원격 교육 시스템의 화이트보드 구현시 고려 사항

이 재 호* · 김 종 훈**
인천교육대학교 컴퓨터교육과*, 한국전자통신연구원 운영체제 연구팀**

요 약

컴퓨터 하드웨어의 발전과 초고속의 네트워크의 출현으로 지리적으로는 떨어져 있지만 컴퓨터 네트워크로 연결되어 있는 교사와 학생간에 동기식 학습이 가능하게 되었다. 전통적인 원격 교육과 비교하여 볼 때, 이러한 학습은 매우 높은 수준의 상호작용과 상호 협력을 제공하지만 전통적인 교실에서의 수업과 비교하여 볼 경우에는 다음과 같은 매우 중대한 결점들을 지니고 있다. 오늘날의 동기식 학습 시스템은 오디오, 비디오, 문서 편집을 제공하는 비디오 컨퍼런싱 기술에 주로 의존하고 있으며, 교수 과정의 조절, 손을 드는 행위, 지적 표시 등과 같은 특별한 교수 방법은 크게 고려하지 않고 있다. 이에 대두된 개념이 공유 화이트보드이다. 화이트보드는 전통적인 칠판의 전자적인 대체물로 이를 통해 원격 학습에 임하는 교사와 학생들이 전통적인 교실의 칠판의 이점을 취할 수 있게 된다. 따라서 본 논문에서는 화이트보드 구현시 고려 사항인 시스템 구조와 동시성 제어 방법에 대해 다양하게 살펴본다.

Implementation Issues in Whiteboards of Distance Learning Systems

Jaeho Lee* · Jonghoon Kim**
Inchon National University of Education*, ETRI**

ABSTRACT

The advent of powerful hardware and advances in high speed networks enabled synchronous learning, with teachers and students being geographically distributed but connected via computer networks. Today's synchronous learning systems are mainly based on video conferencing technology which provides audio, video, and joint editing of documents only, but does not take into account the specific requirements of teaching, for instance, controlling the course of instruction, raising hands, or reference pointing. A shared whiteboard is often the core part of these systems. Therefore, in this paper, we look into implementation issues in whiteboards of distance learning systems. Particularly, system architectures and concurrency control mechanisms are considered.

1. 서 론

컴퓨터 하드웨어의 발전과 초고속의 네트워크[1, 2]의 출현으로 지리적으로는 떨어져 있지만 컴퓨터 네트워크로 연결되어 있는 교사와 학생간에 동기식 학습(synchronous learning)이 가능하게 되었다. 전통적인 원격 교육과 비교하여 볼 때, 이러한 학습은 매우 높은 수준의 상호작용(interactivity)과 상호 협력(collaboration)을 제공하지만 전통적인 교실에서의 수업과 비교하여 볼 경우에는 다음과 같은 매우 중대한 결점들을 지니고 있다. 오늘날의 동기식 학습 시스템은 오디오, 비디오, 문서 편집을 제공하는 비디오 컨퍼런싱 기술에 주로 의존하고 있으며, 교수 과정의 조절, 손을 드는 행위, 지적 표시 등과 같은 특별한 교수 방법은 크게 고려하지 않고 있다. 이에 대두된 개념이 공유 화이트보드[3, 4]이다. 화이트보드는 전통적인 칠판의 전자적인 대체물로 이를 통해 원격 학습에 임하는 교사와 학생들이 전통적인 교실의 칠판의 이점을 취할 수 있게 된다. 원격 교육 시스템과 같이 다중 사용자가 동시에 사용하는 화이트보드는 시스템의 구조에 따라 기본 디자인 특성이 변화될 수 있다. 시스템의 구조는 참여자들간의 동작이나 이벤트를 전달하는 전체적인 작업 흐름을 결정하는 가장 중요한 요소다. 또한 다중의 사용자가 공유 데이터에 상호 작용을 하는 화이트보드에서의 동시성 제어 방법은 이벤트 충돌을 예방하고 데이터의 일관성을 유지할 수 있는 중요한 이슈이다. 따라서 화이트보드를 개발할 때에는 가장 먼저 최적의 시스템 구조와 동시성 제어 방법을 고려해야 한다. 따라서 본 논문에서는 화이트보드 구현시 고려 사항인 시스템 구조와 동시성 제어 방법에 대해 다양하게 살펴본다.

본 논문의 구성은 다음과 같다. 우선 Ⅱ장에서는 기존의 화이트보드들에 대해 살펴보고 Ⅲ장에서 시스템 구조에 대해 살펴보고 Ⅳ장에서 동시성 제어 방법에 대해 설명한다. 그리고 마지막으로 Ⅴ장에서 결론을 맺는다.

2. 관련 연구

WScrawl[5]은 Motif 1.1과 X 윈도우 시스템에서 개발된 간단하면서도 강력한 다중 사용자를 위한 공유 스케치패드로 중앙 집중 시스템 구조로 되어 있다. X 윈도우에서는 사용자들이 여러 개의 디스플레이를 열고 각각의 워크스테이션의 입력을 읽어서 그래픽 표현으로 화면에 출력하는 것이 가능하다. WScrawl과 같은 그룹웨어는 모든 사용자의 입력 스트림을 모니터 하여 입력 스트림 중에서 마우스 이동과 같은 입력 이벤트를 처리하고 모든 디스플레이를 그 결과에 따라 변경하는 중앙 집중 구조로 되어 있다. Rendezvous[6]는 중앙 집중 구조로 되어 있는 하나의 추상 데이터 모델을 관리하는 방법을 사용하는 그룹웨어 툴킷이다. 하나의 추상 데이터 모델의 다양한 뷰 모델은 각 사용자의 화면에서 다양한 형태로 보여질 수 있다. Rendezvous는 하나의 추상 데이터 모델과 여러 개의 뷰 모델을 하나의 프로세서에서 관장하는 중앙 집중 구조로 되어 있다. 개발자들에 의하면 원래의 추상 모델은 언제나 정확한 상태를 유지하기 때문에 이 추상 모델에서 유도되는 여러 개의 뷰 모델은 언제나 정확한 상태를 갱신된다고 한다. 그러나, 이 시스템은 속도가 느리다는 단점을 갖고 있다. 따라서 개발자들은 추상 데이터 모델은 중앙에 놓여 있고 각각의 뷰 모델은 각 호스트에 복제되어 있는 세미-복제 방법을 제안하고 있다. Rendezvous 개발자 중의 한 명인 Patterson은 세미-복제 하이브리드 구조를 연구하여 복제 구조의 컴포넌트로 사용할 중앙 집중 "Notification Server"를 고안하기도 하였다. 복제 구조로 되어 있는 GroupDesign은 Applet LocalTalk이나 EtherTalk으로 연결된 Applet Macintosh과 Ethernet/IP로 연결된 UNIX 워크스테이션에서 실행되는 다중 사용자를 위한 그리기 툴이다. 복제된 애플리케이션의 동작과 데이터를 제어하기 위해 ORESTE 알고리즘을 고안하였다[7]. JAMM은 James "Bo" Begole, Craig A. Struble, Alifford A. Shaffer가 개발한 다중 사용자가 동시에 자바 애플릿을 사용할 수 있는 실험적인 시스템으로 완전 복제 구조로 되어 있다. 다시 말해, 보통의 복제 구조 시스템은 각 사용자들간의 통신을 중개하는 통신 에이전트를 갖고 있다. 그러나 JAMM은 중앙에 에이전트와 같은 추가적인 프로세

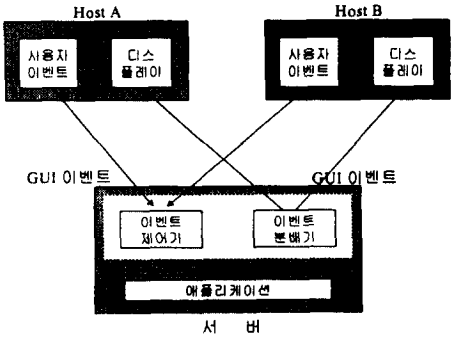
서가 없이 완전히 독립적으로 존재하는 복사된 애플리케이션들만으로 동작할 수 있다. JAMM는 JDK(Java Developer Kit) 1.1 베타 버전을 수정한 프로토타입 Collaboration Transparency 시스템이다. JAMM은 복제 구조로 되어 있으며 각 사용자의 이벤트를 도중에 인터셉트하여 컨퍼런트 에이전트를 통해 전체 참여자에게 전달하는 이벤트 브로드캐스팅 방법을 사용한다[8, 9].

3. 시스템 구조

본 장에서는 시스템 구조에 대해서 살펴본다.

3.1 중앙 집중(Centralized) 구조

중앙 서버가 분산된 참여자들의 입력과 출력을 제어하는 중앙 집중 구조에서는 <그림 1>과 같은 디스플레이 브로드캐스팅을 사용한다. 각 호스트에 있는 클라이언트 프로세서는 발생된 사용자 이벤트를 중앙 서버로 전달하고, 서버는 전달된 이벤트들을 처리한다. 이제 서버는 이벤트의 처리 결과 발생되는 화면의 변경 사항을 다시 모든 클라이언트로 전달하고 클라이언트 호스트들은 중앙 서버가 전달하는 변경된 디스플레이 정보를 화면에 출력한다. 결국, 모든 클라이언트 호스트들은 중앙 서버와 동일한 데이터를 갖게 되기 때문에 모든 클라이언트 데이터의 일치성을 유지하기 쉽고 동시성 제어가 수월하다[10].

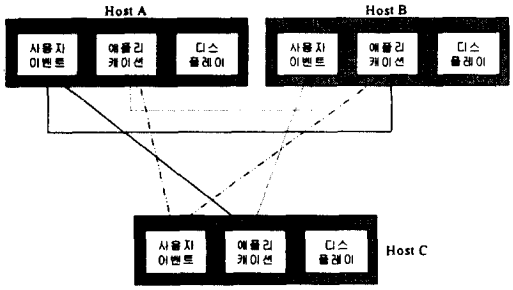


<그림 1> 디스플레이 브로드캐스팅

그러나 디스플레이 브로드캐스팅을 사용하여 중앙의 애플리케이션의 실행 결과의 디스플레이가 모든 나머지 사이트들로 전송되어야 하기 때문에 중앙 서버에 많은 부하가 발생하게 된다. 또한 참여자의 모든 입력을 받아서 처리하고 결과를 재 전송해야 하기 때문에 네트워크나 프로세스상의 병목현상(bottleneck)이 발생될 수 있고 각 클라이언트에 대한 접속이 실패할 경우에 문제가 발생할 수 있으며 참여자의 숫자가 늘어날 경우에 피드백이 지연되는 취약점도 있다. 이 구조는 디스플레이와 이벤트 처리 과정이 분산되어 있는 X 윈도우 시스템과 같은 분산 윈도우 시스템에서나 적당하며 현재의 네트워크 대역폭과 전송 속도를 고려할 때 이 구조를 자바 환경에서 현실화하기는 어렵다.

3.2 복제(Replicated) 구조

복제 구조는 중앙 집중 구조와 달리 각 로컬 호스트 사이트에 공유 애플리케이션이 복사되어 있는 구조로 모든 호스트의 애플리케이션은 나머지 호스트와 직접 통신할 수 있도록 동기화되어 있다. 복제 구조에서는 변화된 화면 정보가 아니라 화면과 데이터 변화를 유발하는 이벤트 자체를 전달하는 이벤트



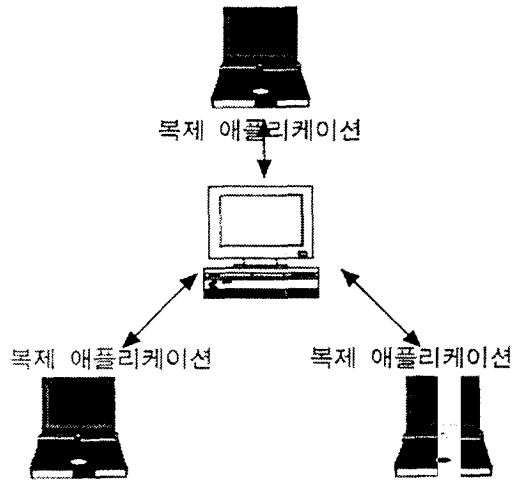
<그림 2> 이벤트 브로드캐스팅

브로드캐스팅 방법을 사용한다. 이벤트 브로드캐스팅의 전체적인 흐름은 <그림 2>와 같다.
복제 구조는 <그림 2>와 같이 간단한 GUI 이벤트만 전달하기 때문에 네트워크나 프로세스의 부하를 줄일 수 있다는 장점을 갖고 있다. 특히 서버를 거치

지 않기 때문에 이벤트의 전달이 빠르고 디스플레이를 변경하는 프로그램의 핵심 부분을 각 사이트가 갖고 있기 때문에 일인용 애플리케이션을 응용하기가 쉽다. 단, 모든 사이트의 이벤트와 애플리케이션 동작 상태를 동일하기 유지하기 위한 동시성 제어가 복잡하며 일관성 유지가 매우 어렵다는 단점이 있다.

3.3 하이브리드(Hybrid) 구조

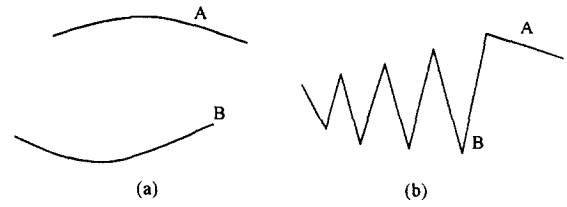
중앙 집중 구조와 복제 구조의 사이에 있는 구조는 하이브리드(hybrid) 혹은 세미-복제 하이브리드(semi-replicated hybrid) 구조이다. 하이브리드 구조는 <그림 3>과 같이 각 지역 호스트에 복제된 애플리케이션이 존재하는 복제 구조의 특징을 지니고 있으며, 지역 애플리케이션의 기능과 별개로 사용자들의 인터페이스를 관리하는 중앙의 서버가 존재하여 중앙 집중 구조의 특징도 함께 지니고 있다. 하이브리드 구조에서는 일관성을 유지하기 위해 공유되는 정보는 중앙 서버가 관리하고, 화면 정보나 사용자 이벤트에 대한 처리는 복제된 애플리케이션이 담당한다. 하이브리드 구조는 이질적인 환경에서 적합하게 사용될 수 있지만, 구현 시에 애플리케이션의 동작과 별도로 서버와의 인터페이스 부분을 분리해야 하는 문제점이 있다.



<그림 3> 하이브리드 구조

4. 동시성 제어 방법

여러 사용자가 화면과 데이터를 공유하고 그룹 활동을 하는 공유 시스템에서는 데이터의 일치성이 특히 중요하다. 특히 자바 클래스를 각 호스트마다 로드하여 실행되는 자바 기반 공유 시스템에는 세션 참가 인원수만큼 자바 애플릿의 복사본이 사이트마다 존재하게 되고 화면에 보이는 객체도 사이트만큼 존재하게 된다. 따라서 한 사람들의 동작이 네트워크를 통해 다른 참가자에게 전달되어 화면 디스플레이를 변경되기까지는 시간차가 발생할 수 있다.



<그림 4> 이벤트 충돌

또한 각 사이트의 이벤트들이 정확한 순서로 전달되지 않거나 연속된 이벤트들이 atomic하게 처리되지 않는다면 <그림 4>와 같은 이벤트 충돌이 발생할 수도 있다. <그림 4>의 (a)는 공동 작업 참여자 A와 B가 비슷한 시간에 각각 곡선을 그렸을 때, 두 사람의 이벤트가 독립적으로 처리되어 사용자들의 의도를 충족시키는 화면을 얻은 결과이다. 반면 <그림 4>의 (b)는 A와 B가 발생한 이벤트들이 교차되어 전달된 모습의 이벤트 충돌의 단적인 예를 보여 준다. 따라서 복제되어 있는 데이터들의 일치성을 유지하려면 기본 시스템 구조에 적합한 동시성 제어 기법을 사용해야 한다. 공동 작업을 지원하는 화이트보드에서 자주 사용되는 동시성 제어 방법으로는 발언권 제어 방법, Serialization, Locking이 있다.

4.1 발언권 제어

데이터 일치성을 유지할 수 있는 가장 쉬운 방법은 발언권 제어(Floor Control)방법으로 발언권을 얻은 한 명의 사용자만이 공동 작업 세션 전체에 대한 액세스 권한과 데이터 수정 권한을 갖는다. 이 방법을 사용할 경우에는 사용자와 시스템간의 상호작용에 의해 발언권 소유자가 결정되기 때문에, 사용자가 시스템에 발언권을 요청(request)하고 반환(release)할 수 있는 다이얼로그와 같은 인터페이스가 필요하다. 일반적으로 이 방법은 서버가 발언권 관리를 담당하여 사용자들의 발언권 요청과 반환에 응답하는 중앙 집중 구조에서 사용된다.

4.2 Serialization

Serialization은 데이터를 변경하는 이벤트를 시스템이 내부적으로 순서를 정함으로써 모든 복제된 데이터에 대한 일치성을 유지하는 방법이다[11]. 이 방법은 크게 두 가지로 나누어 볼 수 있다. 첫 번째는 전달되는 이벤트가 잘못된 순서로 전달될 수 있기 때문에 데이터 불일치를 감지하고 수정해야 하는 Optimistic Serialization이다. 두 번째 방법은 하이브리드 구조이나 세미-복제 구조에서 사용되는 방법으로 중앙의 서버가 이벤트를 분배하여 이벤트들이 언제나 올바른 순서로 전달되기 때문에 Optimistic 방법과 같은 추가적인 수정 작업이 필요 없는 Non-optimistic Serialization 방법이다.
그러나 Serialization을 사용하면 데이터 일치성은 보장할 수 있다라고 하더라도. 다음과 같은 사용자들 사이의 인터페이스 문제가 발생할 수 있다.

● 이상 동작의 발생 가능성이 높다.
공동 그래픽 편집기에 하나의 원이 있을 때, A는 왼쪽으로 움직이려 하고 B는 오른쪽으로 움직이려 한다. 사용자 입장에서는 A와 B가 개별적으로 수행했지만, 시스템 내부적으로는 두 동작이 거의 동시에 발생하여, 결과적으로 원은 두 사용자가 모두 예상치 못한 결과로 움직이게 된다. 따라서 이벤트가 정확한 순서대로 전달된다고 하더라도 사용자들간에 공유한 데이터에는 각 사용자가 생각하지 않은 이상 동작이 발생할 수 있다.

4.3 Locking

Locking은 여러 공동 작업자들 중에 한번에 단 한 사람만 공동 작업장에 그림을 그리거나 문자를 입력하도록 하는 가장 고정적이며 안전한 방법이다[11]. 한 사람만이 실질적인 권한을 갖는다는 측면에서 발언권 제어와 유사하지만, Locking은 Serialization과 마찬가지로 록(Lock)을 얻기 위한 사용자들의 추가적인 동작이 없더라도 시스템이 자동으로 발생된 이벤트에 따라 록을 할당하거나 해제한다. 그러나 Locking을 사용할 경우에도 다음과 같은 문제점들을 고려해야 한다.

● 우선은 록의 단위 크기를 결정해야 한다.
록의 단위에 따라 사용자들의 인터페이스가 변하기 때문이다. 사용자 관점에서는 한 사람이 공동 작업장이나 공동 에디팅 문서 전체에 대한 록을 갖는 것이 가장 좋다. 그러나, 그룹웨어 디자인의 관점에서 판단한다면 자연스런 인터페이스를 제공하려면 여러 사람이 동시에 접근하여 그래픽 객체를 액세스할 수 있어야 한다. 그러한 경우 록의 단위를 세분하여 그래픽 객체 하나씩 심지어 선의 양 끝점을 록의 단위로 삼기도 한다.

● 록을 받기 위한 지연시간이 너무 길면 안 된다.
한 사람이 록을 갖게 되면 다른 사람에게 록이 넘어갈 때까지 계속 그래픽 객체들을 사용할 수 있다. 이때 다른 사람들이 록을 받기 위해 기다려야 하는 시간이 그다지 짧지 않다면, 크게 문제될 것은 없다. 그러나 그 지연시간이 길어지면 응답시간도 길어지고 인터페이스를 부자연스럽게 만들게 된다.

Serialization과 Locking 방법은 발언권 관리를 담당하는 중앙의 서버가 없이 각 호스트의 복제된 애플리케이션이 사용자들간의 인터페이스를 조절하는 복제 구조에서 주로 사용된다.

5. 결 론

본 논문에서는 원격 교육 시스템의 화이트보드 구현에 있어서 고려해야 할 사항들을 살펴보았다.

화이트보드는 전통적인 칠판의 전자적인 대체물로 이를 통해 원격 학습에 임하는 교사와 학생들이 전통적인 교실의 칠판의 이점을 취할 수 있게 된다. 다중 사용자가 동시에 사용하는 화이트보드는 시스템의 구조에 따라 기본 디자인 특성이 변화될 수 있다. 시스템의 구조는 참여자들간의 동작이나 이벤트를 전달하는 전체적인 작업 흐름을 결정하는 매우 중요한 요소이다. 또한 다중의 사용자가 공유 데이터에 상호 작용을 하는 화이트보드에서의 동시성 제어 방법은 이벤트 충돌을 예방하고 데이터의 일관성을 유지할 수 있는 중요한 이슈이다. 그러므로 본 논문에서는 이와 같은 시스템 구조와 동시성 제어 방법에 대해 다양하게 살펴보았다.

참고문헌

[1] D. E. McDysan and D. L. Spohn, *ATM: Theory and Application*, McGraw-Hill, 1995.

[2] N. J. Boden et al., "Myrinet: A Gigabit-per-Second Local Area Network," *IEEE Micro*, 15(1):29-36, February 1995.

[3] Ch. Bacher and Th. Ottmann, "Tools and Services for Authoring on the Fly," In *Proceedings of ED-MEDIA '96*, 1996.

[4] Werner Geyer, Wolfgang Effelsberg, "The Digital Lecture Board-A Teaching and Learning Tool for Remote Instruction in Higher Education," In *Proceedings of ED-MEDIA'98*, 1998.

[5] B. Wilson, "WSCRAWL 2.0: A Shared Whiteboard Based on X-Windows," In *Designing Groupware for Real Time Drawing*, S. Greenburg, S. Hayne and R. Rada ed. McGraw Hill.

[6] S. Greenberg, "Sharing Views and Interaction with Single-user Applications," In *Proceedings of the ACM/IEEE Conference on Office Information Systems*, pages 227-237, Camb-ridge, Massachusets, 1990.

[7] M. Beaudouin-Lafon and A. Karsenty, "Transp-arency and Awareness in a Real-Time Grou-pware System," In *Proceedings ACM Symp-osium on User Interface Software and Techn-ology UIST'92*, November 1992.

[8] James "Bo" Begole, "Flexible Collaboration Transparency," dissertation research proposal, approved April 30, 1997.

[9] James "Bo" Begole, Craig A. Struble, and Clifford A. Shaffer, "Leveraging Java Applets: Toward Collaboration Transparency in Java," *IEEE Internet Computing*, 1(2):57-64, March-April 1997.

[10] W. Renhard, J. Scweitaer, G. Volksen, and M. Weber, "CSCW Tools: Concepts and Architec-tures," *IEEE Computer*, pages 28-36, May 1994.

[11] S. Greenbug and D. Marwood, "Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface," In *Proceedings of the ACM CSCW Conference on Computer Supported Cooperative Work*, pages 22-25, October 1994.