To appear in proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT 2001), 6-8 August 2001, Madison, Wisconsin

Architectures for Computer Supported Collaborative Learning

Daniel D. Suthers
Department of Information and Computer Sciences
University of Hawai`i at Manoa
suthers@hawaii.edu

Abstract

Four architectures for computer supported collaborative learning systems are analyzed using the model-view-controller design pattern and compared from the standpoints of coupling between activities of the users and suitability for educational use, as well as network load and ease of implementation. The architectures are illustrated with examples from the developmental history of Belvedere, an environment for collaborative construction of knowledge representations during problem solving. A hybrid architecture that supports model-level coupling is shown to provide the best design tradeoffs.

1. Introduction

In the last decade we have witnessed a dramatic rise in popularity of computer-mediated learning, as evidenced by the growth in conferences on learning technologies, the emergence of online universities and the efforts of traditional universities to develop online degree programs. Consistent with research that demonstrates the value of collaboration in learning, computer support for collaborative learning has become of greater interest, and various architectures for synchronous and asynchronous collaboration have been explored. In this paper I discuss the suitability of four such architectures from the standpoint of the types of coupling or decoupling between the activities of learners that they can support, the ease of converting existing applications, and considerations of network load. I illustrate the architectures with examples from the developmental history of Belvedere [6]. Belvedere is an evolving environment for student-construction of explicit models (knowledge representations) while learning in science and other domains requiring critical inquiry (evaluation of alternatives with respect to evidence and criteria). In developing Belvedere, we explored all four of the architectures described herein. I conclude that a hybrid architecture offers the most flexibility for collaborative learning applications.

1.1. MVC

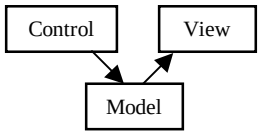

Figure 1. Model-View-Controller

This paper uses the design pattern known as Model-View-Controller (MVC) to analyze the architectures [4]. The *Model* is an internal representation of a semantic model of the problem of interest. The *View* displays the model in some visual representation. Software components implementing a View are registered as observers of the Model, so that changes in the Model will automatically result in an update to the View display. A *Controller* enables the user or the environment to modify the state of the Model. (A controller can be a human-computer interface widget, or it can be software reading from a physical sensor.) Software implementing the Model is registered as an observer of the Controllers, so that actions on the Controllers automatically result in an update to the Model state (and hence of the View).

1.2. Coupling

The architectures to be discussed differ on the degree of coupling that they support (or require) between the activities of different users and the state of applications used by those users. I define three levels of coupling.

Strict "what you see is what I see" or WYSIWIS ("whiz-e-whiz"), provides all users with exactly the same view and controller states. Strict WYSIWIS can support the collaboration of two or at most three users whose activities are *tightly* coupled. Strict WYSIWIS is problematic for larger groups or loosely coupled interactions because everyone sees the same cursor or set of cursors, and view states such as scroll position are the same for everyone. NetMeeting is an example of strict WYSIWIS.

Relaxed WYSIWS does not insist that the state of the view be exactly the same, provided the same view is being presented. Different users can scroll to different viewports on this view, and perform their own operations such as editing or moving objects without distracting the others, at least until a model change forces an update in

the view. An example of relaxed WYSIWIS is Teamwave (www.teamwave.com/).

Model level coupling, which I call "what you model is what I model" or WYMIWIM ("whim-e-whim"), guarantees only that users will see the same semantic state of a shared model. The views may be entirely different, even to the extent of using different representations. For example, in Belvedere 3.x one person can view the model as a graph and another as a matrix.

I now analyze the architectures in terms of the distribution of MVC components and the type of coupling between them.

2. Centralized

A *centralized* architecture provides only one application, and distributes *copies* of the GUI (view and controller) by sending window system events to all participating client machines. The actual model, view and controller all remain on one host machine (Figure 2).

A well known example is NetMeeting, in which the applications run on one Wintel machine and other participating Wintel machines see these applications with strict WYSIWIS. Only one person can use the mouse or keyboard at a time. This architectures' primary advantage is that it is possible to take arbitrary applications and make them collaborative *at run time* by capturing and broadcasting window system events: developers need not know in advance which applications will be shared.

The Centralized architecture's transmission of complete display information and interface events over the network does not make efficient use of bandwidth. In some implementations (exemplified by NetMeeting) it also enforces too strict a form of WYSIWIS for learning applications. Although tight coupling may be appropriate for one-on-one training such as demonstrating the use of a software system, in collaborative learning applications it is more appropriate to allow learners to shift freely between working in parallel and working together.

Belvedere 1.x was implemented in Common Lisp in the Common Lisp Interface Manager (CLIM) using a modified centralized architecture, which addressed the latter problem. The application ran as a single process on one machine, and hence is

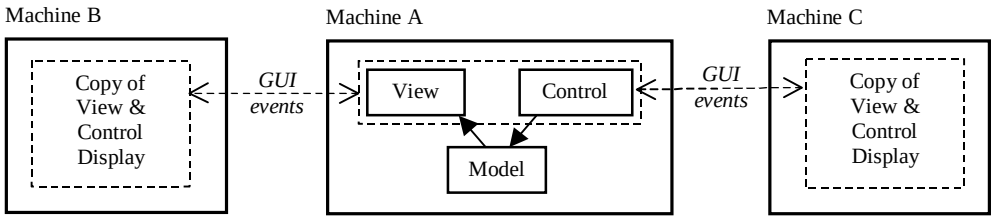

Figure 2. Centralized architecture

classified as centralized because the model and all of the views and controllers remained on one machine. However, we achieved relaxed WYSIWIS by generating multiple view-controller instances, each view-controller instance being a CLIM application frame associated with a client IP number. These frames were displayed on the remote clients via X-windows. Yet this architecture still suffered from bandwidth issues and the need for persons at all machines to coordinate setting up the displays (e.g., provide xhost permission at B, then send display from A to B), which was too complex for classroom use.

3. Replicated

In a *replicated* architecture [1] the entire application is installed and run (i.e., replicated) on each client machine; and some means of synchronization between them is provided. This architecture is characterized by having all three MVC components – model, view and controller – replicated on each client machine (Figure 3). Examples of replicated collaborative systems include E-Slate (E-Slate.cti.gr), Habanero [2], and MatchMaker [7]. All of these systems require that applications be written with collaboration in mind, using an API for event sharing. In contrast, JAMM (Java Applications Made Multiuser, [1]), provides a way to convert existing single user applications to collaborative use without modifying the code: Java Swing interface classes are modified to broadcast the events on each copy of the application to the other copies.

Replicated architectures improve on use of network resources because display data is not transmitted over the network: only Controller events need to be distributed. Also, the client can be used without the network.

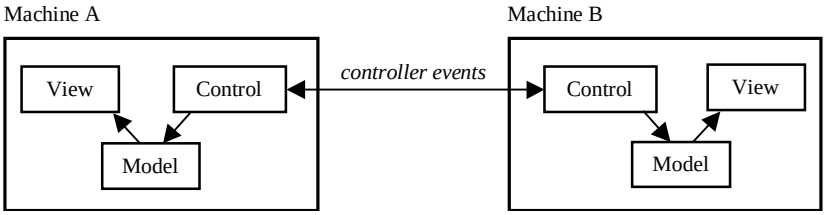

Figure 3. Replicated architecture

Replicated architectures based on the automatic broadcast of controller events (as in JAMM) have the disadvantage that they are most naturally suited for strict rather than relaxed WYSIWIS. This disadvantage can be avoided by selecting relevant events when manually building a collaborative application. Yet synchronization via controller events may be at the wrong level of abstraction for many learning applications. Learners will be more interested in each others' semantic changes (model updates) rather than in the manipulations of the GUI by which other learners achieve these semantic changes. MatchMaker's synchronization is at the model level: one can select which objects are to be synchronized, and even turn synchronization on and off at runtime.

The Belvedere 2.x series used a replicated architecture more complex than that pictured. Version 2.1 is described in [5]. We reimplemented Belvedere as a stand-alone Java application with a self-contained MVC architecture. We then provided the application with a *Listener* on a dedicated port to listen for events from the other clients. Each client also had a component that informed the outside world of changes to its model. However, rather than informing the other clients directly, this component, known as the *Belvedere Object Request Broker Interface* or BORBI, communicated with a server providing persistent storage of the model. We shall see that in this respect Belvedere 2.x represented a hybrid of Replicated and Distributed architectures. BORBI updated the remote database for each change, and also informed a Java process on the server. This Connection Manager kept a table of all active clients and the workspaces they had opened, and would broadcast change events to the Listeners of clients that had opened the workspace being changed. Belvedere 2.x also provided a simple Chat facility: users of any given workspace received messages typed into Chat by others working on that workspace. Belvedere 2.x's replicated architecture transmitted model change events rather than controller events. This reduces network traffic and opens up the possibility of model-based coupling or WYMISIM. The shared persistent store is a step towards supporting asynchronous collaboration. The architecture of Belvedere 2.2 also forms the basis for a coached collaborative distance learning system known as COLER [3].

4. Distributed

A distributed architecture is characterized by the distribution of the MVC components across multiple hosts. Typically, the Model lives on a shared server and each client has its own View and Controller (Figure 4).

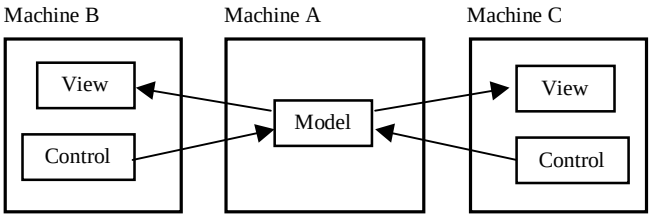

Figure 4. Distributed architecture

The most familiar example of the distributed architecture may be database-backed web sites such as airline reservation systems and other e-commerce systems. The user's web browser provides the view and controller and the server stores the data. This type of distributed architecture shares some features with Centralized in that specifications of the View and Controller are actually constructed on a server and sent to the client. Hence some of the problems of ineffective use of bandwidth apply.

A somewhat more network-efficient implementation is exemplified by Enterprise Java Beans (EJB). Simply stated, EJB enables one to run the Model as a Java Bean on a server, and have this bean shared by multiple clients consisting of View-Controller software. The View and Controller originate on the clients and are not sent over the network. One can program the View-Controller as if the model were running on the client machine. EJB handles the distribution of the model on the network and WYMIWIM updating. Other services such as transactional behavior are provided automatically. During the past two years, my students Hongli Xiang and Bo Yang experimented with an EJB architecture for Belvedere 3.0. We found that EJB provides a high initial learning curve, yet once this is overcome one can program a distributed application quickly.

Properly implemented, a distributed architecture requires only model update events to be sent over the network, making this architecture more efficient in terms of network resources. Since coupling is at the level of the model, the distributed architecture can support WYMIWIM: users can collaborate on the same model while using entirely different visual representations of the model. This motivated our experimentation with an EJB-based distributed architecture for Belvedere 3.0.

From a user's perspective, the primary disadvantage of a truly distributed architecture is the reliance on the network. The ability to run as a stand-alone application has important advantages, particularly in a classroom environment where the network may be unreliable and the teacher must be able to continue class activities after discovery of an outage, with no more than a minute's delay before chaos ensues! This motivates our current hybrid architecture.

5. Hybrid

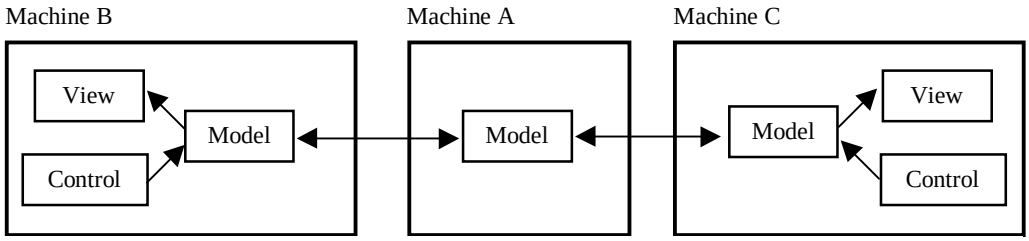

Figure 5. Hybrid architecture

Belvedere 2.x introduced a hybrid between Replicated and Distributed architectures (Figure 5). In this architecture, synchronization is at the model level via a persistent model. Applications can run standalone with their own models, saving state to the local file system, or can connect to a persistent store that provides WYMIWIM updating between active clients.

Belvedere 3.0 differs from the previous versions of Belvedere in one important respect: it provides *multiple views* on a given model. One can construct an evidence model using any of Graph, Matrix or Tree visual representations. Updates in one view are immediately available in the others, and one can switch between views freely as one works. A collaborative version of Belvedere 3.0 *requires* model-level coupling, as the views may be entirely different. We are implementing Belvedere 3.0 using the Hybrid architecture to achieve model-level coupling along with the flexibility of running either networked or stand-alone.

6. Conclusions

I defined three architectures for collaborative learning systems in terms of the location of model, view and controller components and the means of coupling between applications, and identified advantages and disadvantages for each. I described a hybrid architecture that endows each client application with its own model/view/controller components, yet couples these via a shared model on a server. While slightly more complex to implement, this architecture addresses the tradeoff between independence and coupling of applications. More importantly, coupling at the level of model state enables applications to use different visual representations for their views on this model, enabling learners to work within the view that best meets their current needs while still being able to collaborate with others. The architectures were illustrated with a series of implementations of Belvedere. Ongoing work is exploring the design of coupling between shared knowledge representations and computer mediated communication media such as threaded discussion. Future work may be needed to understand how collaboration may be affected by the use of multiple views.

7. Acknowledgements

Thanks to Bin Ma, Bo Yang and Hongli Xiang for their work on Belvedere 3.0, and to Dan Jones for his work on the Belvedere 2.x versions. This work was partially supported by a grant from the National Science Foundation's Learning and Intelligent Systems program.

8. References

[1] Begole, J., C. A. Struble, et al. (1997). Transparent Sharing of Java Applets: A Replicated Approach. 1997 Symposium on User Interface Software and Technology, New York, NY, ACM Press.

[2] Chabert, A., Grossman, E., Jackson, L., Pietrowicz, S., and Seguin, C. (1998). Java Object-Sharing in Habanero. *Communications of the ACM*, Vol. 41 # 6, June 1998, pp 69-76.

[3] Constantino-González, M.A. and Suthers, D.D. (2000). A Coached Collaborative Learning Environment for Entity-Relationship Modeling. In Gauthier, G., Frasson, C. and VanLehn, K. (Eds.) *Intelligent Tutoring Systems Proceedings of the 5th International Conference, ITS 2000*, Montreal, June, pp 324-333. Available: http://lilt.ics.hawaii.edu/lilt/papers/COLER-ITS00.pdf

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, 1995.

[5] Suthers, D.D. and Jones, D. (1997, August). An Architecture for Intelligent Collaborative Educational Systems. In B. du Boulay, R. Mizoguchi (Eds.) *8th World Conference on Artificial Intelligence in Education (AIED'97), pp. 55-62.*

[6] Suthers, D.D. Toth, E. E. and Weiner, A. (1997). An integrated approach to implementing collaborative inquiry in the classroom. In *Proc. of the 2nd International Conference on Computer Supported Collaborative Learning (CSCL'97)* (pp. 272-279). Toronto.

[7] Tewissen, F., Baloian, N., Hoppe, U., and Reimberg, E. (2000). "MatchMaker": Synchronising Objects in Replicated Software-Architectures. *Proc. 6th Int. Workshop on Groupware, CRIWG 2000,* Madeira, Portugal, 18 - 20 October 2000, IEEE CS Press.