FLEXIBLE SOFTWARE FRAMEWORK FOR COLLABORATION SYSTEMS

Mahendra Babu, Nikhil Joglekar, Aliasgar Ganiji and Karthik Ramani
Center for Information Systems in Engineering, Purdue University, West Lafayette, IN

Abstract: CollabFramework (CFW) is a flexible framework that we have created to modularize the development of a thin client collaborative system. The construction of modules is simplified by providing programming abstractions that are common across modules. Multiple views and dataflow between modules is also supported. Clients provide shape input and this information is passed to the server using the internet. The server coordinates the operation of collaboration, assembly, feature modeling, solid modeling and generates the 3D boundary representation. The boundary representation is converted to a faceted representation which is then sent to the clients. The faceted representation allows for a thin client that has lower computing and memory requirements compared to standalone systems. All collaborations can be associated to the product being designed, stored and retrieved from the server for later re-use. This paper describes the CFW design and three experimental modules we have developed using CFW: assembly, feature modeling and 3D markup modules. We claim to be the first to develop a framework for distributed collaborative product design systems.

Keywords: Collaborative Product Design, Object Oriented Collaboration System.

1. Introduction

Collaborative product design systems based on a common product model in a distributed environment is a major research topic in academic and industrial spheres. Prasad [8], Martino [9] and Roucoules [10] are among the number of solutions that have been offered in the literature. Though these systems handle the area of collaborative CAD modeling well, they tend to be very specific solutions. Since these systems are not built as frameworks they do not lend themselves to easy extension by addition of new modules. On the other hand OneSpace Designer™ [11] is an extensible framework that allows the collaboration data to be viewed, marked-up, edited, and saved. OneSpace Designer™ can modify an existing geometry but it cannot create

new geometry collaboratively. A survey of the related work as well as the limitations and a new approach to collaborative creation of geometry were presented in our earlier research [5] and [13].

Collaborative systems are difficult to build with general user interface toolkits because such systems have special requirements. The user interface toolkits provide frames, panels, and buttons, but they do not offer primitives for developing geometry editing modules. CFW is a framework of programming abstractions that simplifies the construction of collaboration modules. To the best of our knowledge, we are the first to develop a framework for distributed collaborative product design systems.

The use of software libraries concentrates on the development of its own application software, which accesses existing routines [6]. As shown in Figure 1, the control lies in the application. The extent of the re-use of library routines is not predictable. Further, the proper design of the application architecture and library routines depends on the developers. Frameworks, on the other hand, already offer a complete architecture that is completed by conforming user-written application modules. The framework calls the user-written modules. As shown in Figure 1, the application software implements variations of abstract functionalities already in the framework. Hence, a noticeably higher software re-use is enabled by using frameworks. The software quality rises since the critical value-added features are defined in the framework and lie out of the module-specific software portion. In CFW, we provide collaboration features within the framework allowing the modules to access advanced functions such as synchronization and session management.

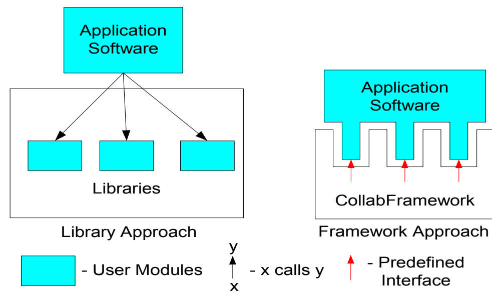

Figure 1. Comparison of framework and library-based architecture

In this paper, we present the CFW architecture and discuss our prototype implementation. The three modules we have built with CFW: assembly module, feature modeling module, and 3D markup module are described.

2 Framework Details

CFW provides a flexible collaboration environment where the users have their own independent views of the session. If necessary, all users can share the same view, but by being permitted different views, they are free to observe and discuss the model as they wish.

2.1 Hybrid Distributed MVC

CFW uses the Model-View-Controller (MVC) design pattern [2] to make the distinction between the state and operations that characterize objects and the way the objects are presented in a particular context. The user input, the system function/state, and the visual feedback to the user are separated and handled by controller, model, and view respectively.

Hybrid distributed MVC [3] is used for synchronization at the model level and provides multiple views of a given model. Updates in one view are immediately available in the others, and a user can switch between views freely. Model-level coupling is necessary as the views may be entirely different. The hybrid architecture provides each client application with its own MVC components, yet couples these via a shared model on the server. Coupling at the level of the model allows applications to use different visual representations for their views of this model. Thus, designers can work within the view that best meets their current needs while continuing to collaborate with others.

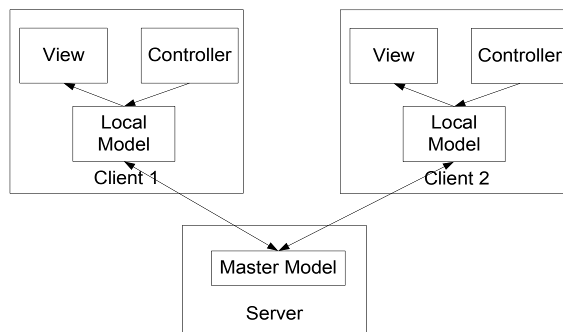

Figure 2. Hybrid distributed MVC

2.2 Distributed Command Objects

Commands are designated as emissaries on the client side and as executants on the server side [1]. An executant is a server-side component that is trusted to perform computations. An emissary is the client-side proxy for the executant. Its job is to help the client form valid requests before sending it to the server, where the corresponding executant takes over. Executants and

emissaries represent the same concepts described in MVC's commands [7]. Since the commands must be split into client and server parts anyway, this emissary/executant role is a natural extension. Command emissaries are declared and instanced on the client side by the model proxy. The model proxy creates a command emissary to form a service request. The command emissary performs preliminary validation and, if successful, the system model marshals the emissary to its remote counterpart. The corresponding command executant then performs a final validation before executing.

2.3 Client-Dispatcher-Server-Receiver

The Client-Dispatcher-Server-Receiver pattern is used to achieve the synchronization of the clients. The communication between the clients occurs in the form of distributed command objects. The dispatcher module on the client end [Figure 3] is responsible for forwarding the changes made via the controller to the server's collaboration module. The interface for the dispatcher module is similar to the model. However, the only purpose of the dispatcher is to create appropriate command objects and dispatch them to the server.

As the server receives the command object sent from one of the clients, it makes the same changes in the master model. The server then sends this or a new command object either to the rest of the clients or to all the clients. Thus, all the clients and the server can be synchronized. If the operation corresponding to the command object is to generate a new solid, then the server sends a new command object containing the approximate faceted model to all of the clients. Otherwise, the same command object is sent to the rest of the clients.

On the client end, the receiver module [Figure 3] receives the command object and calls the execute() method on it. This results in the appropriate method on the client's local model being executed. Thus, the models on the other clients are also synchronized.

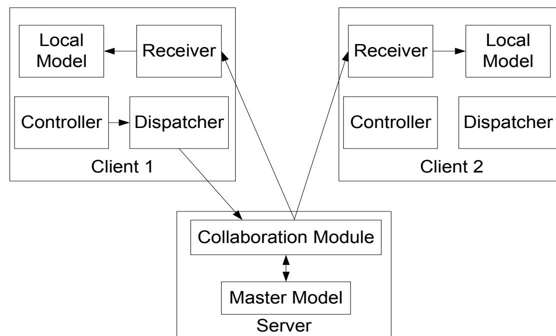

Figure 3. Client-Dispatcher-Server-Receiver

2.4 Session Management

Sessions preserve the state of a discussion on topics selected by session members. Joining a session allows a user to access the data and to collaborate if other users have joined the session. Each session has a channel which is used for transporting the command objects among the participants in the session. The server keeps track of sessions in progress. The clients have the option of joining an already existing session or creating a new session. The clients specify the session to join as an input parameter while connecting. If the session does not exist, a channel for that session is created and returned to the clients by the server. On the other hand, if the session does exist the server returns the user the channel corresponding to that session. While connecting, the clients are sent a copy of the master model on the server. This enables the clients to connect to a session at anytime and still be ensured that they would be in synchronization with the other participants in the session.

3. Server Side

The server provides support for most of the core computational operations, such as solid modeling. It is the hub where data from all the clients are routed through after processing. The server has the master model that is used to update the approximate version of the shape model on the clients. Since the computationally intensive components reside on the server, the clients are thin, freeing the clients from installing and maintaining software at their end. The server has the following modules [Figure 4].

- **Collaboration Module** – Manages collaboration, multiple session creation, transfer of editing control and maintaining a master copy of the Model information.
- **Solid Modeling Module** – Performs the creation and modification of the geometry. ACIS™ is used as the solid modeling kernel. ACIS™ provides the software routines to represent three-dimensional solids and to perform operations, such as Boolean addition, intersection, or subtraction, on these solids.

- **Visualization Module** – Creates facet representation from the boundary representation of the geometry. A polygonal approximation of the BREP model is used for visualization on the client side. The facet representation is generated by approximating the faces of the BREP model by polygons, while maintaining edge consistency between adjacent faces. A face is faceted by subdividing the face in parameter space with a grid whose increments are determined through a user-defined parameter. This parameter determines the accuracy of the faceted representation.
- **Persistence Module** – Gives persistence to product created using the system by storing the data in the database. It uses JDBC data access APIs to store the Java representation of the product being designed. JDBC™ technology [12] allows access to virtually any tabular data source from the Java programming language. JDBC™ provides cross-DBMS connectivity to a wide range of SQL databases.
- **Database Module** – Is the layer between the server and the database that provides an interface to save and retrieve data from the database.
- **JNI Module** – Is the bridge between Java and C++ components. ACIS™ is written is C++ and CFW is implemented in Java. Therefore, ACIS™ is connected to the CFW server using the Java Native Interface (JNI).

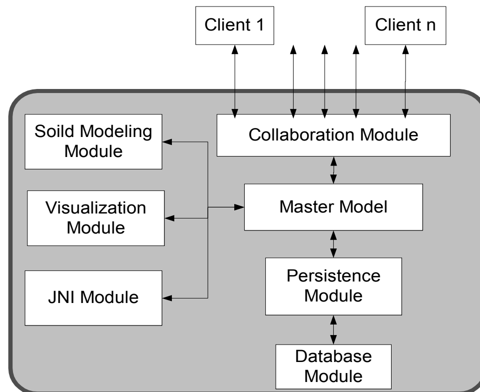

Figure 4. Server Side Modules

4. Client Side

Only the master-client is capable of doing real-time creation, editing, and deletion of the product. The other clients are capable of viewing the details

of the product being designed in the session. They can become the master-client by requesting control from the current master client.

There are two types of commands on the client side. The first type of command is handled by the local model without any computation on the server [Figure 5]. An example for this type of command is adding a new 3D markup which doesn't require interaction with the server modules. The master model on the server maintains the latest state of the model in the session. Though the local model is capable of handling the command, the result of the command execution should be reported to the server. This is done by transmitting a command object containing the result of the command execution to the server using the dispatcher. The collaboration module on the server, upon receiving this type of command object, would update the master model. Further, the server distributes the same command object to the other clients in the session. The receivers on the clients receive this command object and update their local models. This ensures that the local models of all the clients in the session are synchronized.

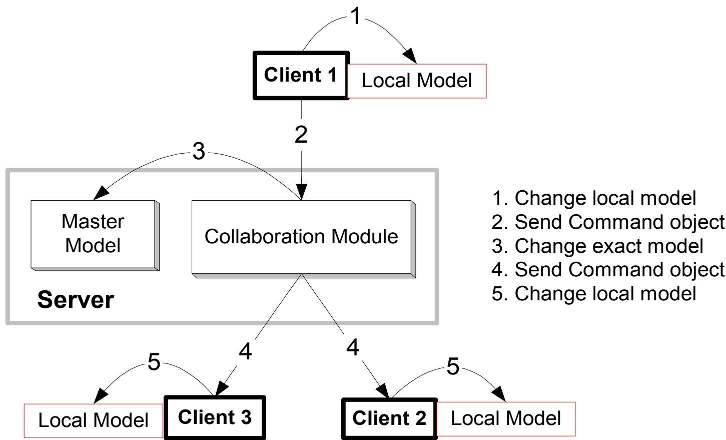

Figure 5. First type of synchronization process showing a series of operations starting at one of the clients

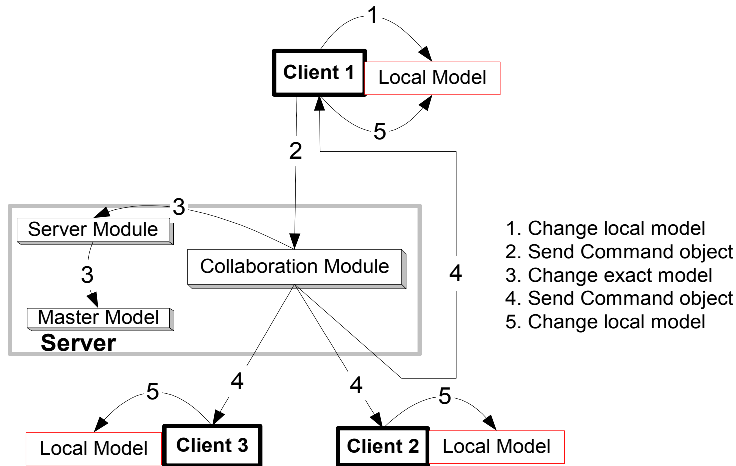

Figure 6. Second type of synchronization process showing a series of operations starting at one of the clients

The second type of command requires interaction with the computationally intensive server side modules. An example of this type of command is creating a cube, as it requires computation on the solid modeling module on the server [Figure 6]. In these cases, the input parameters are warped in the appropriate command object and transmitted to the server using the dispatcher module. The collaboration module on the server transfers the command execution to the appropriate server side module. The server side module handles the computation. The server side module which performs the computation updates the master model on the server with the result of the command execution. The result of the execution is then distributed using the appropriate command object to all the clients in the session, including the client that initiated the request.

5. Adding a new Module

A module is represented by a composite model that is integrated into the root model. We have developed a process consisting of the steps below to add a new module into CFW. We list the steps taken to add the assembly module as an example.

1) Represent the state variables describing the new module in a new sub model. While adding assembly module, the assembly constraints and the computed assembly transformations are stored in the assembly model.
2) Make this sub model a part of the composite root model. In our example, the assembly model is made a part of the root model.

3) Add the interface to modify and query the new sub-model into the main model interface.
4) Special methods that can be implemented only on the server are identified and implemented in the master model on the server. In the case of the assembly module, the special server side functions are interference detection, clearance computations etc.
5) For these special server side functions, create the appropriate distributed command objects in the dispatcher. In the assembly module, the assembly transformations are distributed to all clients in the session using a command object created on the server.
6) Create new controllers for the users to interact. These controllers create the appropriate command objects to modify the corresponding sub model. The assembly module controller enables the users to specify the mate and align constraint.
7) Create new views for displaying the data. These views query the state of the sub-model by creating the appropriate command objects. In the assembly module, the user is presented with a 3D view and a tree view of the assembly being designed.

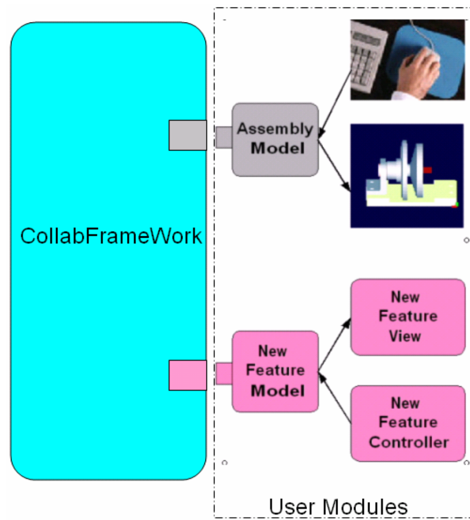

Figure 7. Adding a new module

6. Experimental Modules

We have created three experimental modules using the CFW: Assembly, Feature Modeling and 3D Markup. These three modules form the backbone of any collaborative CAD system. Moreover, there is little overlap in the design goals of these modules. Hence, a framework that eases the development of these modules can be extended to other modules as well.

6.1 Assembly

Figure 8a. Assembly Constraints specified on the two bodies

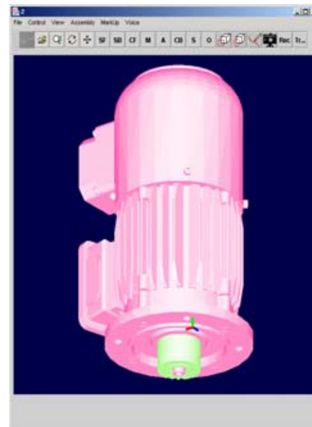

Figure 8b. Assembly Transformations applied

Using the assembly module, the clients can first import the CAD files (in various file formats) of the subcomponents they have uploaded into the server earlier. They can then collaborate and specify the assembly constraints. The transformations computed are communicated to other clients in the session in the form of Command Objects.

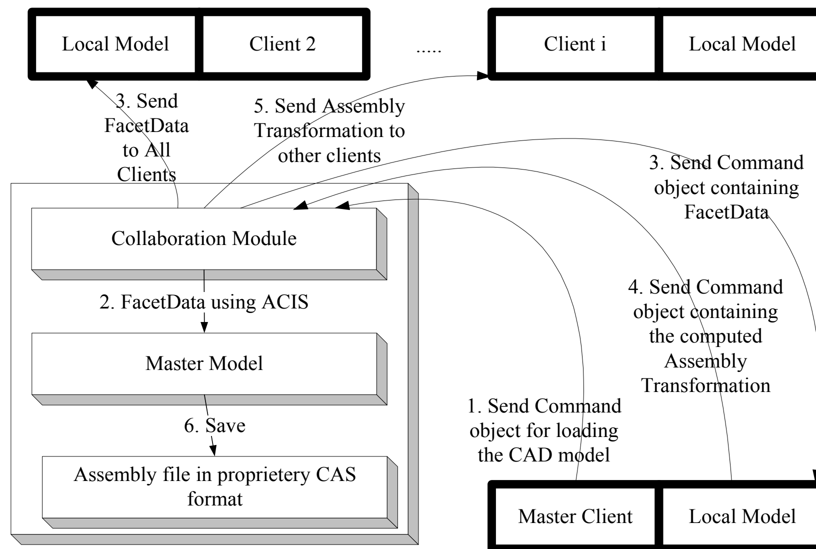

Figure 9. Synchronization process showing a series of operations involving Assembly

6.2 Feature Modeling

Shape creation can be done from scratch, or existing models can also be used as base features. The model is imported directly by the solid modeling module at the server. The master model holds the vector of used features, whereas the local models hold partial representation of the features. These two models have a one-to-one mapping between entities.

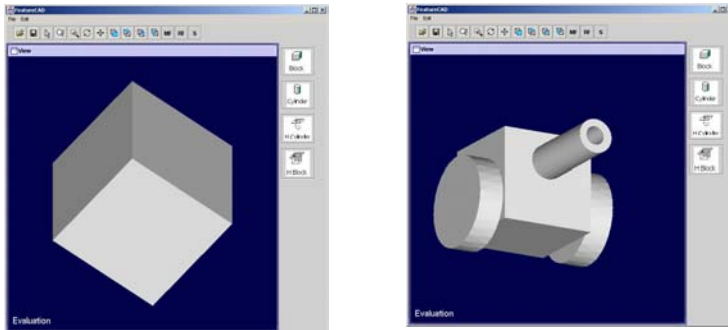

Figure 10. Part modeled using features

If a client edits any of the features on the local machine, the change is sent to the server and the master model is changed. Then the changed model is sent back to all the clients. While editing the features, the user can view the change he/she is trying to make. However, the actual change is implemented at the master model. This file is saved to provide persistence to all the information about the feature for reuse.

6.3 3D Markup

Sketching is carried out directly on the 3D faceted representation of the model. The models to be marked up are imported directly by the solid modeling module. The point(s) obtained by the intersection of the ray cast from the pixel location in the view port to the nearest facet of the 3D model is (are) stored in the model. Multiple such mark-ups can then be viewed simultaneously.

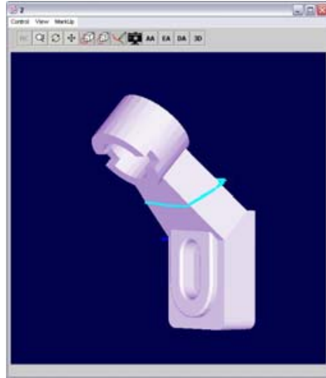

Figure 11. Marked-up object

The mark-up can be viewed in 3D to represent actual 3D artifacts such as a "parting line on a mold component". This enables the mark-up to remain valid from any view direction, unlike a 2D mark-up. A 2D mark-up is relevant only from one particular view direction and has no meaning in another view.

7. Conclusions

In this paper, a framework for a distributed collaborative product design system is presented. We have shown that CFW lends itself to a thin-client model. The thin-client model allows easy installation and use of the system by people who have limited hardware and software resources. CFW has greatly simplified the implementation of our three experimental modules. Being a framework, CFW has narrowed the design space for each module significantly, obviating basic design decisions that are independent of the module. Our experience is that developing modules with CFW is mainly a matter of choosing and implementing the required interfaces. Although collaborative product design and manufacturing systems are still at their infancy, it is expected that with the developments in networking technologies and increased bandwidth, these systems will play a crucial role in expanding distributed design and manufacturing. We view our framework as a contribution towards services that can use such models.

ACKNOWLEDGEMENTS

We acknowledge the 21st Century Technology Funds from the state of Indiana and the University Faculty Scholar Award from Purdue University to Professor Karthik Ramani for seeding this research. We acknowledge the National Science Foundation Partnership for Innovation Award (NSF EHR Award# 0227828) for continued support. We also acknowledge Nikhil

Joglekar for implementation of the feature modeling module and Aliasgar Ganiji for the implementation of the 3D markup module.

REFERENCES

[1] "Distributed MVC: An Architecture for Windows® DNA Applications," Technical White Paper, www.roguewave.com

[2] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., 1995, "Design Patterns: Elements of Reusable Object-Oriented Software," Addison-Wesley, Upper Saddle River NJ.

[3] Suthers, D., 2001, "Architectures for Computer Supported Collaborative Learning," IEEE International Conference on Advanced Learning Technologies, August 2001, Madison.

[4] Balakrishnan, A., Kumara, S.R.T., and Sundaresan, S., 1999, "Manufacturing in the Digital Age: Exploiting Information Technologies for Product Realization," Information Systems Frontiers, Vol. 1, pp 25-50.

[5] Agrawal, A., Ramani, K., Mahendra Babu, A.H., and Hoffmann, C., "CADDAC: Multi-Client Collaborative Shape Design System with Server-Based Geometry Kernel," accepted for publication in Journal for Computing and Information Science in Engineering

[6] Bischoff, J., Cochlovius, E., and Tran, M.T., "Flexible Software Architectures for Embedded Multimedia Systems."

[7] Vlissides, J.M., and Linton, M. A., "Unidraw: A Framework for Building Domain-Specific Graphical Editors," Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology, pp. 158-167, Williamsburg, VA, November, 1989.

[8] Prasad B., Wang F., and Deng J., "Towards Computer-supported Cooperative Environment for Concurrent Engineering," Concurrent Engineering - Research and Applications, Vol. 5, n° 3, pp 233-252, 1997.

[9] De Martino T., Falcidiano B., Hassingert S., "Design and engineering process integration through a multiple view intermediate modeller in a distributed object-oriented system environment," Computer Aided Design, Vol.30, n°6, pp 437-452, 1998.

[10] Roucoules L., Tichkiewitch S., "CoDE: a Co-operative Design Environment. A new generation of CAD systems," Concurrent Engineering - Research and Applications, Vol.8, n°4, pp 263-280, December 2000.

[11] CoCreate Software Inc., 2001, OneSpace Designer™ Software, Fort Collins CO. http://www.cocreate.com.

[12] JDBC™ Technology, http://java.sun.com/products/jdbc.

[13] Agrawal, A., Ramani, K., and Hoffmann, C., "CADDAC: Multi-Client Collaborative Shape Design System With Server-Based Geometry Kernel," 2002 ASME Design Engineering Technical Conference, Montreal, Canada.