분산 객체 지향 소프트웨어 개발 환경에서 동시성 향상을 위한 공유 데이타 분할 모델

(Shared Data Decomposition Model for Improving Concurrency in Distributed Object-oriented Software Development Environments)

김 태 훈 † 신 영 길 ††
(Tae-Hoon Kim)(Yeong Gil Shin)

요 약 본 논문에서는 다중 사용자를 지원하는 분산 소프트웨어 개발 환경에서 동시성을 향상시킬 수 있는 공유 데이타 분할 모델을 제안한다. 제안된 모델에서는 공유 데이타에 해당하는 목표 소프트웨어 시스템을 프로젝트 역할을 기반으로 분할한 후, 분산 환경의 각 클라이언트에 분산시키고 이를 다시 뷰(view) 객체와 코어(core) 객체로 분할하여 저장한다. 여러 클라이언트가 참여하는 협동 작업에서는 뷰 객체만을 각 클라이언트에 복사(replicate)하여 빠른 응답 시간을 보장하도록 하고, 코어 객체는 하나의 클라이언트에만 저장한 한 후 역할 단위의 잠금(locking) 기법을 이용하여 불일치 문제가 발생하지 않도록 하였다. 실험 결과, 제안된 모델은 기존 도구들에서 사용하는 클래스 단위의 잠금 기법보다 12~18%의 성능 향상을 보였고, 클라이언트의 수가 증가하더라도 응답 시간이 급격히 증가하지 않아 확장성(scalability)이 뛰어난 특징을 보였다.

Abstract This paper presents a shared data decomposition model for improving concurrency in multi-user, distributed software developments. In our model, the target software system is decomposed into the independent components based on project roles to be distributed over clients. The distributed components are decomposed into view objects and core objects to replicate only view objects in a distributed collaboration session. The core objects are kept in only one client and the locking is used to prevent inconsistencies. The grain size of a lock is a role instead of a class which is commonly used as the locking granularity in the existing systems. The experimental result shows that our model reduces response time by 12~18% and gives good scalability.

1. 서 론

소프트웨어의 복잡도가 증가함에 따라 소프트웨어 개발을 위해 많은 개발 인력이 필요하게 되었고 소프트웨어 개발 환경도 개발자들의 요구 사항을 반영하여 다양한 기능을 지원하게 되었다. 이러한 소프트웨어 개발 환경은 소프트웨어 개발 인력이 모두 한 곳에 모여 개발 프로젝트를 진행한다는 가정 하에 독립적(stand alone) 방식으로 발전되어 왔다. 즉, 하나의 고성능 워크스테이션에 소프트웨어 개발 환경을 구축하고 프로젝트 구성원들은 모두 여기에 접속하여 개발 작업을 수행하는 형태를 취해 왔다.

하지만 네트워크가 고속의 데이타 전송을 지원하고 컴퓨터의 성능이 향상됨에 따라 이러한 독립적 방식의 단점을 개선한 분산 소프트웨어 개발 환경의 유용성이 부각되고 있다. 분산 소프트웨어 개발 환경은 소프트웨어 개발 환경을 네트워크로 연결된 여러 대의 컴퓨터에 분산시켜 전체적인 성능을 향상시킬 수 있으며 ATM(Asynchronous Transfer Mode) 네트워크를 이용하면 지리적으로 멀리 떨어져 있는 컴퓨터에서도 서로 데이타를 주고받을 수 있다.

최근에는 지리적으로·분산된 사용자들이 네트워크로 연결된 컴퓨터를 통하여 서로의 정보를 공유하며 공동

† 비 회 원 : Pumpkin Networks Inc. 연구원
hoon@pumpkinnet.com
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
yshin@cglab.snu.ac.kr
논문접수 : 1999년 5월 13일
심사완료 : 2000년 6월 5일

작업을 수행하도록 지원하는 컴퓨터 지원 협동 작업(CSCW, Computer Supported Cooperative Work)에 대한 연구가 활발하게 진행되고 있으며 이를 지원하는 그룹웨어(groupware)와 그룹웨어 구현을 지원하는 툴킷(toolkit)에 대한 연구가 활발히 진행되고 있다. 분산 소프트웨어 개발 환경에서는 여러 명의 사용자들이 네트워크로 연결된 클라이언트에서 데이타를 공유하며 협동 작업을 수행하므로 공유 데이타에 대한 불일치(inconsistency) 문제가 발생하지 않도록 하는 동시성 제어(concurrency control) 기법이 필수적으로 요구된다.

본 논문에서는 General Electric R & D 센터에서 Rumbaugh 등에 의해 제안된 객체 모델링 기법 OMT(Object Modeling Technique) [1]을 지원하는 객체 지향 CASE 도구에 CSCW 기능을 첨가시킨 분산 객체 지향 소프트웨어 개발 환경 DOOD(Distributed Object-Oriented Designer) [2]의 개발 과정에서 분석된 요구 사항을 바탕으로 다중 사용자를 지원하는 분산 소프트웨어 개발 환경에서 동시성(concurrency)을 향상시킬 수 있는 모델을 제시하고, 제안된 모델이 분산 동기적 협동 작업에서 동시성을 향상시키고 빠른 응답 시간을 보장할 수 있음을 보인다.

2. 관련 연구

2장에서는 분산 소프트웨어 개발 환경에서의 공유 데이타 분할 모델과 관련된 기존의 연구 결과를 기술한다. 2.1에서는 분산 소프트웨어 개발 환경을 비롯한 CSCW 시스템의 구현에 이용되는 일반적인 런타임 구조와 동시성 제어 기법에 대해 기술하고, 2.2에서는 CSCW 시스템에서의 협동 작업에 널리 사용되는 공유 데이타 모델인 MVC (Model-View-Controller) 모델의 개요를 기술한다. 2.3에서는 다중 사용자가 데이타 공유를 통해 협동 작업을 수행하는 대표적인 시스템인 다중 사용자용 편집기와 다중 사용자용 소프트웨어 개발 환경에서 불일치 문제를 해결하기 위한 관련 연구들에 대해 기술하였다.

2.1 공동 작업을 위한 런타임 구조(Runtime Architecture) 및 동시성 제어 기법

분산 환경의 CSCW 시스템은 런타임 구조로 중앙 집중(centralized) 구조 또는 분산 복사(replicated) 구조를 가진다[3]. 중앙 집중 구조는 보통 비관적 동시성 제어 기법(pessimistic concurrency control method)과 같이 사용되며 서버에 공유 데이타를 저장하고 이를 여러 클라이언트가 공유하는 구조이다. 이러한 런타임 구조는 동시성 제어가 용이하고, 잠금(locking)등의 기법을 사용하여 공유 객체를 항상 일관성 있는 형태로 유지할 수 있으며 구현이 쉽다는 장점을 가진다. 반면 공유 데이타의 접근을 위해 네트워크를 통해 서버의 데이타에 접근해야 하므로 네트워크 지연 시간(delay)으로 인해 응답 시간이 길어지는 단점이 있다. 비관적 동시성 제어 기법은 불일치 문제가 발생하지 않도록 미리 방지하는 방식으로 불일치 문제가 발생하면 이를 해결하기 어렵거나 그 결과가 치명적일 경우 주로 사용한다.

분산 복사(replicated) 구조는 주로 낙관적 동시성 제어 기법(optimistic concurrency control method)과 같이 사용되며 각 클라이언트가 공유 데이타를 복사(replicate)하여 사용자의 입력에 대해 서버에 접근하지 않고 클라이언트에 복사된 데이타에 접근, 수정한 후 다른 클라이언트들에게 수정된 내용을 브로드캐스트(broadcast)하는 방식이다. 이러한 런타임 구조는 서버까지의 네트워크 지연 시간이 없기 때문에 응답 시간이 빠른 장점을 가진다. 하지만 여러 클라이언트가 각기 복사본을 가지고 있으므로 불일치 문제가 발생하기 쉽고 이를 위해 되돌리기(undo) 작업이 필요한 경우가 발생한다. 또한 연속적 되돌리기(cascading undo)가 발생할 수 있으므로 오랜 시간의 작업 내용을 포기해야만 하는 경우가 발생할 수 있다. 낙관적 동시성 제어 기법은 불일치 문제가 자주 발생하지 않는다는 가정 하에 여러 개의 복사본을 가지고 수정 작업을 수행한 후, 불일치가 발생할 경우 이를 되돌리기(undo)하는 방식으로 빠른 응답 시간이 요구되는 경우 주로 쓰인다.

CSCW 시스템의 런타임 구조는 프로그래머가 응용 프로그램의 성격에 따라 위의 두 가지 중의 하나를 선택하여 구현하는 경우가 대부분으로, 절대적으로 우위에 있는 구조는 존재하지 않는다[4]. 다소의 불일치 문제가 발생하더라도 응답 시간이 중요한 경우에는 분산 복사(replicated) 구조를, 일관성 유지가 중요하고 네트워크 지연 시간이 크지 않을 경우에는 중앙 집중 구조를 선택하는 것이 일반적이다.

2.2 MVC(Model-View-Controller) 구조

MVC 구조는 Smalltalk 프로그램 언어에서 처음 도입된 개념으로 협동 작업에서 여러 사용자가 동일한 데이타에 대한 뷰를 쉽게 공유할 수 있으므로 그룹웨어에서 협동 작업을 위한 데이타 공유 모델로 널리 쓰인다. 그룹웨어에 이용되는 MVC 구조에서는 일반적으로 그림 1과 같이 응용 프로그램의 데이타(Model)를 입력 관련 프로그램 코드(Controller), 디스플레이 관리 프로그램 코드(View)와 분리하여 그룹웨어 사용자들은 각자의 뷰와 컨트롤러를 가지고 작업을 수행한다. 이러한 MVC

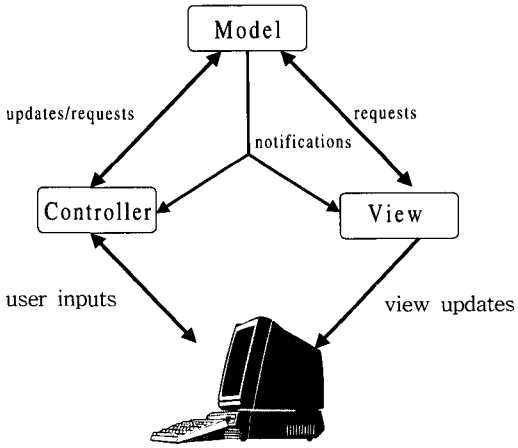


그림 1 MVC 구조

구조에서 프로그래머는 각 사용자의 뷰에 대한 일관성을 유지하기 위한 코드를 작성할 필요가 없는 장점이 있지만 모델과 뷰, 컨트롤러 사이의 메시지 전송이 네트워크를 통해 일어나므로 응답 시간이 길어지는 단점이 있다[5].

2.3 다중 사용자의 데이타 공유

여러 명의 사용자가 데이타를 공유하여 동시에 접근, 수정하는 시스템으로는 다중 사용자용 편집기(multi-user editor)가 대표적이다. 다중 사용자용 편집기에서는 공유 대상이 되는 데이타를 문서(document), 장(chapter), 절(section), 문단(paragraph), 줄(line), 문자(character) 등의 단위로 나누어 각 사용자가 접근, 수정할 수 있도록 관리하며 이를 위해 잠금 등의 기법을 사용한다.

Duplex[6]에서는 문서를 계층적 구조의 독립적인 세그먼트(segment)로 분할하여 각 사용자들이 각각의 독립된 세그먼트에 대한 수정 작업을 마친 후 합병(merge)하는 방식으로, 네트워크 지연 시간이 긴 경우에도 비동기적 협동 작업을 수행할 수 있도록 하였다. Duplex에서 데이타 공유의 기본 단위인 독립적 세그먼트는 문서의 특성에 따라 장 또는 절 등이 될 수 있는 유연한 특성을 가지며 서로 영향을 주지 않도록 독립적으로 분할되었기 때문에 합병 작업이 쉬운 특징을 가진다.

SPE(Snart Programming Environment) [7][8]에서는 소프트웨어 개발 환경에서 발생하는 불일치 문제를 인식(detect)하여 이를 로그(log)형태로 기록하고 다이어그램에 불일치가 발생한 부분을 표시하거나, 다이얼로그 등을 통해 사용자들에게 통보하여 불일치가 발생한 부분을 수정하도록 한다. 여러 사용자가 동시에 공유 데이타에 대해 수정할 필요가 있을 경우에는 각자 공유 데이타에 대한 독자적인 버전을 가지고 작업을 수행한 후 버전 합병(version merging)을 하는 방법으로 동시성 문제를 해결한다. SPE는 독립적(stand alone) 형태의, 다중 사용자용 소프트웨어 개발 환경으로 불일치가 발생한 후에 이를 인식하여 효과적으로 제거하기 위한 목적으로 개발되었다.

3. 분산 협동 작업을 위한 공유 데이타 분할 모델

3.1 요구 사항 분석

분산 객체 지향 소프트웨어 개발 환경에서는 개발 과정의 모든 단계에서 여러 명의 사용자가 목표 소프트웨어 시스템의 구성 객체에 대해 접근, 수정하는 작업이 빈번하게 일어나므로 잠금 등의 기법을 이용하여 불일치 문제가 발생하지 않도록 하여야 한다. 잠금 기법에서는 잠금 단위(locking granularity)를 정하고 사용자는 이에 따라 잠금 요청(locking request)을 잠금 관리자에 보내 원하는 객체에 대한 접근 권한을 얻어 작업을 수행한다. 잠금 단위가 클 경우 잠금 요청의 회수는 줄어들지만 잠금 거부(locking denial)가 발생하는 일이 자주 일어나므로 사용자는 원하는 작업을 수행하지 못하고 대기하여야 한다. 따라서 잠금 단위는 시스템의 특성에 맞게 적절하게 조정하여야 하며 동시성을 최대한 허용할 수 있도록 사용자의 입장에서 결정되어야 한다. 2.3에 기술한 SPE 시스템에서는 다중 버전(multi version)을 이용하여 어떤 객체에 대한 수정을 원하는 사용자는 독자적인 버전을 만들어 수정하고 나중에 다른 사용자가 작성한 버전과 합병하는 방법으로 불일치 문제를 해결한다. 이러한 경우 사용자에 대한 응답 시간은 빠르지만 나중에 버전 합병 과정을 거쳐야 불일치 문제가 완전히 해결되며 그 이전에는 여러 사용자가 각자 다른 버전을 가지므로 뒤늦은 참가자(late comer)가 세션에 참가하려고 할 경우에는 이를 지원하지 못하는 단점이 있다.

객체 지향 소프트웨어 개발 환경에서 사용자들 사이의 협동 작업은 주로 클래스 다이어그램과 프로그램 코드를 기반으로 이루어진다. 사용자들은 클래스 다이어그램에 대한 WYSIWIS(What You See Is What I See) 뷰[9]를 공유하면서 다른 사용자들과 클래스 다이어그램을 통한 협동 작업을 수행한다. 특정 클래스의 내부 구조에 접근할 필요가 있을 경우에는 클래스 다이어그램을 통해 해당 클래스의 소스 코드에 접근하는 방식으로 협동 작업을 수행한다. 클래스 다이어그램은 사

용자들이 관심을 가지는 클래스들의 상호 관계를 한 눈에 볼 수 있는 유용한 뷰이므로 실시간 협동 작업에서 유용한 수단이며 이에 대한 사용자 입력 및 상호 작용은 빠른 응답 시간 내에 처리될 수 있도록 하는 것이 바람직하다.

소프트웨어 개발의 초기 단계에서는 각 사용자들이 개인 작업 환경에서 독자적인 작업을 수행하는 일이 많지만 개발 후반부의 시스템 통합 단계에서는 동일한 객체에 대해 동시에 여러 사용자가 접근하는 일이 자주 발생한다. 클래스 다이어그램과 프로그램 코드를 통한 실시간 협동 작업은 이러한 시스템 통합 단계에서 유용하게 사용될 수 있는데 잠금 단위를 클래스로 고정할 경우에는 잠금 거부가 자주 발생하므로 동시성이 떨어지는 문제가 발생한다. 이를 위해 잠금 단위는 클래스보다 작은 단위로 설정될 수 있어야 한다.

사용자들 사이의 협동 작업은 특정한 세션(session)을 설정하여 각 사용자가 이에 참가(join)하는 형태로 이루어진다. 세션 참가는 각 사용자가 세션 관리자에 참가 요청을 하는 형태로 이루어지는데 세션이 진행되는 도중에 뒤늦게 참가하는 경우도 자주 발생한다. 이러한 경우 새로운 참가자도 현재 진행 중인 세션의 상태를 그대로 전송 받아 세션에 참가할 수 있도록 하는 기능도 제공되어야 한다. 또한 세션 참가자의 사정에 따라 진행 중인 세션을 잠시 중단시킨 후 나중에 재시작(resume)해야 하는 경우도 자주 발생하는데 이러한 세션 저장 및 재시작 기능도 제공되어야 한다.

본 논문에서는 다음과 같은 요구 사항을 만족시킬 수 있는 분산 객체 지향 소프트웨어 개발 환경의 공유 데이타 분할 모델을 제시하였다.

- 잠금 단위는 동시성을 최대화시킬 수 있도록 사용자 입장에서 유연하게(flexible) 결정될 수 있어야 한다.
- 사용자들 사이의 실시간 협동 작업은 주로 클래스 다이어그램과 프로그램 코드를 통해 이루어지며 특히 클래스 다이어그램에 대한 사용자 인터페이스는 빠른 응답 시간을 보장할 수 있어야 한다.
- 세션 진행 중에 뒤늦은 참가자가 세션에 참여할 수 있는 기능과 진행 중인 세션을 저장한 후 다시 재시작할 수 있는 기능을 제공하여야 한다.

3.2 공유 데이타 분할 모델 개요

본 논문에서는 분산 동기적 협동 작업에서 동시성을 향상시키기 위한 모델로 역할(role) 기반 분할과 뷰-코어 분할을 제안하는데 역할 기반 분할은 소프트웨어 개발 프로젝트에 참가하는 소프트웨어 개발 환경의 사용자 역할에 따라 목표 소프트웨어 시스템을 분할한 후 각 사용자의 클라이언트에 저장하여 협동 작업에서의 동시성을 향상시키기 위한 것이다. 소프트웨어 개발 과정에서 각 사용자의 역할은 서로 중복되지 않도록 독립적으로 정의되므로 이를 기반으로 분할한 후 협동 작업에 이용하면 공유 데이타의 접근을 위한 충돌 및 지연을 줄일 수 있다. 역할 기반 분할은 협동 작업이 이루어지는 도중에도 동적으로 수행될 수 있으므로 사용자들은 현재 상황에 가장 적합한 런타임 구조로 변형시킬 수 있다.

뷰-코어 분할은 MVC 모델에 기반한 분할 기법으로 MVC 모델에서 M(모델)에 해당하는 부분인 목표 소프트웨어 시스템의 클래스들을 뷰 클래스와 코어 클래스로 분할하여 뷰 클래스는 각 클라이언트에 복사본을 저장하고, 코어 클래스는 하나의 클라이언트에만 저장하여 공유하는 방식이다. 기본적인 MVC 모델과는 달리 분할된 코어 클래스는 수정 작업이 가장 빈번하게 일어나는 클라이언트에 저장되므로 빠른 응답 시간을 보장받을 수 있다.

이러한 공유 데이타 분할 모델에서는 그림 2의 (b)과 같이 MVC 모델의 M(모델)에 해당하는 부분을 프로젝트 역할에 따라 분할한 후, 각 클라이언트에 분산시킨다. 분산된 각 모델은 뷰-코어 분할에 의해 분할되는데 다음과 같은 소프트웨어 개발 생명주기를 기반으로 적용된다.

단계 1) 요구사항에 기초한 역할 기반 목표 소프트웨어 시스템 분할

목표 소프트웨어 시스템에 대한 요구사항(require--ments)을 바탕으로 목표 소프트웨어 시스템을 하향식(top-down)으로 분할하는 단계이다. 일반적으로 객체 지향 소프트웨어 시스템은 수백 개 이상의 클래스 집합

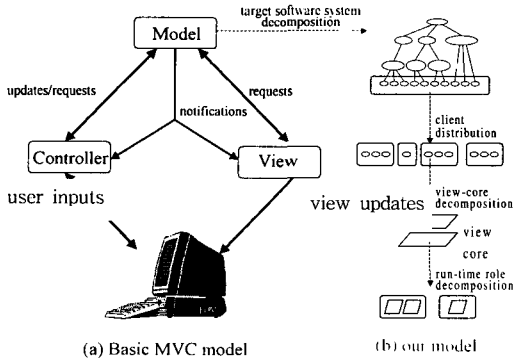


그림 2 기본적 MVC 모델과 개선된 공유 데이타 분할 모델

으로 구성되므로 이를 적당한 수준으로 추상화(abstraction)하여 나타내기 위한 메카니즘이 필요하다. 이를 위해 OMT[1]에서는 모듈(module)을, UML(Unified Modeling Language)[10]에서는 서브시스템(subsystem)과 모델(model)을 제시하고 있는데 UML에서의 서브시스템, 모델의 정의는 다음과 같다.

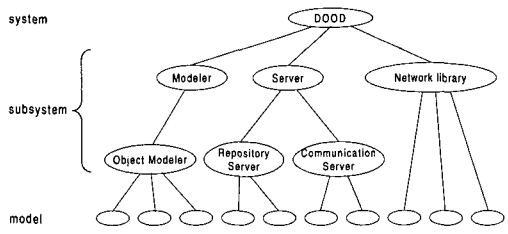
• 서브시스템 : 하향식 개발에 적합한 그룹 메카니즘(grouping mechanism)으로 구현(realization) 전에 그 행위(behavior)에 대한 요구사항을 표현한다. 서브시스템에 관한 기술(specification)은 그 서브시스템에 대한 상위 레벨의 API로 취급할 수 있다.

• 모델 : 시스템의 논리적 또는 행위에 대한 모델링과 같이 특정한 관점(viewpoint)에서의 시스템 추상화를 기술한다.

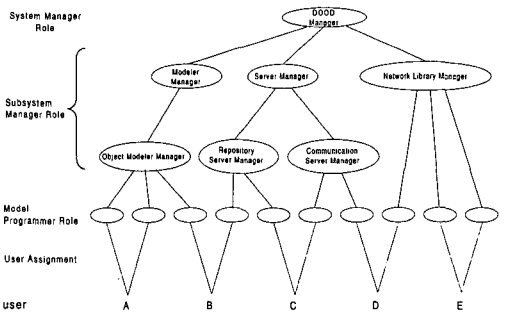
단계 1에서는 목표 소프트웨어 시스템을 서브시스템과 모델로 이루어진 계층 구조로 분할하기 위하여 그림 3과 같이 루트 노드부터 레벨을 증가시키면서 분할한다. 각 단계에서의 분할 기준은 다음과 같다.

• 분할된 각 노드에 유일한(unique) 프로젝트 역할을 지정할 수 있도록 한다.

• 분할된 노드들은 대부분 상호 작용이 노드 내에서 일어나도록 한다.



(a) Role-based System Decomposition



(b) Role Hierarchy Tree and User Assignment

그림 3 역할 기반 시스템 분할과 역할 트리

시스템 분할 결과는 트리 구조로 나타낼 수 있는데 할 트리의 루트 노드는 전체 시스템을, 중간(non-terminal) 노드는 서브시스템을, 터미널 노드는 모델을 나타낸다. 서브시스템은 요구사항에 기반을 둔 분할로 실제적인 클래스 구현 및 모델링 정보를 담고 있지 않으며 하위 노드의 분할 및 상호 작용에 대한 정보를 저장하고, 분할 트리의 터미널 노드에 해당하는 모델은 각 사용자에게 할당되어 모델링 및 실제적인 클래스 구현에 대한 정보를 저장한다.

시스템 분할 과정에서는 그림 3의 (b)와 같은 역할 트리가 생성되어 루트 노드와 중간 노드에는 서브시스템 관리자 역할이, 터미널 노드에는 모델 프로그래머 역할이 각각 대응된다.

단계 2) 각 사용자에게 모델 할당

프로젝트에 참여하고 있는 각 사용자에게 분할된 모델을 할당하여 해당 모델에 대한 모델링 및 구현을 담당하도록 한다. 이는 그림 3의 (b)와 같이 생성된 역할 트리의 터미널 노드에 해당하는 프로그래머 역할을 프로젝트 사용자에게 할당하는 방식으로 이루어진다. 할당된 각 모델은 사용자의 클라이언트에 분산되어 객체 모델링 도구, 코드 생성 도구 등을 통해 모델링, 구현된다.

단계 3) 각 해당 모델에 대한 모델링 및 코드 생성

역할을 할당받은 각 사용자는 해당 모델에 대한 모델링을 한 후 코드 생성 기능을 통해 코드를 생성하며, 이 때 뷰-코어 분할이 일어난다. 뷰-코어 분할에 의해 분할 저장된 모델은 실시간 협동 작업 중 클래스 다이어그램 및 프로그램 코드를 여러 사용자가 공유하려고 할 때 이용된다.

단계 4) 실시간 협동 작업 중 동적 역할 분할

각 사용자는 할당받은 모델에 대한 구현 도중에 다른 사용자와 협동 작업이 필요한 경우가 발생한다. 필요할 경우 자신의 역할 중 일부를 동적으로 분할하여 넘겨줄 수 있다. 단계 1에서의 분할은 요구사항에 기반한 초기 단계에서의 분할이므로 개발 과정 중에 분할 구조를 변경해야할 경우가 생긴다. 따라서, 협동 작업 중 동적 역할 분할이 지원되어야 한다.

3.3 역할 기반 분할

역할 기반 분할은 목표 소프트웨어 시스템의 구성요소인 객체들을 프로젝트 역할에 따라 서로 독립적인 부분으로 분할하는 과정으로 3.2에서 제시한 생명 주기의 초기 단계에서 수행되는 요구 사항에 기초한 목표 소프트웨어 시스템 분할과 모델링 및 코드 생성이 끝난 단계에서 협동 작업 중 발생하는 동적 역할 분할이 있다.

초기 단계의 요구 사항에 기초한 시스템 분할에서는

목표 소프트웨어 시스템을 그림 3과 같은 트리 구조의 서브시스템과 모델로 분할하는데 터미널 노드에 해당하는 모델은 "Graphics Primitive Programmer"와 같은 고유한 프로젝트 역할과 대응되어 사용자에게 할당된다.

사용자에게 할당된 프로그래머 역할은 그림 4와 같은 객체 모델로 정의할 수 있는데 뷰-코어 분할에 의해 분할된 코어 클래스들과 이들 사이의 관계를 모델링하여 클래스 다이어그램 형태로 표현한 역할 다이어그램(role diagram)으로 구성된다. 역할 다이어그램에서 뷰 클래스는 해당 코어 클래스와 연관(association) 관계를 가지며 사용자가 이를 통해 코어 클래스에 접근할 수 있도록 인터페이스 기능을 제공한다.

역할은 그림 4의 클래스 다이어그램에 나타난 것과 같이 코어 객체, 코어 객체 메소드, 코어 객체 속성들의 집합이다. 즉, 하나의 역할은 수십 개의 코어 객체들을 포함할 수도 있고, 코어 객체 내의 단일 메소드만을 포함할 수도 있는 유연한(flexible) 데이타 공유 단위이다. SPE[8]와 같은 기존의 소프트웨어 개발 환경에서는 데이타 공유의 최소 단위가 객체이므로 같은 객체 안의 서로 다른 메소드를 두 명의 사용자가 동시에 수정할 필요가 있을 경우에는 잠금 요청의 순서대로 차례차례 수정하거나 각 사용자가 서로 다른 버전을 가지고 편집한 후 나중에 합병해야 한다. 하지만 역할을 데이타 공유의 기본 단위로 사용하는 본 모델에서는 각각의 메소드를 역할로 지정할 수 있으므로 두 메소드를 두 명의 사용자들이 동시에 수정할 수 있다.

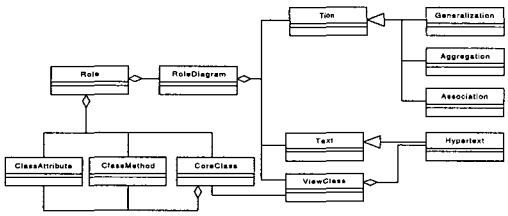


그림 4 역할(role) 객체 모델

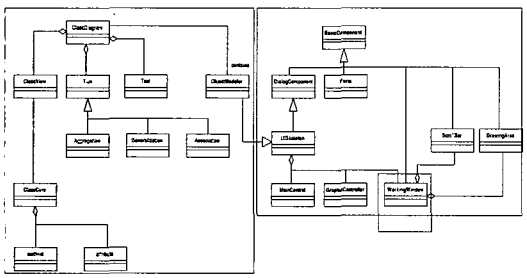


그림 5 클래스 다이어그램 상에 표시된 역할 범위

그림 5는 역할의 범위를 클래스 다이어그램 상에 표시한 것으로, 하나의 역할은 클래스 내부의 1개 메소드에서 수십 개의 객체 집합까지 다양한 단위로 구성될 수 있다.

3.4 뷰-코어 분할

뷰-코어 분할에서는 분산 객체 지향 소프트웨어 개발 환경에서 목표 소프트웨어 시스템의 데이타 객체들을 클래스 다이어그램에서의 x, y 좌표와 같은 다이어그램 관련 정보만을 저장하는 뷰 객체와 의미 정보(semantic information), 프로그램 코드와 같이 실제 구현의 핵심이 되는 정보를 저장하는 코어 객체로 분할한다. 뷰 객체는 각 클라이언트에 복사본이 저장되는데, 클라이언트의 객체 모델러(object modeler)는 뷰 객체들이 가진 정보만으로 클래스 다이어그램을 표시(display)할 수 있다.

뷰-코어 분할은 사용자가 클래스 다이어그램으로 모델링한 상태에서 "코드 생성"을 호출할 때 이루어진다. 사용자는 각 클래스의 속성(attribute), 메소드(method)와 그에 따른 변수 타입, 인자(argument)들을 클래스 다이어그램 상에 입력하므로 사용자가 생성한 뷰 객체와 자동 생성된 코어 객체로 쉽게 분리하여 저장할 수 있다.

뷰-코어 분할을 통해 얻을 수 있는 장점은 실시간 협동 작업이 필요할 경우 뷰 객체들은 협동 작업에 참여하는 각 클라이언트에 복사본을 저장하고, 코어 객체들은 하나의 클라이언트에만 저장하는 분산 복사(replicated) 구조와 중앙 집중 구조를 혼합한 하이브리드(hybrid) 런타임 구조에서 협동 작업이 가능하다는 것이다. 이러한 런타임 구조는 클래스 다이어그램을 통한 협동 작업에는 빠른 응답 시간을 보장하고, 코어 객체에 대한 접근, 수정에 대해서는 불일치 문제가 발생하지 않도록 예방할 수 있다.

협동 작업에 참여하는 각 사용자는 소유자(owner), 검토자(reviewer)와 같은 세션 내에서의 역할을 가지는데 소유자는 코어 객체에 대한 관리를 담당하고, 검토자는 뷰 객체만을 복사한 후 주로 클래스 다이어그램을 참조하여 협동 작업을 수행한다. 프로그램 코드의 내용을 참조할 필요가 있을 경우에는 뷰 클래스를 통해 접근한다. 이러한 소유자-검토자 형태의 협동 작업은 소프트웨어 개발 환경에서 자주 발생하는 협동 작업의 형태로 뷰-코어 분할은 이러한 형태의 협동 작업에 적절한 데이타 구조를 제공한다.

그림 6은 뷰-코어 분할에 의해 분할된 뷰 객체와 코어 객체를 이용하여 소유자 역할을 하는 사용자 A와 검

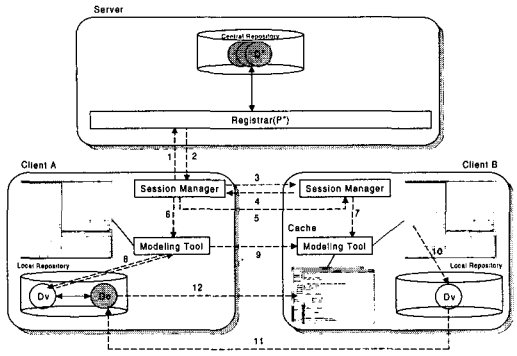


그림 6 뷰-코어 분할을 통한 협동작업

토자 역할을 담당한 사용자 B가 협동 작업을 하는 과정을 나타낸 것이다. 서버에 위치한 등록 관리자(registrar)는 협동 작업을 위한 공통의 접속점(connection point)을 제공하고 등록된 사용자 리스트를 관리한다. 각 클라이언트에 위치한 세션 관리자(session manager)는 협동 작업에 참여하는 다른 사용자들에 대한 정보를 제공하는 인터페이스 기능을 제공한다.

그림 6에서 클라이언트 A는 목표 소프트웨어 시스템을 분할한 모델 중 하나인 "Graphics Primitive" 모델의 뷰 객체와 코어 객체를 모두 가지고 있는 상태에서 다음과 같은 순서에 따라 클라이언트 B와 협동작업을 수행한다.

1. 클라이언트 A의 세션 관리자는 서버의 등록 관리자에 메시지를 보내 "Graphics Primitive Conference" 세션을 등록한다.

2. 클라이언트 A의 세션 관리자는 서버의 등록 관리자로부터 현재 서버에 등록된 사용자들의 목록과 정보를 받아오고 클라이언트의 사용자 A는 이로부터 클라이언트 B의 사용자 B의 주소를 알 수 있다.

3. 사용자 A는 클라이언트 B의 세션 관리자를 통해 사용자 B에게 "Graphics Primitive Conference" 세션에 참여를 요청하는 메시지를 보낸다.

4. 사용자 B가 참여 요청을 받아들이면 클라이언트 B의 세션 관리자는 클라이언트 A에 "Join Graphics Primitive Conference" 메시지를 보낸다.

5. 클라이언트 A의 세션 관리자는 확인(acknowledgement) 메시지를 보낸다.

6. 클라이언트 A의 세션 관리자는 모델링 도구를 호출(invoke)한다.

7. 클라이언트 B의 세션 관리자는 모델링 도구를 호출한다.

8. 사용자 A가 "Graphics Primitive" 모델의 뷰 객체를 클라이언트 정보저장소에서 로드한다.

9. 클라이언트 A의 모델링 도구는 "Graphics Primitive" 모델의 뷰 객체만을 클라이언트 B로 전송하고 클라이언트 B의 모델링 도구가 이를 디스플레이한다. 이 때부터 사용자 A와 사용자 B는 "Graphics Primitive" 모델의 클래스 다이어그램에 대해 WYSIWIS 뷰를 공유하게 된다.

10. 사용자 B가 모델링 도구의 클래스 다이어그램을 통해 특정 클래스의 코드를 보기 위한 명령어를 실행한다.

11. 사용자 B의 모델링 도구는 해당 뷰 클래스에 저장된 코어 클래스 링크를 따라 클라이언트 A에 저장되어 있는 코어 클래스에 접근한다.

12. 사용자 B는 클라이언트 A에서 전송받은 코어 클래스 내용을 이용하여 해당 클래스에 대한 프로그램 코드를 화면에서 볼 수 있다.

4. 평 가

본 논문에서 제시한 공유 데이타 분할 모델을 이용하면 다음과 같이 3.1에서 제시한 분산 객체 지향 소프트웨어 개발 환경을 위한 요구사항을 만족시킬 수 있다.

4.1 동시성을 최대화할 수 있는 유연한 잠금 단위

본 논문에서 제시한 역할은 하나의 객체 메소드, 객체 속성부터 여러 개의 객체 집합까지 포함할 수 있는 객체 지향 소프트웨어 시스템에 적합한 공유 데이타 단위로 사용자들은 이를 기반으로 동시에 각 역할에 대한 수정 작업을 할 수 있으므로 협동 작업에서 동시성을 최대화 할 수 있다.

그림 7은 역할을 이용하여 협동 작업 중 동적으로 객체 안의 메소드를 분할하여 다른 클라이언트에게 전송하는 과정을 나타낸 것으로 (a)에서 클라이언트 1은

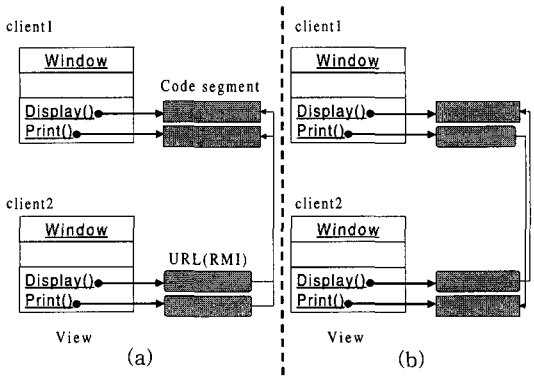


그림 7 동적 역할 분할

"Window" 객체에 대해 뷰 객체와 코어 객체를 가지고 클라이언트 2와 협동 작업을 시작한다. 클라이언트 2는 클라이언트 1로부터 "Window" 객체에 대해 뷰 객체만을 전송받아 복사본을 저장하고 협동 작업 세션을 시작한다. 이 때 클라이언트 2의 뷰 객체 안의 "Display()", "Print()" 메소드는 클라이언트 1에 저장되어 있는 해당 메소드의 코드 세그먼트에 대한 URL만을 가지며 실제 코드 세그먼트는 클라이언트 1에만 저장되어 있다.

협동 작업의 초기 단계인 (a)에서는 클라이언트 1의 사용자는 객체의 소유자로 객체에 대한 수정을 담당하고, 클라이언트 2의 사용자는 검토자로 "Window" 객체에 대해 URL을 통해 접근하여 내용에 대한 보기 연산(view operation)만을 수행한다. 협동 작업 중간에 세션 역할에 변화가 생겨 클라이언트 2의 사용자가 "Print()" 메소드에 대한 수정 작업을 맡게 되었을 경우, 클라이언트 1은 "Window" 객체의 뷰 객체 중 "Print()" 메소드를 선택하여 클라이언트 2로 소유권을 넘길 수 있다. 이 때, 소유권을 넘겨준 클라이언트 1은 해당 메소드에 대해 코드 세그먼트에 대한 포인터를 클라이언트 2에 있는 코드 세그먼트에 대한 URL로 갱신한다.

이와 같이 객체 내의 메소드나 속성을 협동 작업 중 분할하여 소유권을 넘겨주는 경우는 생명 주기의 후반부인 시스템 통합 단계에서 주로 발생한다. 인쇄, 네트워크 인터페이스, 데이타 베이스 인터페이스 등 외부 서브시스템과의 연결을 담당하는 메소드들은 해당 객체의 소유자보다는 외부 서브시스템의 설계자가 소유권을 넘겨받아 해당 메소드를 구현하는 것이 효율적인 경우가 많다. 즉, 외부 서브시스템과의 인터페이스를 담당하는 클래스의 경우에는 클래스의 소유권을 더 작은 단위로 나누어 여러 사람이 나누어 가지는 것이 효율적인 경우가 많다. 하지만 이러한 경우에도 해당 메소드에 대한 수정 작업이 끝나면 소유권을 다시 돌려 받아 하나의 객체는 한 명의 소유자에 소유권을 주도록 하는 것이 일관성을 유지하기 쉽다.

뷰-코어 분할과 역할 기반 분할을 통해 얻을 수 있는 또 하나의 장점은 (b)와 같은 상태에서 프로그램 코드를 얻기 위해 GetCode() 메소드를 호출하면 어떤 클라이언트에서도 항상 같은 결과를 얻을 수 있다는 것이다. 다중 버전을 사용하는 경우에는 병합 작업이 이루어지기 전에는 각 클라이언트마다 서로 다른 프로그램 코드가 생성된다.

본 논문에서는 제시한 공유 데이타 분할 모델의 성능을 평가하기 위하여 클래스 단위의 잠금 단위를 사용한 경우와 역할 단위의 잠금 단위를 사용한 경우에 대해 다음과 같은 조건에서 실제 네트워크에서 클라이언트들의 평균 응답 시간을 측정하였다.

- 각 클라이언트들은 Java 언어로 구현되었으며 네트워크를 통한 공유 데이타 접근을 위하여 RMI(Remote Method Invocation) 기법[11]을 사용하였다.
- 각 클라이언트들은 반경 4Km 이내의, 같은 캠퍼스 안의 LAN에 연결된 워크스테이션(SGI Indigo2 3대, IBM R40 1대)에서 실행하였다.
- 뷰-코어 분할 후 잠금 단위를 클래스 단위로 하는 경우와 잠금 단위를 역할 단위로 하는 경우에 대해 클라이언트 수를 2, 3, 4로 증가시키면서 동일한 시나리오의 사용자 입력에 대해 100번의 응답 시간을 측정하여 그 평균값을 계산하였다. (그림 5의 클래스 다이어그램 및 해당 소스 코드를 이용하여 사용자 수가 2, 3, 4 인 경우에 대해 각 사용자의 입력 이벤트를 순서대로 추출한 후, 별도의 프로그램을 통해 입력 이벤트 간격을 랜덤하게 조정하여 순서대로 이벤트를 발생하도록 하였다.)

표 1과 같이 잠금 단위가 클래스 단위인 경우와 비교하여 역할 단위의 잠금 단위를 사용한 경우, 12% ~ 18% 정도 평균 응답 시간이 감소하였고, 클라이언트의 개수가 증가함에 따라 응답 시간이 완만한 기울기로 증가하였다. 특히, 역할 단위의 잠금을 사용한 경우에는 클라이언트 개수가 3인 경우와 4인 경우에 거의 같은 평균 응답 시간을 보여 클라이언트 개수의 증가에 따라 응답 시간이 늦어지지 않는, 확장성(scalability)이 뛰어난 모델임을 알 수 있다.

표 1 클라이언트 평균 응답시간 비교표(단위: ms)

방법 \ 클라이언트	2	3	4
class locking	91.9	145.3	147.9
role locking	75.4	127.3	127.9

4.2 뒤늦은 참가자 지원 기능과 진행 중인 세션 저장

뒤늦은 참가자 지원 기능은 뷰-코어 분할에 의해 제공된다. 그림 7의 (b)와 같은 상태에서 새로운 사용자가 클라이언트 3에서 세션에 참가하면 클라이언트 1은 자신의 뷰 객체만을 클라이언트 3에 전송한다. 클라이언트 3의 뒤늦은 참가자는 전송받은 뷰 객체의 코어 객체에 대한 URL을 통해 현재 진행 중인 세션의 상태를 그대로 이어받아 세션에 참가할 수 있다.

세션이 진행되는 도중에 참가자들이 세션의 상태를 보존하고 나중에 재개하기로 합의했을 경우 각 클라이언트는 뷰 객체들과 자신이 소유하고 있는 코어 객체들만을 저장하면 나중에 세션을 재개할 수 있다.

4.3 클래스 다이어그램에 대한 빠른 응답 시간 보장

본 논문의 데이타 분할 모델에서는 클래스 다이어그램의 공유를 위하여 각 클라이언트마다 뷰 객체의 복사본을 유지하여 사용자 인터페이스를 제공하므로 코어 객체에 대한 접근이 없을 경우, 빠른 응답 시간이 보장된다. 또한 분산 소프트웨어 개발 환경에서 대표적인 협동 작업의 형태인 소유자-검토자 형태의 세션에서는 그림 1의 MVC 모델과는 달리 소유자가 속한 클라이언트에 코어 객체가 대부분 저장되므로 코어 객체에 대한 수정을 주로 담당하는 소유자의 클라이언트에서는 네트워크 지연 시간이 없어 빠른 응답 시간 내에 작업을 수행할 수 있다.

5. 결 론

본 논문에서는 분산 객체 지향 소프트웨어 개발 환경에서의 협동 작업 중 동시성을 향상시킬 수 있는 공유 데이타 분할 모델과 이를 이용한 실험결과를 제시하고, 이전의 개발 경험에서 도출된 요구사항들을 만족시킴을 보였다.

공유 데이타 분할 모델은 역할 기반 분할을 통해 목표 소프트웨어 시스템의 객체들을 서로 독립적인 부분으로 분리한 후, 뷰-코어 분할을 통해 뷰 객체만을 각 클라이언트에 복사(replicate)하여 빠른 응답 시간을 보장하도록 하였다. 코어 객체는 하나의 클라이언트에만 저장한 한 후, 역할 단위의 잠금 기법을 이용하여 불일치 문제가 발생하지 않도록 하였다.

뷰-코어 분할은 실시간 협동 작업 시 불일치 문제가 발생해도 큰 문제가 되지 않고 빠른 응답 시간을 요구하는 뷰 객체들은 분산 복사 구조를, 불일치 문제가 발생하면 해결이 곤란한 코어 객체들은 중앙 집중 구조를 채택하여 분산 객체 지향 소프트웨어 개발 환경에 적합한 협동 작업 런타임 구조를 제공한다.

참 고 문 헌

[1] J. Rumbaugh. , et al., *Object-Oriented Modeling and Design*, Prentice-hall, 1991.

[2] 김태훈, 신우창, 김태균, 신영길, 우치수, "분산 객체 지향 소프트웨어 개발 환경의 설계 및 구현", 한국 정보과학회 논문지 (C), 제 3권 2호, 1997년 4월, pp. 139-151.

[3] S. Greenberg and M. Roseman, "Groupware Toolkits for Synchronous Work," *Computer Supported Cooperative Work, Trends in Software Series*, John Wiley & Sons.

[4] S. Greenberg and D. Marwood, "Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface," *Proceedings of the ACM CSCW'94 Conference on Computer Supported Cooperative Work*, pp. 207-217, 1994.

[5] T.C. Nicholas Graham, T. Urnes, R. Nejabi, "Efficient Distributed Implementation of Semi-Replicated Synchronous Groupware," *Proceedings of UIST '96*, pp. 1-10, 1996.

[6] F. pacull, A. Sandoz, and A. Schiper, "Duplex: A Distributed Collaborative Editing Eivnironment in Large Scale," *Proceedings of the ACM CSCW '94 Conference on Computer Supported Cooperative Work*, pp. 165-173, 1994.

[7] J. C. Grundy, W. B. Mugridge, J. G. Hosking and R. W. Amor, "Support for Collaborative, Integrated Software Development," *Proceedings of Software Engineering Environment '95*, pp. 84-94, 1995.

[8] J. Grundy, J. Hosking, and W. B. Mugridge, "Inconsistency Management for Multiple-View Software Development Environments," *IEEE Transactions on Software Engineering, Vol. 24, No. 11*, November 1998.

[9] M. Stefik, D. G. Bobrow, G. Foster, S. Lanning, and D. Tatar, "WYSIWIS Revised: Early Experiences with Multi-User Interfaces," *ACM Transactions on Office Information Systems, Vol. 5, No. 2*, pp. 147-167.

[10] *UML Semantics version 1.1*, Rational Software, http://www.rational.com/uml, Sep. 1997.

[11] D. Flanagan, *Java Examples In A Nutshell*, pp. 291-322, O'Reilly, 1997.

김 태 훈

1992년 2월 서울대학교 계산통계학과 학사. 1994년 8월 서울대학교 계산통계학과 전산과학전공 석사. 2000년 2월 서울대학교 전산과학과 이학박사. 2000년 3월 ~ Pumpkin Networks Inc. (Sunnyvale, 미국) 근무. 관심분야는 객체지향 방법론, CSCW, 사용자 인터페이스, WWW, 리눅스 클러스터링.

신 영 길

정보과학회논문지 : 소프트웨어 및 응용
제 27 권 제 2 호 참조