager 32
   6.5 Interactive facilities 33
   6.6 Synchronisation 35
7. Implementation aspects 38
   7.1 Using Java3D 38
   7.2 Using sockets 40
   7.3 Communication between Session Manager and SPIFF 41
   7.4 Security issues in Java 41

8. Examples of collaborative modelling with webSPIFF 43
   8.1 Session operations 43
   8.2 Camera interaction 44
   8.3 Collaborative modelling 45

9. Conclusions and recommendations 48

References 50
1. Introduction

Due to the emergence of wide-area networks, information sources have become much more accessible in recent years. The opportunities offered by Internet technologies have resulted in numerous new types of applications, among which collaborative systems.

Collaborative systems are distributed multiple-user systems that are both concurrent and synchronised. During a collaborative session, several users are connected to each other in order to perform activities, such as evaluation or design, together.

Concurrency involves different processes trying to access an object simultaneously. Measures have to be taken to prevent and handle problems that arise during such situations. If not handled correctly, concurrency problems could, for example, result in inconsistent data structures, used by the different processes of a concurrent system.

Synchronisation is the handling of any type of data changes caused by one of the users of a distributed application. Synchronisation involves propagating changes from one user to other users, but it can also involve propagating changes from a server computer to one or more clients.

Global manufacturing communities will soon be offering services to users via Internet. Users will therefore be able to make use of these services from any location, provided they have access to Internet. The purpose of these shared services is to take away the need for individual users to invest in expensive machinery. Some members of the global manufacturing communities will provide services for designing products, while others will let users perform all sorts of analysis on a product design. Eventually, the design can be sent to a remote manufacturing service, possessing all sorts of specialised manufacturing equipment.

Collaborative modelling systems will be an integral part of the new global manufacturing communities that are being formed (van den Berg 1999). They are computer-aided design (CAD) systems that let users work on a model simultaneously from geographically distributed locations, offering them the possibility to develop, discuss and evaluate a model together. During a traditional modelling process, the design would have to go through several iteration steps in order to be evaluated by different experts. Using a collaborative modelling system, the design could be evaluated in a single session, thus achieving considerable time reduction.

The purpose of the project described in this thesis was to develop a framework that allows several users to collaboratively work on a product model through the Internet, using a feature modelling system.

Feature modelling is a relatively new concept in CAD. In feature modelling, a model is defined by several features, such as through holes and rectangular pockets, each in turn described by a number of parameters.

It was desirable for the new system to be web-based. This means that the system can be run from within any Internet browser, not requiring preliminary installation of a special software system. Internet browsers are convenient tools to navigate on Internet, so making a system web-based considerably increases its accessibility.
Considering the aspects mentioned above, a web-based collaborative feature modelling system, called webSPIFF, has been implemented. It is based on the existing SPIFF system, which is a prototype feature modelling system developed at Delft University of Technology. The SPIFF system offers several advanced modelling 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 (Bronsvoort et al. 1997). Second, it offers advanced feature validity maintenance functionality. This guarantees that only valid feature models are created by a user (Bidarra and Bronsvoort 2000). Third, it offers feature model visualisation techniques that visualise specific feature information (Bronsvoort et al. 2000).

The webSPIFF system has a client/server architecture. All real feature modelling computations are executed on a central feature model, maintained at the server computer. A session manager on the server computer acts as an intermediate between the modelling system and the web clients that remotely operate the modelling system.

The web clients have been implemented using the platform-independent Java™ (Sun Microsystems 2000) programming language. A Java compiler translates platform-dependent text files into platform-independent byte code. The byte code can then be used on every platform for which a Java Virtual Machine is available. All recent versions of the most popular Internet browsers come with plug-ins for Java, thus making it feasible to develop web-based systems in Java. The web clients also use Java3D, an extension to Java, providing extra functionality for showing and interacting with three-dimensional scenes and models.

This thesis describes several aspects of webSPIFF. In Chapter 2, the concept of collaborative modelling and some existing collaborative modelling systems will be looked at. In Chapter 3, the concept of feature modelling and the SPIFF system will be described. In Chapter 4, an overview will be given of the architecture of webSPIFF. In Chapter 5 and Chapter 6, a closer look will be taken at the server and client sides of the system. In Chapter 7, some interesting issues that were encountered during implementation will be discussed. In Chapter 8, examples of collaborative modelling with webSPIFF will be shown. Finally, in Chapter 9, some conclusions and recommendations will be given.
2. Collaborative systems

Research on collaborative systems has been going on for a short period of time only, and therefore, the number of existing collaborative modelling systems is relatively small. In this chapter, collaborative modelling systems are explained and the architectures of some existing research prototype collaborative modelling systems are shown. Also some recommendations for developing successful collaborative systems are given. This chapter is based on (van den Berg 1999).

2.1 Collaborative modelling systems

Collaborative systems are capable of hosting sessions, with multiple participants interacting with the system from geographically distributed locations. Their task is to assist groups of designers in co-ordinating their modelling activities. Instead of sending updated designs to each other sequentially, the designers can evaluate or modify the product simultaneously, reducing the design time and increasing the quality of design.

For a modelling system to be called collaborative, it has to be both concurrent and synchronised. This means that all participants of a session must be able to work on the same model, and changes made to the model by one user must be propagated to all other users immediately. For a collaborative session to pass by free from problems, both concurrency and synchronisation must be handled. If this is not done correctly, users will be working on models that are inconsistent with the model used by the other participants of a modelling session.

2.2 Existing collaborative modelling systems

2.2.1 CollIDE

Collaborative Industrial Design Environment (CollIDE) (Nam and Wright 1998) provides a shared 3D workspace for several users of stand-alone CAD applications. It is designed as a plug-in for the existing 3D CAD system Alias and provides working areas for users, called stages, where the actual modelling takes place. The architecture of CollIDE is shown in Figure 2-1.

A shared stage, including a shared model and a shared camera, is available to all connected Alias users. It is implemented as an extra window on top of the original application and provides a synchronous view to all users. The shared model is kept in a shared database, while a session manager manages the concurrent states of user windows and provides feedback of user actions to all users.

All users have private stages too, each of which has its own local model. Model information can be copied from shared to private stages and vice versa by drag-and-drop using the mouse pointer. When model information is copied from a private stage to a shared stage, this information is sent to the shared database and then propagated to the shared stages of all other instances. Another user can then copy that information to his private stage.
The principles of shared and private stages can be used to create add-ons to more CAD systems, allowing users to keep their own modelling environment, while collaborating with other designers using different design tools.

To avoid confusion among co-workers, colours are used to point out characteristics, such as ownership. A part of the program called Awareness Support displays information about the user state. It also notifies a user about conferencing events and manages the shared telepointers. Telepointers are mouse pointers that can be seen by all users in the shared stage. Every user can have a personal telepointer too.

Observation showed that interaction between co-workers, by means of the shared stage or conferencing tools, happened almost continuously, but was most frequent at the start and at the end of a collaborative session. Communication was necessary to divide the work at the start and to integrate it at the end. Users also frequently communicated just before moving data to and from shared stages. This demonstrated the need for conferencing tools.

Testing of the CollIDE/Alias system in a collaborative environment showed some problems involving concurrency. It was quite difficult to manage the concurrent state of client windows. There was a significant delay in the view of the shared stage, especially when the complexity of the geometry reached a certain level. No solutions for these problems were given, but it does show that synchronisation should be optimised by, for example, reducing the size of the models or minimising the interaction between system components.

### 2.2.2 ARCADE

The objective for the developers of the Advanced Realism CAD Environment (ARCADE) (Stork and Jasnoch 1997) was to create a refine-while-discussing method to be used over the Internet. Refine-while-discussing means that several users work together on a design, interacting with each other in real-time. Traditionally, this process would have to be done sequentially using several iteration steps.

At the time of development of the system, web-based technologies offered little support for group work and therefore it was decided not to use them. Furthermore, application-sharing tools, which send entire screen views to the connected instances as updates, were inadequate for interactive modelling, because of high network load and...
inflexibility. Instead, a message-based approach was chosen, using high-level semantic messages for synchronising ARCADE instances. Every scene change is converted into a short textual message and sent to the distributed ARCADE instances.

ARCADE was implemented as an extension to an existing shared 3D viewer. The viewer had shared and private cameras, asynchronous communication facilities, and one could save an entire evaluation session to continue it later. ARCADE continues on the principles of this viewer, and extends it with modelling capacities. Figure 2-2 shows the architecture of ARCADE. A primitive concurrency control mechanism has been included, allowing only one user to edit a particular part at a time. This user requests the modelling rights for the part and when he receives them the part is locked, thereby restraining other users from modifying it. However, it is possible for all users to create new parts and modify other parts in the same model, provided that these parts have not already been locked by other users.

![ARCADE architecture](image)

**Figure 2-2 : The ARCADE architecture**

Users can view the results of other user actions almost in real-time. The user is provided with shaded, shadowed and textured graphics to enhance visual realism. Three-dimensional input devices can be used in combination with stereo glasses in order to allow more intuitive interaction.

The ARCADE system uses a shared database and a session manager controls the message flow and locking mechanism. Several sessions could run simultaneously, of course using different models. Every session has its own server process on the session manager, managing state information of all instances active in the session.

In a collaborative environment, all participating clients in a session must use the same modelling information. In order to keep the data structures on all clients consistent, the update messages need to arrive at all clients in the same order. Using a communication server that serialises the messages and distributes them to all instances solves this problem for ARCADE, since messages cannot pass each other in this way.

Although ARCADE uses a message-based approach, using high-level semantic messages, an enormous amount of events can be triggered during interactive modelling. This is why some kind of optimisation has to take place. In the case of the ARCADE system, changes are not transmitted over the network after every action, but events are collected, and transmitted together after regular time intervals.

### 2.2.3 Collaborative Solid Modelling-system

The Collaborative Solid Modelling-system (CSM) (Chan et al. 1999) is a collaborative web-based modelling system that uses several methods to prevent concurrency problems.
It creates a multi-user environment where a shared model can be accessed and modified by multiple users in real-time, while every user actually has his own local model. So again, the replicated approach is used in this system. An outline of the structure of CSM is shown in Figure 2-3.

![Figure 2–3: The CSM architecture](image)

When several distributed clients start a modelling session, the Java applets that run in the browsers of those clients connect to the CSM server, synchronising all web clients. Changes invoked by one of the clients are applied to the local copy of the model first. After that, the changes are sent to the server, which updates the global model at the server computer. Finally the changes are sent to all other clients, which in turn update their local copies.

An administrator manually sets some concurrency variables. For this reason, he has two methods at his disposal, namely TOKEN and FUNCTION. When TOKEN is set, the system uses regular token passing. FUNCTION means that particular functions can be blocked for some user or users. Examples of such functions would be Add and Delete. The basic representation within the system is Constructive Solid Geometry (CSG). For viewing, the CSG-model is evaluated and stored using a boundary representation. The CSG-representation is used for sending changes incrementally over the Internet, since a boundary representation is considerably larger than the corresponding CSG-representation.

### 2.2.4 NetFeature

NetFeature (Lee et al. 1999) is a collaborative web-based feature modelling system. Several distributed browsers are the clients that connect to a neutral feature model server. The NetFeature server has several components as shown in Figure 2-4. It communicates with the Database Server, which manages all model data.

![Figure 2–4: Architecture of NetFeature](image)
The Neutral Feature Model contains a boundary representation of the feature model, using attributes to store feature information like surface finish, design intent and tolerance. Attributes can be attached to each feature solid, face, edge and vertex in the boundary representation. The Neutral Feature Model is built upon the ACIS Solid Modelling Kernel. The Feature Agent Manager controls all agents, where each agent is responsible for one client and provides service to this client by answering its requests.

The server provides some basic functions for the Neutral Feature Model, including creation and deletion of geometric entities, sweeping and Boolean modification operations.

A generic naming scheme manages all topological changes to faces and provides naming consistency between geometric entities on server and clients. It puts a tag on all faces of a boundary representation, containing information about the underlying surface type, the step during which the face was formed or modified, and the index it was assigned during that step. It provides a unique name for each entity within NetFeature. When a new feature is created, all affected faces, edges and vertices are updated. New faces are created and old ones deleted if necessary. A graph structure is maintained to record all design steps.

At the clients, two types of representations are used: the Client Side Feature Model and the Attributed Abstracted B-Rep (AAB). The Client Side Feature Model consists of form features or machining features. The AAB is a simplified boundary representation, based on the Neutral Feature Model on the server, containing enough information to provide real-time display, navigation and some geometric query operations like ray test and volume. However, not all processing can be done on the clients, so sometimes the server must be accessed to use some of its functionality, for example by querying for topological information that is not available on the client.

The AAB contains two representations, namely a face-based representation, using triangulated polyhedrons, and an edge-based representation of the wire-frame of the part. Especially the face-based representation can be costly to send over a network. To minimise network usage, changes are propagated incrementally using the generic naming scheme mentioned above. In this way, not the whole model has to be sent to the server and back to all clients, only the new features.

When a feature is created on a client, a solid of the feature is created inside the modelling server. Then generic name identifiers are put onto the geometric entities of the solid. Finally the new generic name identifiers are returned to the client, along with the necessary triangulated faces created by the ACIS solid modelling kernel.

For communication, the CORBA standard protocol is used. CORBA IDL offers the client an interface to the functionality of the NetFeature server. For example, AgentManager is one of the classes in the NetFeature IDL protocol, representing the Agent Manager on the server. It contains all functions necessary to log in, log out, add agents, delete agents, etc. The server has been implemented in C++, and the clients are Java-applets. The rendering and navigation is done in Java3D.

In order to reduce network load, NetFeature tries to let the applications, running on a client, act as autonomously as possible. The data and functionality available on the client is enough to move the viewing camera through and edit the scene, so interaction with the server is only required when changes have to be sent to or received from the server.
2.3 Conclusions

Collaborative modelling systems can assist groups of designers in co-ordinating their modelling activities. Instead of an iterative process, during which the design is sent to several individual designers and back again a number of times, designing becomes an interactive process, in which multiple designers are involved to agree on design issues.

Concurrency and floor control are important issues in collaborative environments. If a user is allowed to change a model entity, while another user is already changing the same entity, problems could arise concerning consistency. To avoid this situation, a strict floor control mechanism can restrict 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 floor control mechanisms. Also, a system developer should always bear in mind that designing is a constructive activity. Users can therefore be given some responsibility for establishing a good collaboration.

Being synchronised is another condition for an application to be called collaborative. Updating design data over a network is difficult, since there is a certain delay between the moment that data is sent and the moment that the same data is received at the other end of the network. During this interval, two distributed instances could both send conflicting update information. Mechanisms to detect these conflicts should be available and recovery mechanisms provided. Good locking can also help to avoid these situations from happening, although sometimes it could be decided not to provide such strict locking in favour of more flexibility for the users.

For a collaborative CAD application to be successful, it should provide a good level of interactivity. Users will not be able to design properly if they have to wait a considerable time after every operation. A solution to this is the replicated approach, where a server co-ordinates the collaborative session and provides functionality that cannot be implemented on the client. The clients perform operations locally as much as possible. Only information necessary for updating will be sent over the network. Sending changes incrementally over the network reduces the network load considerably. This is important, since lower network load causes information to arrive sooner at its destination. The usage of high-level semantic messages can further reduce network load.

Usually in a distributed environment, different hardware and operating systems are used. To integrate heterogeneous environments, many systems use the CORBA architecture (OMG 2000).

In order to avoid implementing an application for several platforms, Java can be used. For many operating systems, Java-capable web browsers exist, so when a system is required to be accessed through a network, the combination of Java and a web browser can provide a solution.

Although many measures can be taken to help a participant of a collaborative session to get good interactive behaviour, the user should always be informed about the current state of the modelling session. Otherwise, he might wonder what he is waiting for, since in a distributed environment delays occur regularly.
Collaborative modelling systems will become increasingly important in future design environments. Expanding network bandwidth will allow faster interaction between co-workers, in a way that it shall not be relevant whether the co-workers are in the same room, or on different continents.
3. The SPIFF modelling system

Feature modelling is a relatively new concept in Computer-Aided Design. This chapter looks at the concept of feature modelling, in particular at the SPIFF feature modelling system.

3.1 Feature modelling

Feature modelling is a discipline in Computer-Aided Design in which models are defined as a set of interrelated feature instances. Feature instances are created by specifying parameter values for generic feature classes, such as cylindrical blind holes and rectangular pockets. Besides containing shape information, a feature class also includes non-shape information such as feature intent and constraints. Examples of feature intent are adding strength to the product, or allowing connections to be made with other products. Constraints are conditions that must be satisfied for a feature model in order to be valid. Feature classes contain constraints that are assigned during the initialisation of a feature instance, but additional constraints can be defined on or among feature instances at a later stage in the modelling process.

In feature modelling systems, modelling operations can be grouped into two major categories, namely feature operations and constraint operations. Feature operations involve adding, removing and editing features. Constraint operations involve adding, removing and editing validity constraints.

3.2 Structure of the SPIFF modelling system

SPIFF is a prototype multiple-view feature modelling system, developed at the chair of Computer Graphics and CAD/CAM at the Faculty of Information Technology and Systems of Delft University of Technology. Compared to conventional feature modelling systems, SPIFF not only presents features at the user interface level, but it really stores and processes feature information. It offers, in particular, advanced facilities for multiple views and validity maintenance.

The SPIFF modelling system can be divided into several components. The Feature Model Manager executes modelling operations and maintains the validity of the Feature Model. It receives input from the Graphical User Interface and uses several other managers, for example to manage feature model geometry and constraints. The Feature Model contains, among other things, the Cellular Model, described in Section 3.4.2. The SPIFF system makes extensive use of the ACIS Modelling Kernel (Spatial 2000).

A simplified architecture of the SPIFF modelling system is shown in Figure 3-1. It is described in more detail in (Bronsvoort et al. 1997). The remainder of this chapter will describe the SPIFF modelling system.

3.3 Multiple views and validity maintenance

During product development, the same product model can be looked at from several perspectives, or views, for different product life cycle activities. Every activity can have its own view on a product, using its own feature library. In SPIFF, for example, a design view
and a manufacturing view are implemented. In the design view, a user can use additive and subtractive features. An example of a product in a design view is shown in Figure 3-2a. Manufacturing machines allow only subtractive features, since milling and cutting operations remove portions of raw material. For this reason, the manufacturing view was implemented. Figure 3-2b shows the same product in a manufacturing view. Compared to the design view, it has another set of features.

![Simplified architecture of the SPIFF modelling system](image)

**Figure 3 - 1 : Simplified architecture of the SPIFF modelling system**

In order to be able to support different views, conversion between views must be possible. This conversion takes place when a view is opened or when a view is edited. In the first case, a new feature model is built, based on the product model already specified in the other views. In the second case, the changes made in one view have to be propagated to the other views. In this way, different views are kept consistent at all times. Feature conversion is described in (de Kraker et al. 1997).

When a user performs a modelling operation, it has to be ensured that this operation does not overrule the design intent of the feature instances already in the feature model. The process of monitoring each modelling operation, in order to ensure that all features remain valid, is called **validity maintenance**. A feature modelling operation is considered **valid** if it results in a feature model that conforms to all its constraints. When one or more features become **invalid**, the system must be able to detect these changes and take appropriate action to maintain consistency.
constraints in a feature model are violated, the feature model is invalid. After an invalid modelling operation, the user is assisted in adjusting the parameters of the modelling operation in order to overcome the constraint violations. 

SPIFF contains advanced functionality to perform validity maintenance (Bidarra and Bronsvoort 2000).

In a feature class, several feature validity constraints are defined (Bidarra et al. 1998). All feature validity constraints of a feature class are applied to all its instances. An example of a feature validity constraint illustrates this for a cylindrical blind hole. A cylindrical blind hole has a top face, a side face and a bottom face. Both the bottom and the side faces have to be on the boundary of the modelled object. The top face, however, is not allowed to be on the boundary of the modelled object. These requirements are specified by means of so-called topologic constraints (Bidarra and Teixeira 1993). If another feature were to be moved over the blind hole, causing the top face to be on the boundary of the modelled object, the topologic constraint on the top face would be violated, and the blind hole would no longer be valid, as shown in Figure 3-3.

![Figure 3-3: Invalid operation for blind hole](image)

Besides feature validity constraints, it is also possible to specify additional model validity constraints. These define relations between features or between their elements, like faces and edges. For example, model validity constraints can enforce that two faces of different feature instances remain parallel at all times.

Constraint solving, the process of ensuring that all constraints are satisfied, is performed in SPIFF by two dedicated solvers, namely a geometric constraint solver and an algebraic constraint solver. Geometric constraints include relations such as co-planarity and paralellism, whereas algebraic constraints include relations in which a dimension of a feature instance depends on another dimension of the same feature instance or a dimension of another feature instance.

### 3.4 Models

The SPIFF system maintains a two-layer feature model, consisting of a set of canonical shapes, representing the individual form features, and a Cellular Model. These will be explained using the part visualised in Figure 3-4.
3.4.1 Canonical shapes
Each feature contains a canonical shape definition. For example, a through hole is represented by a cylinder and a rectangular pocket is represented by a block. In Figure 3-5, all features in the feature model of the part in Figure 3-4 have been visualised, except the base block. Each letter points to a separate canonical shape.

Additional information about a feature is kept with the feature instance. One of the attributes kept with feature instances is their nature, specifying whether its volume is additive, adding material to a product, or subtractive, removing material from a product. Protrusions, for example, are additive, whereas through holes are subtractive features. Feature shapes also contain a set of shape elements, such as shape faces and shape edges.

3.4.2 Cellular Model
The Cellular Model represents the geometry of all feature instances in a product. It is shared by all views and, therefore, provides a common representation for the product geometry. In Figure 3-6 the cells of the feature model are visualised by wire frames, and some of them marked by numbers.
The Cellular Model consists of a connected set of volumetric cells. The cells are defined in such a way that each cell either lies completely inside a canonical shape, or completely outside it. Each canonical shape is represented by a connected subset of cells. Each cell keeps an owner list, containing all canonical shapes that share the cell. Two cells can never volumetrically overlap, so when two canonical shapes overlap, their cells are split, with the overlapping area consisting of shared cells, each of them having multiple owners. The Cellular Model is described in detail in (Bidarra et al. 1998).

### 3.5 Cameras

The SPIFF modelling system visualises feature models using cameras. Cameras possess sophisticated functionality with which the user can not only view the resulting final shape of a feature model, but also customise the way a product model is visualised. Feature instances may be assigned specific colours and be individually visualised as wire frames, with visible line only, with shaded faces, or combinations of these. The user may also chose to visualise faces that are not on the boundary of the resulting object and which would normally not be visualised, e.g. closure faces of holes and pockets.

Using the functionality of the cameras, users can easier focus on specific regions of interest within the feature model. Figure 3-7 shows an example of a feature model that was visualised in SPIFF, using a camera. In Figure 3-7a, the visible on-boundary faces of the feature model are shaded, while the occluded on-boundary faces are visualised by showing their edges as dashed lines. In Figure 3-7b, only the not on-boundary faces are shaded, while the other faces are visualised by their edges, as continuous lines.

Besides visualisation functionality, cameras also provide functionality to graphically interact with the model. This includes adjusting the viewing parameters, thereby moving the viewing camera around the visualised model. Also, selecting features and feature faces is handled by the cameras. Each feature face has a name that is unique within the feature instance. In combination with the feature name, the modelling system can identify each face in a feature model by this name. Cameras are used to interactively select faces so that they can easily be identified, for example when a modelling operation is being specified.
(a) Shaded on-boundary faces

(b) Shaded not on-boundary faces

Figure 3-7: Camera images
4. Architecture of webSPIFF

Before developing a new collaborative modelling system, several alternative architectures were considered. This chapter discusses these architectures, and globally describes the chosen architecture.

4.1 Alternatives for architecture

Before implementing webSPIFF, several alternative architectures for the new system have been considered. The choice between these alternatives was based on the following criteria. First, the new system should provide interactive facilities to the remote users. Second, the system should preferably be web-based, since web-based applications are much easier accessible for remote users. Most importantly, the system should be able to properly handle the problems inherent to distributed and concurrent systems; otherwise, inevitably consistency problems would arise. Finally, the effort that would have to be put into handling concurrency and synchronisation had to be taken into consideration.

In distributed environments, it is common to use a powerful server computer to serve several clients. In this way, specialised, expensive hardware can be shared by many users. For a recent survey of client/server architectures, see (Lewandowski 1998).

A first possibility was to move all functionality of the original SPIFF modelling system to the clients, creating so-called fat clients. This approach would have meant rewriting the complete system, so that it could have been executed from within a web browser. However, this is technically impossible, because SPIFF makes extensive use of the ACIS Modelling Kernel, which is typically not available at the clients. Therefore the execution of modelling operations could not be transferred to the clients.

An additional difficulty here would have been that every client should have processed its own modelling operations, on its own local feature model. Extra measures would have had to be taken in order to prevent inconsistencies between feature models on different clients. Users could, for example, be executing a modelling operation at the same moment, resulting in different feature models at several clients. The system should be prepared for these situations, being able to synchronise all users with each other.

An alternative was to keep all functionality on the server, moving none of it to the clients. Clients that do not possess functionality of their own are often called thin clients. Here the server would send images of the user interface to the clients. Clicking on the image in this case would mean generating an event containing the screen co-ordinates of the location the user clicked on. The event would then have to be sent to the server, where the co-ordinates in the event would be associated with a click on the particular widget. Finally, the action associated with the click on the widget would have to be processed. An image of the resulting user interface would be sent to the client.

This approach is very expensive in terms of network traffic, since it would be necessary to almost continuously send images of the user interface from server to client and in the reverse direction. The response time following a user interface action would be intolerably high, thus making it very unpleasant to run the system remotely.
To overcome the shortcomings of the alternatives sketched above, a balance between them had to be found. A client/server architecture using a session manager on the server provides a good solution. Here, some of the functionality of the original SPIFF modelling system, in particular functionality for interaction with feature models, is moved to the clients. However, all real modelling functionality, requiring much computational power and using functionality provided by the ACIS Modelling Kernel, is kept at the server. The session manager controls all communication between the SPIFF server and its clients.

An important advantage of this architecture is that there is only one feature model in the system. Clients send their modelling operations to the server, and receive feedback after any modelling operation has been performed on the central feature model on the server, thus avoiding inconsistency between multiple versions of the feature model. The session manager deploys the functionality for receiving messages concurrently from all clients, serialising them on the server, and propagating them to the SPIFF modelling system. The client/server architecture used for webSPIFF is described in some detail in the remainder of this chapter.

4.2 Components of webSPIFF

webSPIFF consists of several components. On the server side, two main components can be identified. First, there is the SPIFF modelling system, which is a slightly extended version of the stand-alone SPIFF modelling system described in Chapter 3. For every modelling session, there is an instance of the SPIFF modelling system running. Second, there is a Session Manager that manages all communication between the Web Clients and the SPIFF modelling system. In addition, the Session Manager also provides functionality to start, join, leave and close a modelling session.

Outside the server computer, several Web Clients can be present, linked to the Session Manager through a network connection. Initially, a user browses to the webSPIFF portal, and an HTML file is loaded into the user’s web browser. This HTML file contains a link to the Web Client program, allowing this software to be automatically downloaded from the webSPIFF portal, and executed. The first action of the Web Client is to establish a connection with the Session Manager, enabling the client to start or join a modelling session. Web Clients can connect from various locations, local through a network or remote via Internet. Figure 4-1 shows the global architecture of webSPIFF.

![Figure 4-1: Architecture of webSPIFF](image)

The Session Manager is the interface between the SPIFF modelling system and the distributed Web Clients. It stores information about the currently ongoing sessions and its
users, and handles the information streams between Web Clients and SPIFF. Since several users can send modelling operations and queries to SPIFF at the same time, concurrency must be handled at the Session Manager. Practically, this means that parallel information streams have to be serialised.

The Session Manager can be compared to an operator, sitting in front of a computer, collecting messages from several other people, and using the information in those messages to operate the SPIFF modelling system. When several messages are handed over to the operator at the same moment, he must choose which one to handle first, and he must also store the other ones during the processing of the chosen task. The Session Manager has been implemented using the Java programming language.

The Web Clients are in fact remote user interfaces through which modelling operations can be specified on the SPIFF modelling system. In order to be able to offer the client the same functionality that the original SPIFF modelling system possesses, some high-level feature model information is copied to the clients. Although webSPIFF is a multiple-view feature modelling system, each Web Client can use only one view, specified when the user joins the modelling session. If another view is needed, a new Web Client can easily be started alongside the existing Web Client, by visiting the webSPIFF portal using a new instance of the web browser that was used to start the first Web Client. The Web Clients have also been implemented using the Java programming language.

4.3 Models

For consistency reasons, only one central feature model is maintained on the server. This feature model includes all canonical shapes, representing individual features in a specific view, and the Cellular Model (see Section 3.4.2). Although all web clients fetch information from this source, temporary local inconsistencies can still occur at a clients, for example after the execution of a modelling operation, such as an add feature or edit feature operation. Such a modelling operation is specified on a Web Client, but executed on the central feature model. Local models are present at the Web Clients, but they are not real feature models. They contain just enough information for clients to be able to autonomously interact with the feature model, without needing continuous feedback from the server. The local models need to be updated after a modelling operation, but since sending information from the server to all clients takes some time, the feature information on the clients is not up-to-date for this short period, and therefore inconsistent with the central feature model.

Besides modelling operations, also camera operations can be specified at the Web Clients. These involve changing the visualisation and viewing parameters for a camera window. Therefore, they have no influence on the central feature model.

A user can specify modelling operations by using panels on the Web Client, selecting parameter values from a list or entering parameter values into text fields. However, a more convenient way of user interaction is by using a visualisation of the feature model to graphically select faces or shapes.

Changing the viewing parameters is also possible using panels on the Web Client, but, again, it would be more convenient to do this by graphically interacting with the model. The SPIFF system provides cameras for these purposes, as described in Section 3.5. In order to replicate these cameras on the Web Clients, three models were introduced:
• **a sophisticated image of the feature model**
  The sophisticated image is a rendered image of the feature model. Many visualisation options can be specified before it is rendered. Individual features can, for example, be visualised as wire frames, with visible line only or with shaded faces, and be assigned different colours. The functionality of the SPIFF cameras is described in more detail in Section 3.5. A separate image is needed for every camera, and this image, generated by the SPIFF modelling system, must be updated after every modelling operation. Also, after a camera operation has been executed at the SPIFF modelling system, the sophisticated image of the camera on which the camera operation was specified must be updated. The images are stored using the GIF image format.

• **a visualisation model**
  Rendering a sophisticated feature model image at the server takes some time. Therefore, interactively changing the viewing parameters requires an additional local model at the clients. The visualisation model is a model of the resulting final shape of the product. It can be used for moving the viewing camera around the product model. After the desired viewing position has been reached, the new viewing parameters are sent to the server, where a new sophisticated feature model image is rendered. This image is then sent to the client, where it is displayed. All cameras on a particular client use the same visualisation model, but each camera has its own viewing parameters, including a viewing camera position. The visualisation model is generated by the SPIFF modelling system and stored in a VRML file. The VRML format is very convenient for propagating a model across a network.

• **a selection model**
  The selection model is a collection of three-dimensional objects representing the canonical shapes of all features in the feature model. Its purpose is to support the selection of feature faces, on the sophisticated feature model image, as parameters of a modelling operation. Feature faces in the selection model are only visualised for highlighting selected faces. Again, the selection model is identical for all cameras on a client, each applying its own viewing parameters. The selection model is generated by the SPIFF modelling system and stored in several VRML files, one for each canonical shape.

In Figure 4-2, a collaborative modelling session is shown for webSPIFF. Three users are evaluating the same product model using several cameras, each camera applying its own viewing parameters.

Model information on the clients is never modified directly by the clients themselves. When a feature operation has been specified, the operation is sent to the server, where it is executed. After that, updated model information is sent back to the client, and if the central feature model has been changed, updated model information is sent to all other users in this session as well. Updated model information consists of a new sophisticated feature model image, a new visualisation model, and a collection of updated canonical shapes. The canonical shapes are updated incrementally, so only new and modified canonical shapes need to be sent to the clients.

After a camera operation has been specified and sent to the server, a new sophisticated feature model image is generated by the SPIFF modelling system. Since the feature model
is unaffected by camera operations, only the new sophisticated feature model image must be sent to the client that requested the camera update.

**Figure 4 – 2 : A collaborative modelling session**

### 4.4 Data communication

The different components of webSPIFF have to exchange information at several stages during a modelling session. The Web Clients specify modelling operations, camera operations, and a variety of queries. These commands have to be sent to the Session Manager, and the Session Manager, in turn, must send a response back to the Web Clients. The Session Manager collects messages, sent by the Web Clients, and propagates modelling operations, camera operations and queries on the feature model to the SPIFF modelling system. The SPIFF modelling system, in turn, must execute these messages, and send a response to the Session Manager. Therefore, a communication medium between the components had to be introduced.

Communication between the Session Manager and the Web Clients is handled by socket connections. A socket is one end-point of a two-way communication link. This means that there is a socket on the client and a socket on the server, with a socket connection set up between them. One socket connection can handle communication in two directions. Besides textual messages, also more complex objects can be sent over sockets, at least if the programming languages used to implement the sockets support this. The messages sent from Web Clients to Session Manager are simple textual messages, whereas the messages sent in the reverse direction are more complex objects, such as arrays of data. Sending complex objects was possible because the Java programming language, used
for implementing both the Session Manager and the Web Clients, supports the exchange of complex objects over a socket connection.

Socket connections also provide a good solution for creating a communication channel between the Session Manager and the SPIFF modelling system. Textual messages are used to pass information from the Session Manager to the SPIFF modelling system. Textual messages are also sent back, but several data structures, such as the visualisation and selection models, are saved into files by SPIFF, and read from disk by the Session Manager.

Section 7.2 will explain the socket connections used in webSPIFF in more detail.
5. The server

On the server side of webSPIFF, the SPIFF modelling system and the Session Manager can be identified. This chapter describes these components.

5.1 Extensions to the SPIFF modelling system

The SPIFF modelling system, as part of webSPIFF, does not differ much from the original SPIFF modelling system. Here, the central feature model is stored and modelling are executed. However, some changes have been made in order to integrate the SPIFF system in webSPIFF.

First of all, functionality to allow communication with the Session Manager has been added. The Feature Model Manager of SPIFF was extended with communication facilities, waiting for incoming connection requests and, after a connection has been established, messages from the Session Manager. A separate instance of the SPIFF modelling system is used for each session.

Second, functionality has been implemented so that requests that are received over the connection with the Session Manager, can be interpreted and executed. A file containing templates has been introduced, in which all permissible input messages with their parameters are specified. For each modelling operation or query, the templates file instructs SPIFF which commands to execute. Practically, this templates file introduces a protocol for communication from the Session Manager to the SPIFF modelling system.

Third, notification and confirmation requests that originally appeared as pop-up windows in the user interface of the SPIFF system, are retained as textual messages, suitable for propagation to the Web Clients.

Fourth, often information has to be extracted from the feature model in the SPIFF modelling system. The feature model information is used to create files containing the visualisation and selection models for the clients. Also, sophisticated feature model images are saved to file, so that the Session Manager can send them to the clients.

Finally, some extra functionality has been added to the SPIFF modelling system. An example of this occurs during the add feature operation. When a user is specifying such an operation in the SPIFF system, he can define a name for the new feature. After specifying all other parameters for the feature, it is checked whether the name already exists and, if so, the user is prompted for a new name. In the web-based SPIFF system, this would be very inconvenient, since a message would have to be sent back across the network in order to ask the user to specify a new feature name. Therefore, the names of new features are automatically generated by the SPIFF system, ensuring that the feature is assigned a unique name at the first attempt.

5.2 The Session Manager

Two important types of processes run on the Session Manager. First, a Client Manager is maintained for each Web Client, managing the socket that is used for communication with the Web Client. It receives messages from the client, interprets them, and either processes a message itself, if possible, or propagates it to the SPIFF modelling system. Besides Client Managers, Event Managers can be identified within the Session Manager. For every
session one Event Manager exists, initiated by the Client Manager that opened the session. The Event Manager has an event queue that collects all tasks that have to be passed on to the SPIFF modelling system. The structure of the Session Manager is shown in Figure 5-1.

![Diagram of the Session Manager](image)

**Figure 5 - 1 : Structure of the Session Manager**

The Client Manager and the Event Manager processes, called *Threads* in Java, run independently from each other. The main function of a Client Manager is to handle all communication coming from the corresponding Web Client by interpreting messages and taking an appropriate action. A separate Thread is needed for every ClientManager, because within a Thread, it is only possible to listen to a single socket. Three types of actions can be distinguished. First, session operations have to be handled. Session operations involve starting a session, logging into and out of a session, and also querying for session information. Second, modelling operations can be received that have to be propagated to the SPIFF modelling system. The results of a successful modelling operation must be sent to all Web Clients. This is handled by the Event Manager, which retrieves a list of users from the SessionInfo object, and starts updating the appropriate Web Clients. If an operation was not successful, a single message, containing information about the failure, is sent to the Web Client that initiated the operation. Third, queries for information about the feature model and camera operations must be sent to the SPIFF modelling system. The results of such actions must be sent only to the Web Client that asked for the action to be executed.

When a user starts a new session, a session record (SessionInfo) is created by the Session Manager. In this object, all relevant information about the session is stored, for example the name of the session, a reference to the Event Manager associated with this session, and a reference to the client record (ClientProfile) for each user currently participating.

In the client records, information about individual clients is stored. Information is kept about the connection between the Web Client and the Session Manager, the modelling view the user is currently using, and a list containing the names of the Cameras the Web Client has opened (CameraInfo). When a user leaves the session, his client record is removed again, and the SPIFF modelling system is notified that no more feature model images, visualisation models and selection models have to be generated for this client. For each entry in the list of cameras in the client record, a *remove camera* operation is therefore sent to the SPIFF system. When the last user of a particular view logs off, this view is closed on the SPIFF modelling system. When the last user of a session logs off, the session is considered to be over, and all session information is disposed of. This means
that the view the client was using is closed, the connection to the SPIFF modelling system is closed, the session record for this session is removed, the Event Manager is shut down and the SPIFF instance is killed.

When a modelling operation, a camera operation or a query on the feature model needs to be propagated to the SPIFF modelling system, a message is placed in the event queue of the Event Manager. When several clients are logged onto the session, and thus several Client Managers have been created, concurrency problems could occur, because Client Managers operate independently. For example, when different Client Managers try to add an event to the event queue at the same time, a conflicting situation arises. Fortunately, Java offers locking mechanisms that can be used to solve this. This handling of concurrency on the Session Manager is described in Section 5.4.

After a modelling or camera operation has been processed, it is the task of the Session Manager to synchronise the users of a session, by sending them the updated data structures. However, several possible types of feedback are possible here, depending on the type of operation and whether the operation was successful. In case of a successful modelling operation, all clients will need to be sent updated data structures. Messages are created and filled with new sophisticated feature model images, VRML strings containing the resulting final shape of the product model, and VRML strings containing the canonical shape objects that were changed during the modelling operation. Also, some additional information on the feature model is added to the messages, such as a list of all feature instances that exist after the modelling operation was processed. A separate message must be created for every user, since each user will have different cameras, and therefore different feature model images. Also, the feature model information will be different for users that use different modelling views. After a message has been defined, it is sent to a Web Client, using the connection information that was stored in the client record for this user. After a Web Client has received its update message, it can extract the information in order to update its data structures.

When a modelling operation fails, and the central feature model enters an invalid state, the client that asked for the execution of the modelling operation is notified about the failure. Information about why the operation failed, generated by the SPIFF system, is shown to the user, in order to allow him to correct the problems. Of course this information does not need to be sent to all clients, although their modelling facilities are frozen until the model is valid again.

Camera operations include adding and removing cameras, and changing the viewing parameters of an existing camera. After an edit camera operation has been executed successfully, not all data structures on the client need to be updated. Only a new version of the sophisticated feature model image is sent to the client owning the camera, by including the image in a message and sending it to the client. When an add camera operation has been executed successfully, also a sophisticated feature model image is sent to the Web Client. However, if the new camera is the first camera for this client, also the selection and visualisation models must be included in the message.

5.3 Data structures
Several data structures can be identified at the SPIFF server. First of all, since the SPIFF modelling system has only undergone minor changes, the feature model information of
SPIFF is still available. This means that there are still canonical shapes and a Cellular Model, as described in Section 3.4.

Additional data structures are stored by the Session Manager. The session information contains information about the ongoing sessions and its users, such as the session and client records described in the previous section. Also, an event queue is maintained at the Event Manager of the Session Manager. Each event stored in the event queue is a Java Vector object, being served by a first in, first out basis.

The selection and visualisation models, containing information that was extracted from the feature model of SPIFF, are stored in files located at the server. These files can be loaded by the Session Manager, which includes the information in the update messages for the Web Clients. The selection and visualisation models are stored in VRML format and are textual files. They can easily be read by the Session Manager, which stores them in Java String objects. The sophisticated feature model images are handled in a similar way. These are GIF images, which can be imported by the Session Manager as Java Image objects.

The update messages that are built at the Session Manager are also Java Vector objects. They serve as containers for other data structures, which can be extracted by the Web Clients in the same sequence as they were added by the Session Manager.

5.4 Concurrency handling

In a distributed multiple-user environment, at some stage concurrency must be handled. If this is not done properly, it can result in very serious problems, such as inconsistency of data structures, or processes waiting for each other, causing deadlock.

However, not only distributed applications can suffer from concurrency. An application that uses multi-threading, meaning that several processes run concurrently within the application, must also make sure that concurrency involving its resources is managed in a proper way.

Concurrency at the EventManager

The most delicate case of concurrency occurs at the Session Manager. Communication streams from all clients come together, requesting information to be sent back and modelling operations to be carried out. The Session Manager serialises data that arrives in parallel streams.

As seen in Section 5.2, each Web Client is represented on the Session Manager by an instance of the ClientManager. When two events arrive at the Session Manager at the same moment, the two ClientManagers will try to add an event to the event queue of the EventManager simultaneously. However, only one process should be able to add an event to the event queue at one time, so the system must determine which ClientManager will be allowed to add the event first.

Fortunately, the Java programming language provides useful locking mechanisms for this purpose, the most important being the synchronised mechanism. When a class is accessed by a synchronised method, all its methods and data structures are locked, preventing them from being accessed by more than one process at the same time.

In the following Java code fragment, as an example, a class is shown in which two synchronised methods have been included:
1. class Number {
2.   private int number;
3.
4.   public Number() {
5.     number=0;
6.   }
7.
8.   public synchronized int getNumber() {
9.     return number;
10. }
11.
12.   public synchronized void setNumber(int new_number) {
13.     number=new_number;
14. }
15. }

Synchronising the methods ensures that, for example, two processes cannot update the Number class at the same time, but also that a process cannot query the class using the getNumber() method, while another process is already updating the class using the setNumber() method.

In the EventManager, the event queue is synchronised, to ensure that no two ClientManager processes can add an event at the same moment, which could corrupt the queue.

The following example shows another situation where concurrency could occur:

1. if (! isBusy()) {
2.   setBusy(true);
3.   startProcessing();
4. }
5. else {
6.   wait();
7. }

In this code fragment, a busy state variable has been introduced, to make sure that no other users are currently processing a task. If this is the case, the busy variable is set to true and the user starts processing. However, the following code fragments show what could happen if two users concurrently execute this code:

<table>
<thead>
<tr>
<th>Process A</th>
<th>Process B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>if (! isBusy())</td>
</tr>
<tr>
<td>2. if (! isBusy())</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>setBusy(true);</td>
</tr>
<tr>
<td>4.</td>
<td>startProcessing();</td>
</tr>
<tr>
<td>5. setBusy(true);</td>
<td></td>
</tr>
<tr>
<td>6. startProcessing();</td>
<td></td>
</tr>
</tbody>
</table>

The following situation occurs here. Right after the isBusy() method has been executed by process B, process A performs the same method. Both processes conclude that the necessary resources are available and prepare for processing. Both perform the method setBusy(true), and start executing the task. However, the if-statement was meant to prevent this from happening. Synchronising the Busy class was not enough in this case. If
the first two statements had been taken together, forming an atomic statement, this would not have happened. The following code fragment shows how this can be achieved:

```java
Process A                          Process B
1. if (setBusy(! isBusy()))
2. if (setBusy(! isBusy()))
3.                     startProcessing();
4. else wait();
```

Process B checks if someone is busy and immediately sets the busy flag. Since the setBusy() and isBusy() methods are in the same synchronised class, the class stays locked until the busy variable has been set. The Java programming language supports this mechanism, where a synchronised method is allowed to be called by another synchronised method from the same class. Process A concurrently tries to set the busy variable too, but finds that Process B has already set the busy flag, and now waits for Process B to finish.

The problem sketched above could, without the right measures, occur at an EventManager instance. The EventManager processes an event and then checks to see if there are any more events in the queue to process. If this is not the case, the EventManager will set the wait flag, and wait for new events to arrive. When a new event is added to the event queue, the wait flag is reset again. However, if a new event was added right after the event queue had been checked, but before the wait flag was set, the EventManager would never wake up again, since it would not have noticed the new event arriving. This problem was solved by checking the state of the event queue, and setting the wait flag using a single instruction, similar to how the busy variable was set in the previous example. Both variables do have to be accessed by methods from the same synchronised class, thereby ensuring that no other instructions from other processes are executed between the check of the state of event queue and the setting of the wait flag.

**Downloading data using HTTP**

Another possible concurrency problem arises when the sophisticated feature model images and the VRML files have to be propagated to the clients. Using the hypertext transfer protocol (HTTP), it would be very easy to transfer the files parallel to the existing socket connections between the Web Clients and the Session Manager. However, this construction could result in concurrency problems. The model files are saved on disk and a client would download the files directly from the server, parallel to the socket connection between Web Client and Session Manager. If another modelling operation had been executed, and the model files were updated before the client had completely downloaded the files, invalid model files could reach the Web Client.

Therefore, the sophisticated images and VRML files have to be sent using the existing socket connections. The socket connections guarantee that messages are received in the same sequence as they were sent. Furthermore, the Session Manager creates the update messages in between operations, so there is no chance that updated information could become available while the messages are being created. The image objects and VRML strings are therefore included in the regular update messages that are sent from the Session Manager to the Web Clients. The disadvantage of this approach is that it is slightly slower than HTTP, since the models have to be added to and extracted from messages. Also, the sophisticated images have to be converted to a platform-independent format in order to be sent over the socket connection (see Section 7.2).
Conflicting modelling operations

During a modelling session, two users may decide to execute a modelling operation at the same moment. Both will have been warned by the *traffic light* in the user interface of the client, described in Section 6.2, that at least one other user was also preparing a modelling operation, thereby encouraging the users to co-ordinate their activities. Still, it could happen that both clients decide to execute their modelling operation. Consider the following situation: the modelling operation that is handled first by the Session Manager will remove a certain feature, while the second modelling operation will try to edit this feature. In the SPIFF modelling system, this situation could not occur, since operations were always performed serially, not concurrently. After a feature had been removed, it would not be possible to edit it, since it could not be selected in the user interface anymore. Therefore, no measures were originally taken to check that the features in a modelling operation still existed. In webSPIFF, however, where operations are performed concurrently, operations can potentially be defined on features that do not exist anymore. Validation checks have therefore been added, to ensure that these operations are rejected. The user trying to execute such an operation is notified that his operation is not meaningful anymore.
6. The Web Clients

The Web Clients provide the remote user interface to the users of webSPIFF. In order to be able to offer them the same interactive functionality as the existing SPIFF modelling system does, it is not enough to just replicate the user interface of SPIFF on the Web Clients. Additional functionality has to be available on the clients. The functionality on the Web Clients will be described in this chapter.

6.1 Structure

Several components can be identified within the Web Clients. In Figure 6-1 the architecture of the Web Clients is illustrated.

![Figure 6 - 1 : Architecture of the Web Client](image)

The user interface is the component used for interaction between users and modelling system. It includes widgets such as windows, menus, buttons, text labels and list boxes. Three major user interface classes can be distinguished, namely a Main Menu, a View Menu and a Camera. The first two classes provide plain interfaces with standard widgets. The Camera class provides functionality for graphical interaction with the model.

All communication with the server is handled by the Communications Manager, which is an independent process that manages the socket through which the client is connected to the Session Manager on the server side.

6.2 User interface

A Web Client can be seen as a remote control for the server-based SPIFF modelling system. For the clients to be fully functional, all widgets available on the SPIFF system had to be copied to the Web Clients. Since the original user interface, written using the Tcl/Tk scripting language, could not easily be copied into a web-based application, a completely new user interface has been implemented in Java. Java supplies the Abstract Window Toolkit (AWT) for building user interfaces. In fact, Tcl/Tk scripts could be run from within a web browser, using a plug-in, but the scripts would have needed to be adjusted so thoroughly that it was easier to implement a new user interface.
The user interface of webSPIFF has been built in such a way that it resembles the existing user interface of the SPIFF modelling system as much as possible. However, extra menus were necessary in order to be able to join and leave a session. In a network environment, it is also very important for distributed applications to have extended state information available, because delays could lead a user to wonder about the state the application is currently in. Therefore, extra objects that supply state information were added.

Figures 6-2 and 6-3 show two typical user interfaces during a modelling operation. Figure 6-2 shows the user interface of the original SPIFF modelling system, while Figure 6-3 shows the user interface of a Web Client of webSPIFF. Both interfaces contain a Main Menu panel, used for specifying general options, such as loading a model and, for webSPIFF, starting or joining a modelling session. In the View panel, modelling and camera operations are specified, while the Camera window in both interfaces shows a shaded image of the feature model.

In the top right corner of the Design View panel of webSPIFF, one of the new state indicators can be found. When it is safe to perform a modelling operation, this indicator shows a green box. However, as soon as one of the session participants starts preparing a modelling operation, this traffic light shows an orange box, to indicate that other users are currently active. When the SPIFF modelling system is actually executing a modelling operation on request of a user, it is advised not to perform a modelling operation, because the operation may be in conflict with the operation currently being executed, and the traffic light turns to red. Users can still choose to perform operations when the traffic light is orange or even red. It has been decided not to implement strict floor control policies, since modelling is considered to be a constructive activity. Normally, a designer will not
join a modelling session in order to destroy the model, or harass other designers. Communication between users, for example by telephone or Internet chat, will always remain necessary, because it does not make sense to have several users performing modelling operations simultaneously, without co-ordination. Synchronisation is discussed in Section 6.6.

6.3 Data representation

On the Web Clients, several data structures are used to store information. As pointed out in Section 4.3, no real feature model is maintained at the client side, just the information needed for the user to be able to interact with the model in a convenient way.

In order to be able to specify a modelling operation, information is needed to fill widgets with names of feature instances and feature classes. For example, when an *add feature* operation is being specified, the Web Client needs to have information on all feature classes in its view.

The list of feature classes is a textual string, containing the names of all feature classes. It is obtained via a query to the server at initialisation of the client, and does not need to be refreshed during a modelling session.

For other modelling operations, information about feature instances is needed. The list of feature instances is also a textual string, containing the names of all feature instances currently in the model. It is refreshed after every modelling operation. Upon initialisation of the Web Client, a query is sent to obtain the current list of feature instances, but after

![Figure 6-3: User interface of webSPIFF](image)
that no explicit query needs to be sent by the Web Client, since a new list of feature instances is included with every update message.

The lists of feature classes and feature instances are used for specifying operations on the product model via panels of the user interface of the Web Clients. However, the Cameras also provide functionality for graphical interaction with the product model. To keep response times at an acceptable level, several data structures extracted from the central feature model are available at the client; the characteristics of these models are shown in detail in Section 4.3.

The sophisticated feature model image, created by the SPIFF modelling system, is represented as an Image object in Java. The visualisation model arrives at the Web Client as a textual string, defining a polygon mesh using VRML code. The canonical shapes in the selection model are defined in VRML code too, which is converted to Java3D objects at the Web Client. The interactive functionality of the Cameras is discussed in Section 6.5.

6.4 The Communications Manager

The Communications Manager on the Web Client manages all communication with the Session Manager on the server. It contains a separate process that continuously listens for incoming messages on the socket connected to the Session Manager. Also, outgoing messages bound for the Session Manager are sent via the Communications Manager. Two kinds of messages can be identified here:

a) messages whose response is not awaited, before the Web Client can be operated again; these messages include all modelling operations, such as adding, removing and editing a feature

b) messages whose response is awaited, suspending all activities on the Web Client concerned; these messages include queries and camera update messages.

When a message of type b is sent, all activity on the Web Client is suspended until the response to the message is received. However, it is still possible that other messages arrive at the Web Client earlier than the expected response. These messages are stored in a queue and ignored until the expected response has been received and processed. All remaining messages in the queue will be processed after completion of the ongoing operation. Synchronisation problems can occur when messages are temporarily stored and ignored. These problems are discussed in Section 6.6.

The reason that camera update messages fall into the second category is that after a camera operation has been specified and sent to the Session Manager, the interactive functionality cannot be used any more to continue operating webSPIFF. Specifying camera operations is typically done interactively, using the visualisation model. Specifying modelling operations is typically done using the sophisticated image, in combination with the selection model. However, after the visualisation model has been transformed, both the visualisation model and the selection model are inconsistent with the sophisticated feature model image. Therefore, parameters for the modelling operations cannot be selected on the sophisticated feature model image until an updated image arrives. For this reason, the Web Client is suspended until the new image has been received.

Every message is given a unique identification number at the Communications Manager, in order to be able to determine whether a received message is the message the Web Client is waiting for. Since the server includes the same number in its response
message, it can easily be checked whether a certain message is the one that was expected. Update messages that are sent to multiple clients are assigned a special identification number by the server. This identification number cannot be assigned to regular messages by the Communications Manager, since it is used to distinguish update messages from expected messages.

After a message has been received, it has to be processed. The role of the Communications Manager is to dispatch the right information to the right place. Camera update information has to end up at the Cameras, and information on the product model will be stored in other data structures.

Whereas the messages sent to the Session Manager are merely textual strings, the messages that a Web Client receives from the Session Manager are more complex, containing multiple other objects. The first object in all messages, regardless of the type of message, is a command string. This command string is parsed by the Communications Manager to find out which kind of message it is, and how it should be handled. It contains, for example, the identification number. Besides a command string, the message can contain the visualisation and selection models, the sophisticated feature model image, and additional information on the feature model, such as a string containing a list of feature instances. When the Communications Manager has determined which kind of message has arrived, it knows which objects it should contain. Therefore, the objects in the message can now be extracted by the Communication Manager and passed on to the right data structures. Figure 6-4 shows the format of the different kinds of messages that can arrive at the Communications Manager.

![Figure 6-4: Format of the messages](image)

### 6.5 Interactive facilities

Cameras provide interactive visualisation and modelling facilities on the clients. A Camera is a separate window in which a graphical representation of the product model is shown. Several visualisation techniques can be used for the sophisticated feature model image, by setting parameters in the Camera panels at the Web Client. Using these techniques, extra insight into the selected feature is offered, such as its shape and location in the model. This could, for example, mean that some features are visualised as wire frames, instead of shaded objects. Often, features are occluded by other features in a way that it is not clear how they are exactly positioned. By representing some features as wire
frames, effectively allowing users to see through them, other features can be examined much better. Facilities for rendering images with such properties make extensive use of the ACIS Modelling Kernel, and are therefore not available on the clients. The images therefore have to be rendered at the server, before being sent to the clients. Sending an image from the server to a client is very cheap, both in terms of network load (approximately 10 Kbytes) and display time at the client.

The Camera panels on the Web Clients provide facilities to redefine the parameters of the viewing transformation. However, graphical interaction on the product model is desired, meaning that its position and orientation, relative to the viewing camera, can be changed by using the mouse. During a mouse operation, the viewing parameters are continuously changed according to the mouse events that are generated, creating a smooth animation. Since rendering a sophisticated feature model image at the server and sending it to a client takes quite some time, it is impossible to interactively position and orient a product model relative to the viewing camera using the sophisticated feature model images. The time between the arrival of two successive images on the client would simply be too long, hindering smooth animation.

The call for graphical interaction on the model led to the introduction of the visualisation model. Using the visualisation model, the product can be positioned and oriented relative to the viewing camera in real-time. The process of positioning and orientating a model starts with creating a Camera. First, a sophisticated feature model image is shown in the Camera window. When a mouse button is pressed, the sophisticated feature model image disappears, revealing the Java3D visualisation model. The user can then move the product model, until it has reached the desired position and orientation. Section 7.1 describes how this was implemented with Java3D. The server is then asked to update the sophisticated feature model image with the new Camera parameters. The feature model is rendered, and an updated feature model image is delivered to the Web Client, where it is displayed in the Camera window. The visualisation model then disappears again. The visualisation model only needs to be regenerated by the server and updated at the clients whenever any user modifies the product model. Sending the VRML code to the clients is reasonably cheap in terms of network load (in the order of 100 Kbytes for a moderately complex feature model). Figure 6-5 shows a Camera window in which the visualisation model of a product model is shown.

![Figure 6-5: Visualisation model](image)
Graphical interaction is more than visualising the product model. It is also important to be able to select features and their faces on a sophisticated feature model image. For this, the selection model was introduced on the Web Clients. The selection model is not displayed. It is “invisible” behind the sophisticated feature model image. When a user clicks on a certain position in the Camera canvas, a conceptual ray is sent into the selection model and it is then checked whether the ray intersects any canonical shape objects. A list of intersected faces is returned, sorted by distance to the viewing camera, and the user can browse through this list, until the desired face is being highlighted. Its name can then be used as a parameter value for a modelling operation. 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 modelling operation. Sending VRML code of the canonical shapes is again cheap in terms of network load (in the order of 5 Kbytes per canonical shape. Figure 6-6 shows three sophisticated feature model images, on top of which faces of the selection model are displayed after they were selected by repeatedly clicking on the position indicated by the mouse cursor.

![Figure 6 – 6 : Selecting a feature face](image)

6.6 Synchronisation

Synchronisation is the process of propagating data from one component of a distributed system to another, in order to keep the information on the components consistent. Before data structures can be updated, it must be made sure that the involved Web Clients are in the right state for processing the update. Three types of synchronisation can be distinguished. First, updated feature model information, resulting from a successful modelling operation, can be received. Second, feedback from camera operations can be received. And third, updated state information can be received. The order in which these updates are received at the clients is not known in advance, and different scenarios must therefore be handled. Three scenarios are described here.

Updating feature model information

In webSPIFF, the feature model can be modified at any time by one of the users. After such a modification, new feature model information will have to be sent to all clients. At the client side, preparation of a new modelling operation could be underway when the update arrives. It is not convenient to force the user into cancelling his operations, since it might
well be that the user co-ordinated his operations with the other users in a way that the update does not have any influence on the modelling operation that is being prepared. For example, one user could be editing a feature, while the update concerns another feature, not having any influence on the edit feature operation of the first user. This could have been co-ordinated using communications channels outside webSPIFF, such as Internet chat or telephone. Cancelling the operation would mean that all the parameters that have been specified so far would have to be specified again. Therefore, the user is allowed to continue specifying the modelling operation, but a warning appears to notify him that an modelling operation has been carried out at the server. The user can now choose to finish specifying his operation, or to cancel it himself.

The state diagram for the update process at the Web Client is shown in Figure 6-7. The system is in idle state after the Web Client has been initialised. This state is left either after a message from the Session Manager is added to the event queue, or when the user starts performing a modelling operation. Starting such an operation is only possible when the message queue is empty, and it is only when the operation is finished that the message queue is checked again for new messages.

![State diagram for processing model update messages](image)

**Figure 6 - 7 : State diagram for processing model update messages**

**Updating the Cameras**

When a Web Client is specifying new viewing parameters for a Camera, other incoming updates are queued until the camera update has been completely processed. Now, the following situation could occur when a user is editing a Camera using the camera panels on the Web Client. An update arrives that was issued after a feature modelling operation from another user. This update message is queued until the camera operation has been applied and its feedback has been received from the server. After the feedback from the camera operation has been processed, the image displayed in the camera window is a more recent picture of the model than the one that is included in the update message that is still being queued at the client. The visualisation and selection models at the camera, however, are still representing the “pre-operation” feature model. Right after the Camera update has been carried out, the update message in the queue will be processed. However, this update message also replaces the more recent feature model image again. The Web Client should therefore ignore the image in the update for this particular Camera, and keep the most recent feature model image. The underlying VRML models for visualisation and selection,
on the other hand, do need to be refreshed for this Camera. The other Cameras that the client has opened must, of course, be updated completely. The state diagram for this process is shown in Figure 6-8.

**Figure 6-8: State diagram for processing an update resulting from a camera operation**

### Updating state information

Besides updating the feature model information at the client, also state information, such as the traffic light, must be kept up to date. The difference with the other data structures is, however, that state information must always be processed as soon as possible. Whereas other messages that arrive unexpectedly can be put into a queue at the Communications Manager, awaiting their processing, the most recent state information must always be available to the client immediately. Since the purpose of state information is to inform the user of the current state of the modelling session, the information would be useless when it is not immediately shown to the user. Therefore, for every message that arrives, it must be immediately checked whether it is a state update message. If not, it can be safely stored in the message queue, otherwise the state information must be updated instantly. The state diagram for this process is shown in Figure 6-9.

**Figure 6-9: State diagram for processing a state update message**
7. Implementation aspects

During the development of webSPIFF, several interesting issues had to be overcome. This chapter will describe some aspects of the implementation in a greater detail.

7.1 Using Java3D

Before implementing the Cameras on the Web Clients, it had to be decided how to visualise the three-dimensional models. For web-based modelling using a Java-applet, two obvious alternatives are available, namely the Virtual Reality Mark-up Language in combination with the External Authoring Interface (VRML/EAI) and the Java3D API.

With VRML/EAI, a Java program creates VRML code that is sent to a conventional VRML viewer. The interactive functionality of the VRML viewer is then used to position, orient and make selections in the model. The model and the Java code are not closely connected to each other, and the interactive functionality is limited to the functionality the VRML viewer offers. Although the VRML standard defines selection on objects, it does not prescribe selection on particular objects occluded by others. Therefore, it cannot be guaranteed that the VRML viewer available to the user is capable of selecting occluded objects. However, this functionality is very useful, especially for reasonable complicated feature models, where many faces are fully or partly occluded by other faces.

The Java3D API is obviously very closely connected to Java, since it is an extension to the Java programming language. Interactive functionality can be implemented by the programmer, and many standard functions are even already available. Selection is possible, also on faces occluded by other faces.

For these reasons, the cameras were built using the Java3D API. However, VRML is still being used in webSPIFF for transmitting models from the server to the Web Clients. Java3D has built-in VRML loaders at its disposal, so models defined in VRML code can be imported, converted to Java3D objects and eventually rendered, not having the restrictions VRML/EAI suffers from.

Java3D scenes are set up by defining all its components and connecting them to each other in a scene graph. Figure 7-1 shows the scene graph used on the Web Clients, containing the visualisation and selection models and the sophisticated feature model image.

The root of every Java3D scene graph is its virtual universe. Every scene has exactly one virtual universe. The Locale class defines the origin of the scene, usually chosen at (0, 0, 0). The left branch of the scene graph defines the geometry of the objects in a scene, while the right branch of the scene graph defines the viewing transformations.

Two types of groups can be identified in the scene graph. First, BranchGroups serve as the roots of a scene graph branch. They are the only components that can be attached to a Locale object. TransformGroups are components that contain a transformation that is applied to all its children.

In the geometry-defining-branch, the objects that are to be displayed are contained. All client side Camera models, such as the visualisation model, the selection model and the sophisticated feature model image, reside here in order to be displayed in a Camera window. Besides scene objects, the geometry-defining-branch also contains mouse behaviours. These mouse behaviours define which mouse actions will be interpreted as rotations, translations and zoom operations. When such a mouse action is performed, the
transformation applied to the model is adjusted accordingly, thus rotating, translating or zooming into and out of the model.

![Diagram of Java3D scene graph on the Web Clients](image)

**Figure 7 - 1 : The Java3D scene graph on the Web Clients**

The viewing transformation is set up in the following way. A ViewPlatform object is defined, to which one or more View objects are connected, each View object representing a separate Camera window. Since each Camera window is independent from the others, all Cameras need to have their own scene graph. If Cameras would share a scene graph, rotating the visualisation model, for example, would cause this rotation to be applied to all Camera windows. Therefore, only one View object is defined in each scene graph.

The mouse behaviours attached to the scene root, transform the model and do not affect the viewing transformation. However, the Cameras of the SPIFF modelling system move the viewing camera through the scene, thus directly changing the viewing parameters and leaving the model unaffected. These two modes of operation therefore had to be synchronised, since otherwise the SPIFF modelling system would not be able to generate new images with the viewing parameters the user had specified on the client. The solution for this was to first take the transformation matrix from the TransformGroup parenting the selection model and visualisation model, and apply its inverse transformation to the position of the viewing camera, the view reference point and the view-up vector. The same inverse transformation is applied to the transformation matrix in the TransformGroup, effectively resetting it to the identity matrix again. This causes the whole scene, including the viewing parameters and geometry, to be transformed as if the viewing camera has been moved around a stationary model, resulting in the same scene image as before, but with the viewing parameters that are used by the cameras of the SPIFF modelling system.

Besides real-time visualisation and interactively changing the viewing parameters of the Visualisation Model, also selection must be handled. In Java3D specific capabilities must be set in order to allow certain functionality to be applied to objects in a scene. For
selection, this means that every canonical shape object in the Selection Model must be explicitly set to allow intersection computations being applied to it.

The selection functions are called when a user clicks a mouse button. A ray is sent into the scene, using the screen co-ordinates from the mouse event. This means that the ray originates from the eye of the user, and goes through the position on the viewing plane that was clicked on by the user. When the ray intersects a canonical shape, the name of the face that was intersected is included in a list of intersected canonical shape faces, sorted by distance to the viewing camera. The canonical shape face closest to the viewing camera is highlighted first, and when the user clicks again, the second closest intersected face is highlighted. Obviously, only the branch in the scene graph containing the Selection Model needs to be searched, and not, for example, the branch containing the Visualisation Model.

Every feature canonical shape is read from a separate VRML file. This VRML file in turn consists of several nodes, each containing a polygon mesh of a feature face. The names of the nodes coincide with the names of the feature faces, and the VRML loader of Java3D uses these names to build a hash table. The keys in this hash table are the feature face names, and the objects they refer to are the polygon meshes that represent the feature faces. Upon selection, Java3D returns a set of intersected feature face objects and, using the hash table, their names can be looked up. The name of the selected feature face can then be used during the definition of a modelling operation.

When a feature face has been selected, its polygon mesh is highlighted over the sophisticated feature model image. This is done by changing its appearance parameters, setting the transparency attributes from fully transparent to fully opaque.

### 7.2 Using sockets

Within the webSPIFF system, socket connections are used to allow the different components to communicate with each other. Figure 7-2 illustrates the process of connecting two components via sockets.

![Figure 7-2: Setting up a socket connection](image)

Component A runs a socket server that listens for incoming connections on port x. Component B requests a connection by contacting port x. If the connection is granted, a new port y is assigned on Component A for communication with Component B. Server port x returns listening for possible new connections, while a new connection is being established between server port y and client port z. The two components need not be on the same computer; connections between sockets on different computers can be established over a local network and via Internet in the same way.

During initialisation of a modelling session, a connection between a socket assigned to an instance of the SPIFF modelling system and a socket assigned to the Session Manager is
established. For this reason, every instance of the SPIFF modelling system contains a socket server listening to a port known to the Session Manager. During a modelling session, textual messages are sent from the Session Manager to the SPIFF modelling system, and vice versa, via the socket connection that was created.

Transmitting images over a socket connection
In webSPIFF, images created on the server have to be transmitted to the Web Clients through a socket connection. Since both the Session Manager and the Web Client have been implemented in Java, some Java objects can be sent without conversion. Such a complex Java object, sent to the Web Client by the Session Manager, is immediately available at the Web Client after it has been received. Object classes that possess this property are called *serializable* in Java. However, the Image objects are represented in different ways on different platforms, and can therefore not immediately be transmitted as Image objects. The images must be converted to arrays of integers, which are serializable. After the array of integers has arrived at a Web Client, it can be converted back to an Image object again.

7.3 Communication between Session Manager and SPIFF
Transmitting objects between two distributed components implemented in Java is relatively straightforward. However, the Session Manager communicates with the SPIFF modelling system via the Feature Model Manager, implemented as a set of Tcl/Tk scripts. Fortunately, Tcl/Tk provides facilities for accessing sockets, corresponding to the way Java handles sockets. Therefore, textual strings can easily be sent from a Java application to Tcl/Tk scripts, and vice versa.

In the Feature Model Manager, the command strings arriving from the Session Manager are used to select the corresponding template from a collection of templates that was loaded from file. Each template contains a command name, several parameters, and a list of instructions that must be executed. When a particular command arrives at the Feature Model Manager, the command name is looked up in the collection of templates, and the parameter values are inserted into the instructions that are to be executed. These instructions can then be executed sequentially, after which the results of the command are sent back to the Session Manager as a textual string.

7.4 Security issues in Java
In distributed environments, it must be made sure that malicious people do not abuse a system. In this context, one could think of stealing product model information or gaining access to forbidden regions of the system. The Java programming language automatically poses several restrictions to what resources Java applets are allowed to use. These restrictions had to be cancelled or worked around in webSPIFF, because otherwise some of the functionality that is required by webSpiff could not be used.

Java applets are normally not allowed to write and read local files. Only if a user would adjust his personal security policy file, which the Java virtual machine uses to know what it is permitted to do, explicitly setting permissions for read and write operations, the Java applet would be able to load and save models to the local disk at the client side. However, by asking the users to do this, possible hostile applications would be allowed to perform read and write operations too.
To overcome this, the Session Manager stores the necessary information at the server computer. Only a single message needs to be sent by a Web Client to the Session Manager in order to store and load a product model, or store and load Camera settings. An additional advantage of this is that information is globally accessible, so when a user would log onto webSPIFF from another location, which is not unusual in an Internet environment, he would still have access to the files.

After the Web Client applet has been downloaded from the webSPIFF portal, it will try to set up a socket connection with the Session Manager serving the client. Java applets are restricted to exclusively set up socket connections with the computer from which the Java applet was downloaded, namely the web server. For this reason, the webSPIFF portal was moved to the system that runs the Session Manager.

Another way to avoid the security issues mentioned above would be creating certificates. Certificates are produced by the developer, specifically for an application, requesting access to restricted resources. They can be downloaded by users and imported into the client’s web browser, thereby allowing a Java applet to use the resources specified in the certificate. For other Java applets, new certificates must be created if they want to make use of restricted resources. However, importing certificates requires additional skills from the user, and therefore it was decided not to use them.
8. Examples of collaborative modelling with webSPIFF

In this chapter, the functionality offered to the users is illustrated, and three scenarios for parts of a collaborative modelling session are shown.

8.1 Session operations

A user can be involved in a modelling session in two ways. A new session can be started, or an existing session can be joined. The user who starts a session, the *initiator*, enters the URL of the webSPIFF portal into his web browser. The Web Client is downloaded and executed, showing the Main Menu panel in Figure 8-1 to the initiator.

![Main Menu panel](image)

The initiator chooses to start a new session, by pressing the *Start* button. He is now asked to enter some session and user information, as shown in Figure 8-2.

![Session and user options](image)

(a) 

The View panel, shown in Figure 8-3 is created, and the modelling session is underway.

Besides starting a session, existing sessions can also be joined by users. When a user enters the webSPIFF portal, and the Web Client is executed, the Main Menu panel shows a list of active sessions, as shown in Figure 8-4. A session can be selected, and when the *Join* button is pressed, after some user information has been entered, a new View Menu panel is created for this user, and the session has been joined.
8.2 Camera interaction

Interactively changing viewing parameters

Cameras can be used to interactively change the viewing parameters for a product model that is shown. The mouse can be used to move the viewing camera around the product model. Three operations are available, assigned to different mouse buttons. If a mouse does not have three buttons, combinations of mouse buttons and keys can be used. First, the viewing camera can be rotated around the product model. Second, the viewing camera can zoom into or out of the product model. Third, the viewing camera can be translated. Figure 8-5 shows two Camera windows, with the viewing parameters shown in the corresponding Design View panel. Figure 8-5b shows the same product model as is displayed in Figure 8-5a, after a rotation has been performed.
Selecting feature faces

Selection functionality is available, to allow users to select features and feature faces in order to use their names as parameters in a modelling operation. When the user wants to select a certain feature face, he clicks on the desired face, in the sophisticated feature model image, and the nearest intersecting face is visualised by highlighting the corresponding face in the selection model. By clicking again, without moving the mouse, the second nearest face in the selection model is visualised. When the desired face is highlighted, its name can easily be used as a parameter in the panel in which a modelling operation is being specified. Figure 6-6 in Section 6.5 shows three Camera windows, in which different feature faces have been highlighted.

8.3 Collaborative modelling

In this section, three example scenarios during a collaborative modelling session with two participants are given. For simplicity, all clients will be modelling in the design view.

Example 1: valid modelling operations

Client A has logged on and initiated a new session, using the panels described in Section 8.1. A simple model has been loaded at initialisation of the session, containing three features: a BaseBlock feature, a BlindHole feature and a RectPocket feature. This feature model is shown in Figure 8-6.

Client B joins the session and uses a telephone to contact client A. Together, they decide that client B will adjust the BlindHole feature, decreasing its radius. Client A has the task to adjust the RectPocket feature, which is positioned at the other side of the model, and which is very unlikely to interfere with the BlindHole feature that is being edited by client B. Client A presses the Edit Feature button, and a message is sent to the Session Manager, notifying that client A has started specifying a modelling operation. The Session Manager immediately sends a state update message to client B, causing the traffic light of client B to turn orange. Client B also starts modelling, and the traffic light of client A also turns orange.

Both client A and client B are now specifying a modelling operation. After a some time, client A has finished specifying his modelling operation, while client B is still busy. Client A presses the Apply button, resulting in a message sent to the Session Manager. The Session Manager adds the message to the event queue, and since this is the only operation currently in the event queue, it is immediately executed. Both the traffic light of client A and client B now turn red, meaning that it is advised not to perform any operations.
However, client B knows his operation will not interfere with the current modeling operation, and thus continues. After the modeling operation of client A has been processed by the SPIFF modeling system, resulting in the feature model shown in Figure 8-7a, updated feature model information is sent to both client A and client B. Client A processes this information immediately, but client B is still specifying his modeling operation, and therefore the update message is stored in its message queue. In the meantime, the traffic light of client A turns back to orange, because client B is still busy, and the traffic light of client B turns green, because client A is inspecting the results of his operation, which is not relevant to client B. Right after client B has pressed the Apply button, the modeling update issued by client A is processed on client B. After the modeling update issued by client B has been executed at the SPIFF modeling system, both clients are updated again. The result is shown in Figure 8-7b. All traffic lights have now turned green again, reflecting the idle state of the modeling session.

![Figure 8–7: Valid modelling operations](image)

**Example 2: specifying an invalid modeling operation**

Client A and client B are still modeling the simple feature model described in the previous scenario. After evaluating the previous modeling operations, it is decided that client B must adjust the BlindHole feature again, and client A will add a protrusion on top of the BaseBlock feature. Both clients now independently start modeling. However, due to a misunderstanding, client B moves the BlindHole feature precisely to the position where client A is adding the protrusion. The validity constraints for the BlindHole feature prescribe that it must remain open, and therefore the operation will turn out to be invalid. Client B has finished his operation, which is not invalid in itself, before client A and the BlindHole is moved to its new position, as can be seen in Figure 8-8a. Client A receives a message that Client B has performed a modeling operation, but since he expected this message, he decides to continue. Client A finishes his operation, not being aware of the details of the previous operation issued by client B. The modeling operation is now executed by the SPIFF modeling system, which detects that the feature model has entered an invalid state. The result of this modeling operation is shown in Figure 8-8b.
A message is sent to client A, containing information about the invalidity and how to overcome it. A message is sent to client B, freezing its modelling facilities until client A has cancelled his operation or corrected its parameters.

**Example 3: specifying a meaningless modelling operation**

Client A and client B are still collaboratively working on the simple feature model described in the previous scenarios. However, due to a lack of co-ordination, both clients are specifying an operation on the RectPocket feature. Client A starts editing the feature, but meanwhile, client B is removing the same feature. Client B applies the remove feature operation first, and it is applied to the feature model. However, the modelling operation being specified by client A still uses the “pre-operation” feature model information, which includes the RectPocket feature. The edit feature operation of client A is sent to the server, but at the SPIFF modelling system it turns out to be meaningless, since the RectPocket feature does not exist any more. A message is sent to client A, informing him that his operation could not be executed. The feature model information has by now been updated at client A, so no further action is needed, and the edit feature operation is simply discarded. The resulting feature model is shown in Figure 8-9.
9. Conclusions and recommendations

webSPIFF is a new web-based collaborative feature modelling system, based on the SPIFF feature modelling system. The distribution of functionality between the server and the clients has resulted in a well-balanced system. On the one hand, the server offers the full functionality of the original feature modelling system. On the other hand, all desirable interactive modelling functionality is offered by the clients, ranging from display of sophisticated feature model images to interactive selection facilities. A good compromise between interactivity on the clients and network load has already been achieved. However, several additions could be implemented to further improve webSPIFF.

Communication within webSPIFF is performed using socket connections. The advantage is that every aspect of the communication between the components can be completely controlled. The disadvantage, however, is that several low-level aspects of the communication must be handled, such as assigning ports to sockets. More sophisticated approaches, such as the Common Object Request Broker Architecture (CORBA) and the Java Remote Method Invocation (RMI), offer higher-level services. They are, however, more complicated to implement, and it is questionable whether it is profitable to use them in webSPIFF. The communication protocol in webSPIFF is fairly simple, consisting of textual messages and Java objects being sent between the components. On the other hand, CORBA offers services for security, session management and concurrency handling, which could be integrated with webSPIFF.

During the development of webSPIFF, one of the main interests was to keep the network load as low as possible. However, one of the most obvious mechanisms of reducing network load has not been utilised so far, namely compression. Taking into account the relatively small sizes of the different models in webSPIFF, it is questionable whether compressing and uncompressing a model will reduce the overall time needed to replicate a data structure across a network. For slow connections between fast computers, compressing data structures will be profitable, but for fast connections between slow computers, this will not be the case.

Introducing an extra event manager on the Session Manager has been considered. Via this event manager, it could be avoided that queries must go through the regular event queue on the Session Manager in order to be processed. Query information could be replicated at the Session Manager, and queries could be answered without consulting the SPIFF modelling system first. Users would, for example, not have to wait for a modelling operation to finish, before a list of feature classes can be sent. However, creating an extra event manager, perhaps using an additional socket connection, would introduce new concurrency and synchronisation problems. For example, the two event managers could be updating one client at the same moment, meaning that two information streams would reach a Web Client concurrently.

In a network environment, it could happen that a connection is disrupted. In the current situation, if the connection of the only user in a modelling session were broken, a feature model that was not saved would be lost, and the session closed. A simple recovery
mechanism, which would save important information, would be a useful addition to the system.

Since the focus of the webSPIFF system has been on other aspects of collaborative modelling, security issues have been more or less ignored. For commercial collaborative modelling systems, however, security issues must be well considered, since otherwise both the model and the system could be abused by outside influences. WebSPIFF could also be improved in this respect.

It is indispensable for participants in a collaborative session to be able to co-ordinate modelling activities. Users could use existing facilities, such as telephone or Internet chat, but it would be very convenient if webSPIFF itself would supply such facilities. Users would then be encouraged to co-ordinate tasks, which will eventually improve the quality of the design. One such facility would be a shared camera. In webSPIFF, all users have their own cameras, with their own viewing parameters, making it more difficult for users to understand each other. The viewing parameters of shared cameras, on the other hand, are synchronised between multiple users, making sure that all shared cameras provide the same image. Additionally, telepointers, one for each session participant, could be implemented on the shared cameras, allowing a user’s mouse cursor to be displayed at the shared camera windows of all other participants.

With webSPIFF, a framework has been created, with which many issues of collaborative feature modelling can be tested and eventually improved. It is not yet a fully functional modelling system, but it can certainly be used as a starting point for further research. As Internet technology improves rapidly, faster and better collaboration will be possible. It is therefore clear that, although the development of collaborative modelling systems is only at its early stages, they could soon play an important role in the whole product development process.
References


The fearless spaceman Spiff pilots his craft at speeds never before imagined!