韓國經營科學會誌
第20卷 第3號
1995年 12月

그룹협동 시스템을 위한 변화관리 모형의 설계

허순영*

Design of A Change Management Framework for Group Collaborative Systems

Soon-Young Huh*

Abstract

Group collaborative systems are recently emerging to support a group of users engaged in common tasks such as group decision making, engineering design, group scheduling, or collaborative writing. This paper provides an change management framework for the group collaborative systems to facilitate managing dependency relationship between shared objects and dependent user views, and coordinating change and propagation activities between the two in distributed computing environments. Specifically, the framework adopts an object-oriented database paradigm and presents several object constructs capturing dependency management and change notification mechanisms. First, it introduces change management mechanisms with transient shared objects and secondly, it extends them into persistent object computing environment by integrating transaction management mechanisms and change notification mechanisms. A prototype change management framework is developed on a commercial object-oriented database management system.

* 한국과학기술원 산업경영학과 교수
본 연구는 한국과학재단의 연구비 지원으로 이루어졌음.

1. 서 론

최근들어 문서 공동 작성, 그룹의사결정, 그룹스케줄링과 같이 하나의 공동 작업에 여러사람이 동시에 일하는 그룹 업무들이 생산성 제고를 위하여 그룹 협동 시스템 또는 그룹웨어 시스템(Groupware System)으로 불리워지는 소프트웨어 시스템들에 관한 연구가 활발하게 진행되고 있다. 이러한 협동 시스템하에서는 여러명의 사용자들이 하나의 객체를 공유하고 동시에 고칠 수 있으므로, 공유 객체의 변화결과가 동시에 작업하고 있는 여러명의 사용자들에게 적절히 알려져야 할 필요가 있다. 변화관리 기법이란 공유된 객체가 수정되었을 때, 발생된 변화를 관련된 다른 사용자에게 제때에 알리게 하는 기법으로 그룹 협동 시스템의 부상과 더불어 그 중요성이 점점 부각되고 있다.

구조적으로 볼때 그룹 협동 시스템은 대화식 사용자 인터페이스, 분산 컴퓨팅, 실시간 시스템, 객체저장 기법등 여러 기술분야의 혼합이 요구된다. 그러나 개별적인 기술의 발달에 비교해 볼때 그룹 협동 시스템 관점에서 변화관리를 지원해 주기 위한 개별기술의 통합은 아직 초보적 단계에 머물고 있다. 예를 들어, Smalltalk 환경의 표준 사용자 인터페이스로 제공되는 Model-View-Controller(MVC) 체계 [9, 15, 22]의 경우 일시적인 후원자에 관련한 종속관계는 지원하나 지속적인 후원자나 여러명이 동시에 사용할 수 있는 분산시스템으로까지는 아직 발전되지 못하고 있다. 한편, 객체에 기반을 두고 대규모의 복잡한 응용시스템을 구현하기 위하여 개발된 Argus [19], Emeralid [13], GUIDE [16] 같은 기존의 분산 프로그래밍 체제 [2]의 경우, 지속적 객체를 저장할 수 있으나, 다중 사용자 지원등에 한계를 가지고 있으며, 변화관리에 필요한 종속관계 유지와 변화 통보 기능은 전무한 상태이다. 그외에 복잡한 객체를 다루고 데이타 공유기능을 제공해 주기 위하여 기존의 그룹 협동 시스템 연구 [5, 6]에서는 개별적인 객체 화일 시스템(proprietary persistent object file system)을 내부적으로 사용하는데, 다중 사용자의 데이타 공유를 위해 화일 시스템, 지속성 유지, 병행성 제어(Concurrency Control), 금지(locking), 트랜잭션 관리 기능등과 같은 상용 데이타베이스 시스템에서의 기능들을 추가하고, 그 응용 시스템의 부속물로서 변화통보(change notification)기법을 개발하였다. 그러나 응용시스템에 데이타 공유 기능을 추가한 결과 다음과 같은 약점들도 나타나게 되었다. (1) 데이타 공유 기능까지 직접 개발해야 하는 부담으로 개발자들이 응용 시스템 개발에 전념하지 못할 수 있다. (2) 데이타 공유 프로그램모듈은 대체로 범용이 아니므로 융통성 결여되기 쉽고 또 시스템 안정성이 떨어진다. (3) 데이타 공유 프로그램모듈은 제대로 구성되지 않으면 응용시스템에 비효율성과 복잡성을 야기시킨다. (4) 이러한 접근방법은 변화관리와 관련된 시스템구조를 일반적인 아닌 특정방식으로 제한함으로서, 전체 메카니즘을 이해하기 어렵게 하고, 하부시스템간의 모듈화도 어렵게 하는 단점이 있다. 이처럼 변화관리 관점에서 개별기술의 통합은 아직 초보 단계에 머무르고 있는데 그 주된 요인중의 하나는 각 기술의 모델링 패러다임들이 서로 잘 맞지 않기 때문으로 여겨진다. 바람직한 통합용 패러다임은 개별적 기술들을 가능한한 하나의 개념틀안에서 포용할 수 있어야 하며, 변화관리기법을 이해하기 쉽고 모듈화가 용이하도록 모형화하며, 일반적

이고 유연한 모델링 체계를 가져서, 협동컴퓨팅 시스템이 보다 쉽게 개발되고 다양한 응용시스템으로 확장될 수 있도록 지원해야 한다.

이 논문에서는 협동컴퓨팅 구조에서의 변화관리를 위한 객체지향모델을 제시한다. 모델의 기반으로서 특히 객체지향 데이타베이스 시스템을 채택하였는데, 이것은 객체지향 데이타베이스 시스템이 일시적인 후원자와 지속적인 후원자를 하나의 형식론(formalism)안에서 자연스럽게 모형화할 수 있게 해주며 또한 일반적으로 객체 지향 모델이 분산시스템이나 구조적으로 복잡한 시스템에 효과적으로 사용될 수 있기 때문이다[1, 2, 21]. 이 논문은 첫째로 사용자 뷰와 후원자간에 발생하는 기초적 종속관계를 서술하고 종속관계의 구축과 이들 둘 종속 쌍을 위한 변화관리기법을 기술하며, 둘째로, 이 방법을 지속적 후원자의 경우로 확장하여 변화관리기법이 모다 일반적으로 다양하게 쓰일 수 있도록 한다. 변화관리기법의 구성체들의 주요 기능들은 단순화된 C++ 클래스를 통하여 설명하였으며 상업용 객체지향 데이타베이스 시스템을 이용하여 원형(prototype) 시스템을 개발하였다. 제안되는 변화관리기법은 그 구성 요소가 되는 제반 객체들에 내부적으로 정보은닉(information encapsulation)되어 있으므로 개념적으로 보다 간결하고 이해하기 쉽게되었으며, 부산 시스템으로의 확장이 가능하고 상속(inheritance)기법을 이용하여 다양한 그룹 협동 여러 컴퓨팅 환경에서 사용될수 있다고 여겨진다. 이 논문은 다음과 같이 구성된다. 2절에서는 종속관계를 구성하는 주요 구성체의 기본 개념과 의미를 정의한다. 3절에서는 종속관계와 변화통보 기법과 관련된 구성체를 구현하는 일반적인 클래스를 정의를 제시함으로써 변화관리 기법의 설계 내용을 정리한다. 4절에서는 3절의 변화관리 기법을 지속적인 후원자 관리를 위한 기법으로 확장한다. 특히 4절에서는 정보의 공유를 위한 데이타베이스 트랜잭션 관리 메카니즘을 도입하여, 변화관리 기법을 다중 사용자 환경하에서도 활용되도록 하였다. 마지막으로 5절에서는 이 논문을 요약하고 향후 연구분야를 소개한다.

2. 변화관리의 기본적 개념과 구성체

이 절에서는 엔지니어링 디자인의 예를 통해 변화관리의 기본적 개념과 구성체에 관해 설명하고자 한다.

그림1의 a)에 있는 자전거 설계도는 한회사의 설계, 생산, 마케팅 부서의 담당자들이 각 부서에서 동시에 참조하고 수정하는 대상이다. 이와 같은 협동 컴퓨팅 환경은 최근 대두되고 있는 동시공학(Concurrent Engineering) 환경하에서 많이 볼 수 있다. 이런 그룹 협동 컴퓨팅 환경하에서, CAD 설계 부서에서는 이 도면을 구성하는 트리 구조의 객체를 그림2의 b)와 같이 자전거의 모양을 직접 보여주는 뷰를 통해 설계 작업을 할 것이다. 이러한 자전거 객체에 대해, 자재 구성(Bill of Material) 내역에 관심이 있는 생산 담당 부서에서는 그림1의 c)와 같은 리스트의 형태의 화면을 만들어 참조하게 된다. 또한 마케팅 부서에서 역시 그들의 필요에 맞게, 자전거의 설계에 다른 원가 구조를 정확히 파악하기 위해 그림 1의 d)와 같은 계층적 인적구조를 보여주는 뷰를 참조하여 각 모델마다 그에 따른 원가를 바로 계산하게 되다. 이처럼 하나의 복잡한 객체를 여러가지 다른 뷰로 나타내 줄때 가장 기본적으로 요구되는 기능 하나는 공유 객체와 이를 참조하는 뷰간의 상호 일관성이다. 즉, 사용자 뷰들이 공유객

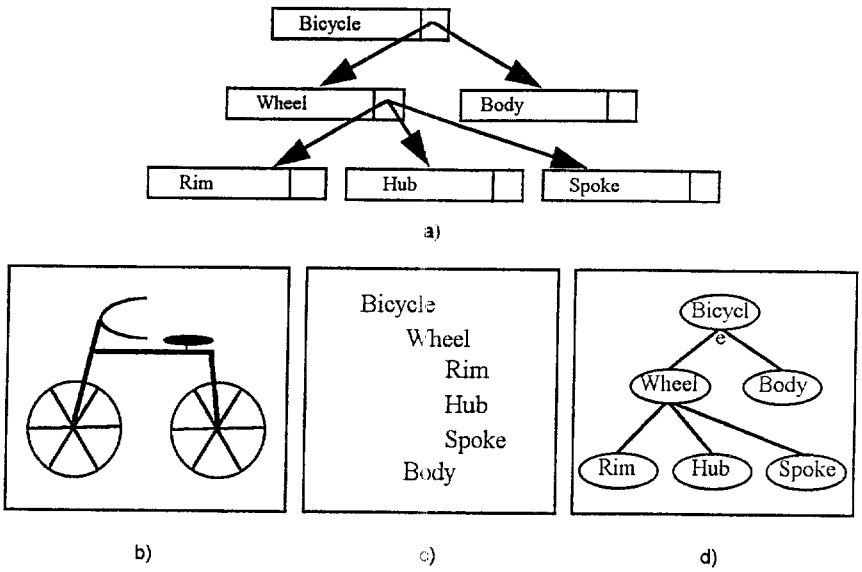

그림 1. 공유객체와 여러가지 사용자 뷰

체를 항시 반영하고 있으려면, 공유 객체가 어떤 한 뷰를 통하여 수정되었을때 다른 뷰들은 이러한 변화를 자동으로 통보받아 수정되어야 한다. 이러한 유기적 관계를 유지하기 위해서는 종속관계와 변화통보로 구성되는 두가지 형태의 연결관계가 필요하게 된다. 먼저 종속관계는 사용자 뷰들이 공유 객체에 종속적이며 이 공유 객체가 있을 때에만 그 존재 의미가 있음을 나타낸다. 따라서, 공유 객체가 삭제되면 그 종속 뷰들도 따라서 소멸되어야 한다. 다음으로 변화통보는 사용자 뷰들이 공유 객체의 상태변화를 계속적으로 주시함으로써 공유 객체의 현재 상(current image)을 반영해야 한다. 반대로 공유 객체는 그것에 종속된 뷰들에게 항상 자신의 변화를 통보해 주어야하는 의무가 있다. 그림 1의 에에 나타난 개념은 일반적으로 다음과 같이 정의될수 있다.

○ 후원자(supporter)는 가장 중요한 자료원으로서 참고되고 수정되는 공유 객체이다. 이것은 데이타베이스 서버에 저장된 지속적인 객체일수도 있고 메모리에 저장된 일시적인 객체일수도 있다. 그림1의 예에서는 a)에 있는 자전거 구조를 나타낸 트리가 후원자가 된다.

○ 종속자(dependent)는 후원자의 구조를 사용자의 관점이나 필요에 맞게 표현하는 객체이다. 종속자는 운영체제에서 하나의 프로그램으로 실행되어지는 프로세스안에서 만들어진다. 그림1의 예에서 b), c), d)의 세가지 뷰가 종속자가 된다. 분산 컴퓨팅 환경하에서 한 후원자의 종속자들은 현재의 시스템에서 생성된 프로세스안에서 내부적으로 만들어지거나 아니면 분산된 시스템들의외부 프로세스안에서 만들어 질 수 있다.

○ 종속객체(depending object)는 종속자를 일반화한 구성체로서 종속자 또는 종속자가 포

함되어 있는 프로세스를 한꺼번에 지칭한다. 예를 들어 그림1의 b)가 A라는 프로세스에서 만들어 지고 c)와 d)가 B라는 다른 프로세스에서 만들어졌다고 할때 A와 B는 트리의 종속객체가 된다. 또한 이 경우에 한 후원자의 모든 종속객체들을 종속자집합(dependent set)이라고 부른다.

○ 종속쌍(dependency pair)은 하나의 후원자와 그 종속자 집합의 쌍을 지칭하며, 종속관계(dependency relationship)는 모든 종속쌍의 집합을 나타낸다.

일반적으로 종속관계에서 하나의 후원자가 여러개의 종속자를 가질수 있는 것처럼 하나의 종속자나 종속객체도 여러개의 후원자를 참조할수 있다. 또한 분산 컴퓨팅환경하에서 종속객체는 한개의 시스템에 모여 있을 수도 있고 여러개의 시스템에 분산되어 있을 수도 있다. 이와같은 그룹 컴퓨팅환경하에서 후원자는 서버시스템의 프로세스에 의해 관장되어지며, 종속자들은 다른 클라이언트 시스템의 여러 프로세스들에 분산되어 독립적으로 후원자와 상호작용한다. 객체 수명관점에서 볼때, 후원자는 지속적일 수도 있고 일시적일 수도 있는 반면 종속자는 모두 일시적이다. 다시말해서 후원자는 그것을 만들어낸 프로세스가 끝나도 살아남을 수 있는 반면에 종속자는 프로세스가 끝나면 이와함께 사라지게 된다.

변화통보를 위하여, 후원자는 그 자신과 자신의 종속자에게 변화를 통보해 주는 기능을 가지고 있어야 한다. 변화통보는 일반적으로 변화를 초래한 사용자의 종속 객체에 의하여 수정이 끝남과 더불어 시작되게 할 수도 있으며, 이와 달리 후원자가 변화 내용을 축적하였다가 어떤 기준에 의하여 한꺼번에 통보를 하게 할 수도 있다. 어느 경우이든, 통보를 받은 종속자들은 사전에 약속된 프로토콜에 따라 적절히 수정되어져야 한다.

다음 절에서는 이러한 변화관리 기법을 좀더 자세히 논의하다. 특별히 데이타 모델의 간결성을 위해 이 논문에서는 실제보다 단순화된 객체 모형으로 설명하되, Rumbaugh의 객체모형기법(OMT : Object Model Technique [21])을 이용하여 객체간의 관계도를 그렸다. 각 객체는 본 연구에서 시스템의 하부구조로 C^{++}프로그래밍 언어를 지원하는 객체지향 데이타베이스 시스템을 사용하기 때문에 C^{++}로 구현되었다.

3. 일시적 후원자에 대한 변화관리

이 절에서는 모든객체들이 한 프로세스 안에서 생성되어 메모리내에서만 사용되는 단순한 컴퓨팅 환경에 대하여 설명한다. 앞서 언급된 바와 같이, 변화관리 기법은 종속관계의 관리와 후원자와 종속자간의 변화통보로 구성된다. 먼저 종속관계의 유지를 위해서 두개의 단계화된(layered) 접근방법을 제시한다. 하위 단계에서는 일반적인 종속적 유지체계로 종속성 사전(dependency dictionary)이 제공되며, 상위 단계에서는 후원자가 이 종속성 사전을 자신의 속성중 하나로 포함(encapsulate)한다. 이렇게 함으로서, 후원자는 종속성 사전을 종속성 관리를 위한 기본도구로 사용하며 종속자가 발생할 때마다, 사전에 등록 관리한다. 이후 어떤 변화가 후원자가 발생하면 후원자는 종속성 사전에 등록된 자신의 종속자 집합을 참조함으로서 변화통보를 수행하다.

3. 1. 종속성 사전

한 후원자에 대한 종속자들은 일반적으로 사용자들이 임의로 만들고 제거하므로 생성소멸이 잦

게 일어난다. 종속성 사전은 이러한 종속관계의 잦은 변화를 일반적이고 효율적인 방법으로 처리하기 위해 만든 관리도구 객체이다. 특별히, 종속성 사전은 종속자의 타입에 제한을 주지않고 범용성을 갖도록 종속자 클래스를 추상클래스(abstract class)로 정의함으로서, 종속자들이 문안데이타 또는 멀티미디어 데이타같은 구체적인 응용프로그램의 객체뿐만아니라 수행중인 프로세스 객체등의 여러가지 객체타입을 가질수 있도록 하였다. 종속성 사전에서 후원자와 종속자 사이의 관계는 다대다(many-to-many)의 관계로 구성되어진다. 종속성 사전의 키값은 후원자(엄밀하게는 후원자의 객체식별자)가 되며, 특정 후원자에 대하여 그 종속자 집합이 바로 참조될 수 있게 하였다. 본 연구에서는 후원자와 종속자간의 종속관계의 효율성을 위하여 객체지향 데이타베이스에서 제공하는 연결 리스트 배열, 해쉬 테이블등의 방법중 해쉬테이블을 사용하였다. 그리고 종속관계를 지원하기 위해 다음과 같은 세가지 부류의 데이타 조작 기능-후원자및 종속자의 추가(add()), 제거(remove(), removeDep(), removeDepFromAllSup()), 검색 함수(getDepSet())-을 제공한다. 이러한 기능들은 종속성관리를 위한 가장 낮은 단계의 작업이 되는 데, 상위 단계의 후원자 클래스의 함수들의 내부에서 불려짐으로서, 후원자의 추가, 제거, 또는 검색 함수가 수행될 때, 이들과 함께 후원자의 종속관계를 자동으로 유지하도록 도와준다.

3. 2. 후원자와 종속자간의 종속관계 관리

종속성 사전을 기초하여, 상위단계에서는 후원자(Suppoter)와 종속자(Dependent)클래스가 정의된다(그림 2참조). 후원자 클래스는 여러 종속자를 함께 관리하기위하여 dependencies 속성을 갖는데 이것은 하위단계의 종속성 사전에 직접 연결된다. 즉, dependencies 속성은 후원자는 종속사전에 의해 제공되어지는 모든 종속성 유지함수와 종속자들을 포함하게 있다. 특별히 이 종속사전은 후원자가 속해 있는 프로세스내에 존재하는 종속자를 관리하는 데 목적을 두기때문에 내부종속자 사전(internal dependent ditionary)이라고 부른다. 후원자와 종속자 사이의 참조적 무결성(referential intigrity)을 보다 효과적으로 관리하기 위하여 몇가지 기초적인 기능들이 제공된다. 첫째로 후원자 클래스에서 addDep()와 removeDep()는 종속자들이 생성 또는 삭제시마다 후위자의 종속자 집합에 종속자를 등록 또는 삭제하도록 한다. 한 종속자가 여러개의 후원자를 갖는 경우에는 addDep()를 각 후원자에 대하여 한번씩 수행함으로서 종속자가 동시에 여러 후원자에 연결되도록 한다. create()와 remove()는 생성자와 소멸자로서 후원자의 생성및 소멸을 수행한다. 만일 소멸되는 후원자에게 종속자가 있다면 그 종속쌍의 참조적 무결성을 유지하기 위해서 종속자들도 함께 제거되도록 하였다. 후원자가 데이타베이스에 저장되는 지속적인 후원자에 대해서는 4절에서 설명하기로 한다. 둘째로 종속자 클래스에서 remove()는 하나의 종속자를 지우고 그것의 후원자로부터 종속관계를 없애는 역할을 한다. 내부적으로 remove()는 먼저 그 종속자가 의존하는 모든 후원자들을 찾아내고 각 후원자에서 부터 removeDep()를 실행한다. 이와같이 하나의 후원자에 대하여 종속자가 생성되고 제거될 때마다 종속관계를 유지하는 함수들이 가동되어 종속성 사전안에서 종속쌍의 참조적 무결성이 항상

유지되도록 하였다.

종속자와 후원자의 클래스 정의는 앞서 본 바와 같이 종속관계와 변화통보에 관한 최소한의 일반적인 기능을 제공하는데 중점을 두었다. 이렇게 한 것은, 종속자와 후원자 클래스가 다른 응용프로그램에 베이스 클래스로서 쉽게 응용되게 하기 위함이다.즉, 이 후원자와 종속자들은 응용프로그램의 파생 후원자와 파생 종속자 클래스로 상속(inheritance)되어서 파생클래스로 하여금 종속성 관리기능을 그대로 전수 받게하고 그위에 응용프로그램에 필요한 속성과 함수들을 더할 수 있게 한다. 이러한 상속체제는 결과적으로 베이스 클래스와 파생 클래스 사이에서 변화관리 작업과 응용시스템의 기능구현을 효과적으로 나누게 하는 장점이 있다. 곧, 베이스 클래스는 종속 관계 및 변화통보의 공통적이고 핵심적인 작업을 맡으며 파생 클래스들은 각 응용 시스템의 작업들을 맡는 것이다. 이렇게 작업을 분할시키면 복잡한 상속계층구조(inheritance hierarchy)가 형성되어도 파생 클래스 층이 단순해지고 모듈응집도가 높아지게 된다. 일단 후원자 클래스가 파생 클래스로 상속되어지면 변화 관리기법은 파생 클래스에게는 은닉되기 때문에 복잡한 관리 내역을 일일이 기억하지 않더라도 자동적으로 수행이 되어진다. 즉, 파생클래스에서의 생성되는 모든 종속자들은 종속성 사전에 항시 등록되며 은밀히 베이스 클래스의 변화관리기법에 의해 관리되게 된다.

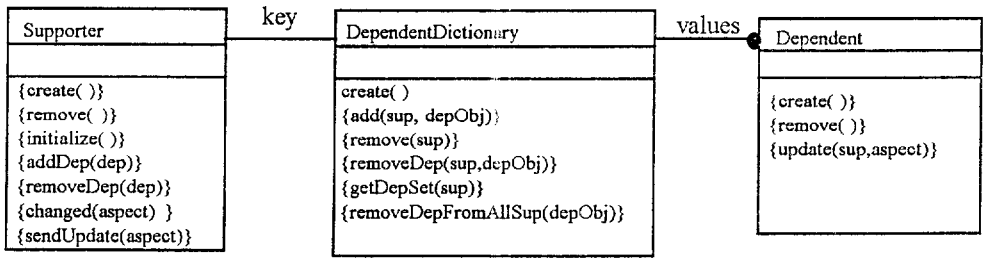

그림 2. 종속 관계를 위한 객체 정의

3. 2. 변화 통보

후원자가 수정되었을때 후원자는 변화통보기법을 사용하여 변화된 사항을 종속자들에게 알려야 한다. 변화통보기법은 후원자가 수정되어 그 상태가 변화가 되면, 후원자 클래스의 changed()를 실행시킴으로서 시작된다. 그림 3은 변화통보과정을 사건 발생 순서로 설명한 추적도(Event Trace Diagram)이다. 그림에서 세로로 그어진 선들은 변화통보 메카니즘에 간여하는 객체들이며, 가로로 그어진 선들은 시간별로 발생하는 사건들이다. 일반적으로 change()는 후원자를 수정한 프로세스가 수정직후 직접 실행할 수도 있으며, 후원자 자신이 스스로 변할때마다 자동적으로 실행하게할 수도 있는데, 각각의 경우 처리에 따르는 비용이 다르므로, 효율성을 고려하여 응용프로그램마다 다른 방식을 택할 수 있다. 후원자가 일시적 객체인 경우는 일단 change()가 실행되기 시작하면, sendUpdate()가 change()안에서 트리거(trigger)

된다. sendUpdate()가 실행되면, 종속성 사전은 현 후원자의 종속자 집합을 모두 검색하여 후원자에게 알려준다. 후원자는 이들 각각에게 자신의 변화내용과 그에 맞는 수정 요청 메시지를 보내게 된다.(지속적인 후원자의 경우 change()의 수행내역은 이와 다른데 다음절에서 다룬다.) 종속자는 수정 요청을 받자마자 update()를 실행하여 후원자가 변화상태를 반영한다. 필요한 경우에는 후원자가 change()와 update()의 인수를 사용하여 종속자가 어떻게 고쳐져야 하는지를 보다 구체적으로 요청할 수 있다.

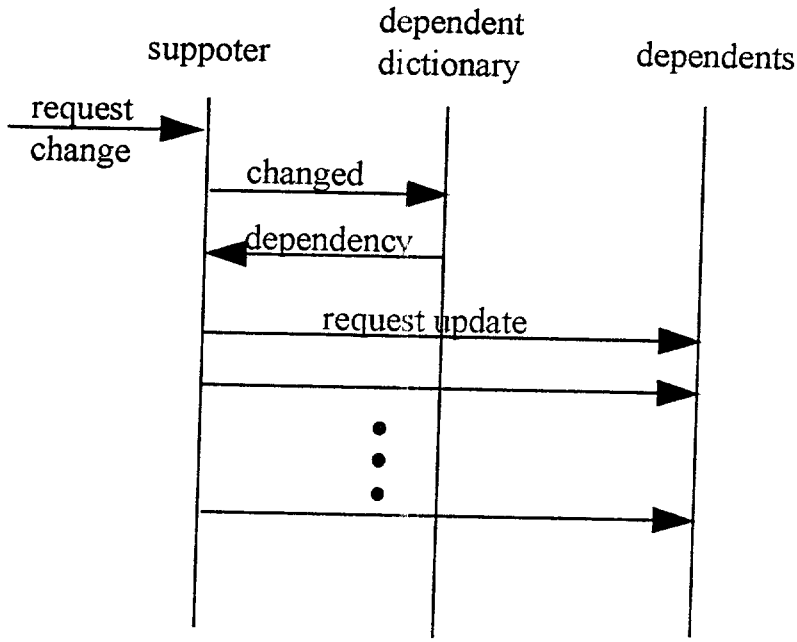

그림 3. 변화 통보를 위한 추적도(Eventj Trace Diagram)

앞에서 말한바와 같이 변화통보기법의 장점은 후원자의 파생 클래스에서 나타나게 된다. 즉, 파생 후원자가 변화되었을 경우, 후원자는 변화를 통보받아야 할 파생 종속자를 일일이 기억할 필요없이 메시지 방송 프로토콜(message broadcasting protocol) 방식으로, 자신이 변화된 것을 방송하기만 하면 베이스 후원자 클래스에 구현된 변화통보기능이 저절로 수행된다.

종속관계 유지기법이 후원자와 종속자의 클래스에서 최소한의 일반적 기능만을 갖도록 정의되었듯이, 변화통보기법 또한 간결하게 정의되었다. 한가지 유의할 점은 후원자의 클래스의 경우 변화가 발생했을 때 변화통보 프로토콜을 수행하는 change()를 가지고 있는 반면

후원자를 직접 수정하는 함수는 없다. 마찬가지로 종속자 클래스의 update() 함수도 가상 함수로 정의되어서 구체적 수정방식은 상위단계로 위임하고 있다. 이것은 후원자를 수정하는 함수와 종속자에서 변화를 통보바다 수정하는 함수가 모드 응용시스템에 의하여 결정되기 때문이며 따라서, 베이스 클래스에서는 불필요하기 때문이다. 현재의 후원자와 종속자는 응용 시스템의 파생 클래스로 상속되며 후원자의 수정함수나 종속자의 수정함수는 개별 파생 클래스에서 구체화되어 진다. 대부분의 경우 change()함수의 실행은 파생 후원자 클래스에서 수정함수의 내부에서 작동될 것이다. 예를 들면, 변화를 일으키는 함수의 마지막 문장으로 change() 함수를 포함시킴으로써 수정함수에 의해 변화가 일어났을때 변화통보기법이 자동적으로 즉시 실행되게 할 수 있다.

변화관리기법을 그림1의 자전거 공학 설계의 예에 적용하여 보자. 트리구조(그림1의 a))와 나머지 세개의 뷰들(그림 1의 b)~d))은 종속관계의 후원자와 종속자들이므로 Supporter와 Dependent의 파생 클래스의 인스턴스(instance)들로 등록이 된다. 파생 후원자로서 트리구조는 후원자에 없는 기능들, 즉, 설계도의 일부분을 추가하거나 삭제하고 한 곳에서 다른 곳으로 이동시키는 등의 수정변화함수들을 갖게될 것이다. 반대로 뷰들은 트리구조에서 이루어지는 변화에 반응하여 각각 별도의 수정작업을 수행하기 위해서 Dependent 클래스마다 구체적인 update()함수를 가질 것이다. 이런 상황하에서 둘 사이의 변화통보과정은 간단하다. 예를 들어 자전거 트리구조의 앞쪽 휠이 다른 형태로 바뀌었을 경우에 트리구조는 즉시 change()를 트리거하며 변화된 내용을 인수에 반영하여 자신이 변화되었음을 방송한다. 이에 따라 세개의 종속된 뷰들은 변화를 통보받아 개별적으로 update()를 실행하여 각각 용도에 맞게 수정한다. 이처럼 제안된 변화관리기법은 별도의 코딩수고없이 여러 응용시스템에 적용될수 있다.

4. 지속적 후원자(persistent object)의 변화관리(change management)

앞절에서는 후원자와 종속자가 모두 일시적인 객체로 메모리에 저장되어 있다는 전제하에 논의를 하여왔다. 이절에서는 지속적인 후원자가 서버 시스템의 데이타베이스 시스템에 저장되어 있고 종속자는 클라이언트 시스템의 프로세스에 존재하는 컴퓨팅환경하에서 확장된 변화관리기법에 대하여 논의한다.

4. 1. 트랜잭션 관리과 변화관리기법

지속적(persistent)후원자란 후원자가 데이타베이스 시스템에 저장되어서 프로그램 세션이 끝나도 계속 그 내용이 보존되는 경우를 말한다. 일반적으로 후원자의 지속성은 화일시스템에서 단순히 후원자의 snopshot를 제공하는 기능만으로도 지원이 된다할 수 있으나, 여기서는 다중 사용자의 동시 병행접근까지 허용해야한다는 점에서 이보다는 좀더 복잡한 기능을 전제한다. 즉, 지속적 후원자를 지원하려면 첫째 다중 사용자에 의해 후원자가 공유될 수 있어야 하며, 변화관리기법이 지원되어야 한다. 이것은 곧, 후원자가 데이타베이스 시스템에 저장되어 있더라도, 변화관리메카니즘이 후원자의 상태변화를 계속 살펴서, 변화발생시 이 변화를 바로 종속자에게 알려주어야 함을 의미한다. 그러나 지속적 후원자의 경우 변화관리메카니즘의 세부적인 내용은 일시적 후원자인 경

우와 다르게 된다. 즉, 메모리에 머무르는 일시적 후워자인 경우 단일 사용자를 전제하므로 변화가 발생하면 현재 변화된 후원자의 상태(valid state)를 기초로, 즉시 종속자에 대한 변화통보를 하게된다. 그러나, 지속적 후원자으 이 경우는 이와 같은 즉시적 변화통보는 바람직하지 않다. 그 이유는 다중 사용자가 데이타베이스 시스템에 저장된 후원자를 동시에 병행 접근하여 수정하다보면, 그 사용자들의 작업간에 상호 충돌이 발생하여, 짧은 순간이나마 후원자 상태를 오류의 상태로 만들수 있기 때문이다. 이러한 오류발생시는 후원자가 변화된 것 같아도 그 변화 상태가 유효한 것으로 판정되어질 때까지, 변화통보를 연기할 필요가 있다. 구체적인 변화통보연기의 방법은 지속적 후원자를 저장하는 객체 지향 데이타베이스 시스템이 갖는 다음과 같은 기본적인 성질과 관련지어 생각해 볼 수 있다.

- 개체 무결성(object integrity) : 데이타베이스 시스템은 지속적 후원자가 수정 전이나 후에나 항상 오류가 없는 유효한 상태를 유지하도록 지원을 해주어야 한다. 따라서, 후원자 수정작업은 변화를 완료하여 새로운 상태로 되게하거나 아니면 전혀 변화를 주지 않은, 원래의 상태 그대로를 유지하게 한다. 이때, 수정을 완료한 작업을 "완료(commit)"라하며, 중간에 어떤 오류가 발생해서 수정을 완료하지 못한 작업을 "중지(abort)"라 한다.
- 순차성(Serializability) : 한 후원자에게 동시에 수행되는 다중 수정작업은 서로 간섭하지 않고 순차적으로 진행되어서 후원자의 개체 무결성을 유지하도록 수행되어야 한다. 즉, 어떤 작업에 의해 부분적으로 수정된 후원자를 아직 완료되지 않은 시점에서 중간에 다른 작업이 수정하는 일이 허용되어서는 안된다.
- 지속성(Permanence) : 한번 수정작업이 완료되면, 그 결과는 데이타베이스에 반영되어서 원래 상태로 되돌릴 수 없다. 그 결과를 바꾸려면 새로이 다른 수정작업을 수행해야 한다.
- 트랜잭션 관리(Transaction Management) : 트랜잭션 관리를 수정작업시 전술한 성질을 보장하기 위해 데이타베이스 시스템에서 사용되는 메카니즘이다. 트랜잭션 관리는 시작점과 종료점의 경계를 가지고 있다. 트랜잭션 경계내의 모든 변화는 개별적으로 보호되며 끝날 때까지 다른 사용자는 볼 수 없다. 경계내에서 트랜잭션 관리는 logging, roll back, roll forward, locking, deadlock detection 등의 여러가지 전형적인 데이타베이스 기법을 사용한다. 지속적 객체의 수정작업은 항상 경계 내에서 수행되어야 하고 트랜잭션이 성공적인가 아닌가에 따라서 작업이 중지되거나 완료된다. 그러므로 경계내의 후원자의 상태는 트랜잭션이 끝날 때까지 단언할 수 없는 반면 일단 트랜잭션 경계 밖에서의 후원자의 상태는 항시 유효하다.

트랜잭션이 실행되는 동안 한 경계내에서 트랜잭션은 보통 여러 단계를 거친다. 우리는 네가지의 대표적인 단계들-시작(begin), 시완료(precommit), 완료(commit), 중지(abort)-에 초점을 맞추는 데 이러한 단계들이 변화통보 기법과 관련하여 활용될 수 있기 때문이다. 시작단계는 트랜잭션이 시작되는 시점이다. 트랜잭션 경계내의 일련의 수정작업이 모두 끝나면 다음의 시완료단계에 도달한다. 이 단계에서는 현재의 트랜잭션이 실행중인 다른 트랜잭션을 간섭하지 않았는지 확인하기 위해서 몇가지 병

행성 검사를 한다. 검사가 성공적이면 완료단계에 도달하고 트랜잭션은 성공적으로 끝난다. 그러나 병행성 검사에서 문제가 발생하면 중지단계로 가서 트랜잭션이 끝날 때 모든 수정작업을 무효화한다. 이상의 각 단계들은 사용자 프로시저와 함께 정의될 수 있는 데, 각 프로시저내에 변화관리 메카니즘을 삽입하여서, 후원자의 저장, 제거, 변화전파 등의 변화관리 기법이 각 단계에 효과적으로 실현되게 할 수 있다.

4. 2. 변화의 유보 지연

지속적 후원자에 대한 수정작업이 완료되면 변화통보 메카니즘은 이변화를 즉시 전파해야만 한다. 트랜잭션관리의 관점에서 보면 종속자는 트랜잭션 경계내에서 작업이 완료(commit)된 수정작업에 대해서만 변화를 통보받을 필요가 있다. 반대로, 트랜잭션이 중지된 경우는, 중간에 후원자가 부분적으로 수정되었더라도 아직 유효한 변화가 아니므로, 종속자는 알 필요가 없다. 그림 1의 예에서 한 트랜잭션 경계내에서 일련의 수정작업이 지속적 후원자를 수정하는 경우를 보자. Wheel (하위의 객체들과 함께)를 현재의 Bicycle에서 Body로 이동하는 작업은 먼저 Wheel을 Bicycle에서 떼는 일과 그 Wheel을 다시 Body에 연결하는 일로 구성되어 하나의 트랜잭션안에서 두가지의 수정작업을 요한다. 변화관리 관점에서 보면, 첫번째의 Wheel을 떼어내는 작업이 수행된 후라도, 전체적으로는 이동작업이 절반만 수행되었으므로 이 수정작업 내용을 기록하되 두번째 연결하는 작업이 완료될 때까지 변화통보를 지연하는 것이 안전하다. 그 이유는 만일 두번째 작업이 무효화되어야 하며, 이에 따라 유보지연되었던 변화도 없었던 사건으로서 소멸되어야 하기 때문이다. 이처럼 한 트랜잭션이 수정작업 도중에 중지될 수 있기 때문에 일련의 여러작업중 각 개별 작업의 수행결과는 종속자에게 즉시 전파되기보다는 하나씩 저장되어서 전체가 끝날때까지 통보를 유보하는 것이 바람직하다. 그리고 이 변화의 통보는 그 트랜잭션안의 모든 단위 작업이 끝나서 트

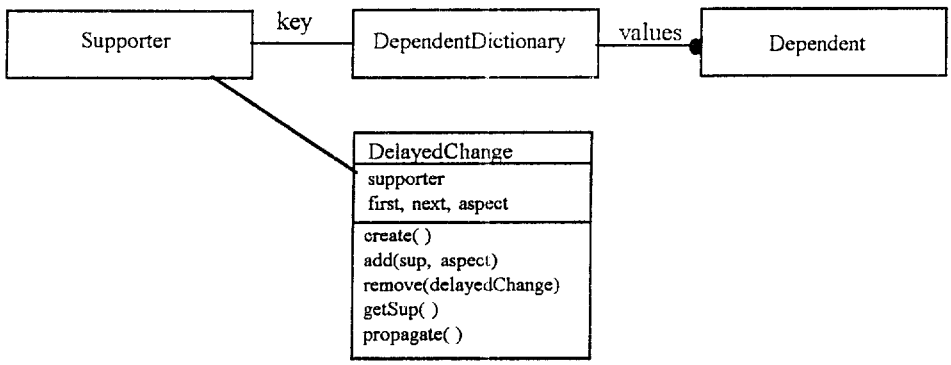

그림 4. 지연된 변화의 객체 정의

랜잭션이 완료단계에 이를 때에 시작되도록 한다.

트랜잭션 경계내에서 변화 통보를 용이하게 하고 변화의 유보 지연을 관리하기위해 지연된 변화 클래스(delayed change class)가 제공된다 (그림 3 참조). 지연된 변화는 후원자의 추상화된 전달자 객체로서 종속자에게 무엇을 어떻게 변화시킬 것인가를 전달하는 역할을 한다. 지연된 변화는 변화작업이 트랜잭션 경계내에서 수행될 때 만들어진다. 실제 수행과정을 상술하면, 후원자의 수정함수가 수행되어 changed() 함수가 작동되면, changed() 함수내에서 후원자의 지속성 여부가 먼저 검사된다. 후원자가 지속적이면 이에 대응하는 지연된 변화 객체(delayed change)가 자동적으로 만들어진다. 이때 지연된 변화 객체의 속성중에서 supporter는 변화된 후원자의 값을 가지며 aspect는 변화내용을 기록한다. 또한 지연된 변화 객체는 first, next 속성을 통해 연결 리스트로 계속 이어지게 되므로, 한 트랜잭션내에 여러개의 지연된 변화 객체가 생겨도 이를 시간적 발생순서에 의거하여 순차적으로 보관할 수 있게된다. 후에 수정 트랜잭션이 성공적으로 완료되면 하나의 객체 리스트로 연결된 유보 지연된 변화 리스트가 일시에 종속자들에게 전파된다.

지연된 변화의 통보를 보다 잘 이해하기 위해서 그림 5에 일련의 변화작업을 포함하는 트랜잭션을 예로 들었다. 이 트랜잭션은 T0 시점에서 시작되며, T1, T2, T3 시점에서 세 가지의 후원자 S1, S2, S3에 대해 변화가 수행된다. 여기서 S1, S3는 지속적 후원자로, S2는 일시적 후원자로 가정하자. 이때 이 트랜잭션은 T4 시점에서 성공적으로 완료되거나 T0와 T4사이의 어떤 중간에 중지될 것이다.

트랜잭션의 제 단계 별로 앞서 기술한 프로

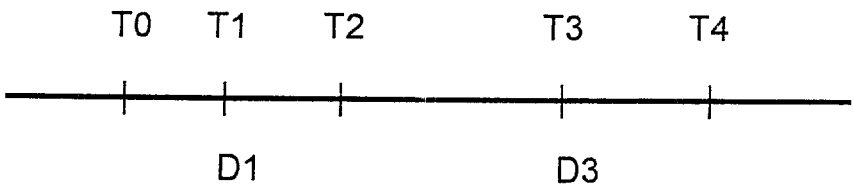

그림 5. 2개의 지속적 후원자를 포함하는 트랜잭션

시저를 준비하여 수행시킬 수 있는 데, 먼저 시작(begin) 상태인 T0를 만나게 되면 시작 프로시저가 가동되도록 한다. 시작 프로시저는 객체지향 데이타베이스 시스템으로 하여금 모든 후원자와 종속자간의 종속 관계를 읽게하며, 후원자들의 초기 상태를 기억하게 한다. 이것은 트랜잭션이 중도에 중지 상태에 이를 경우 원래 상태로 되돌릴 수 있게(roll back)하기 위함이다. 이후, T1시점에서 S1 후원자가 수정되면, 아직 트랜잭션이 완료(commit)전이므로 이 수정작업에 대한 지연된 변화, D1이 만들어진다. T2시점에서 S2에 또 다른 수정이 이뤄지면, 이때는 S2가 일시적 후원자이기 때문에 이때 생긴 변화는 지연되지 않고 바로 통보된다. T3시

점에서 S3 후원자에 세번째로 수정변화가 생기면, 또 다른 지연된 변화 D3가 D1 다음에 연결 리스트로 연결된다. 이처럼 지연된 변화를 시간적 순서로 배열하게되면 나중에 변화통보 과정에서 후원자의 객체무결성을 효과적으로 유지할 수 있게 된다. 마지막으로 T4 시점에서 완료상태에 이르면, 완료 프로시저가 가동된다. 즉, D1, D3 각각에 대해서 먼저 현재의 프로세스 내부의 종속자들에게 변화통보를 수행하고 이어서 다른 외부 프로세스들에서 S1, S3의 종속자로 생성된 외부종속자들에 대해 변화를 전파한다. 만일 이 시점에서 트랜잭션이 중지되면, 그 순간 지속적 후원자에게 가해진 모든 변화는 트랜잭션 내에서 무효가 되어 후원자는 원상태로 복귀가 되어지며, 또한 중지 프로시저가 가동되어서 현재까지 연결리스트에 누적 지연된 변화들이 모두 소멸된다.

지금까지의 설명에서처럼, 지속적 후원자의 변화관리를 위해 객체지향 데이타베이스 시스템의 트랜잭션 관리 메카니즘에 지연된 변화 개념을 결합하게되면, 복잡한 변화관리 메카니즘을 효과적으로 구현할 수 있을 뿐 아니라, 여러가지 투명성이 확보되는 장점이 있다. 첫째, 지속적 투명성(persistence transparency)이 확보되는 데, 이 것은 후원자를 수정할 때 후원자가 일시적인 것이든, 지속적인 것이든 관계없이, 또한 단일 사용자 환경이든 다중 사용자 환경에서 공유되는 후원자이든 상관없이 사용자는 항상 동일한 수정함수를 사용하나 변화관리 메카니즘은 후원자의 지속성에 따라 자동으로 수행되다. 즉, 사용자가 후원자의 상태를(일시적이든 지속적이든) 의식하지 않고 수정을 할 수 있으며, 후원자가 수정될 때, 지속적 후원자의 경우는 그 결과가 자동적으로 데이타베이스에 기록되고, changed()함수안에서 후원자의 지속성 여부에 따라 적절한 방식의 변화통보가 수행되어져서, 후원자와 종속자 모두 항상 객체무결성을 유지하게 된다. 둘째, 접근 투명성(access transparency)이 확보되는 데, 후원자가 메모리에 위치하거나 데이타베이스에 위치하거나에 관계없이 프로그램내에 있는 하나의 객체로서 바로 접근이 가능하여 생성, 수정, 제거 작업을 할 수 있다. 셋째, 병행 투명성(concurrence transparency)이 확보되는 데, 개발자는 다중 사용자들이 동시접근하는 경우를 위해 따로 별도의 프로그램을 개발할 필요가 없다. 다만 검색, 수정 작업을 수행할 때, 트랜잭션 경계안에 포함시킴으로서 병행 접근이 해결된다.

5. 결 론

그룹 협동 시스템환경하에서 공유된 객체가 계속 수정될 때, 사용자들이 공유 객체에 대한 현재의 실상을 반영하는 뷰를 갖도록 유지하는 데 있어 효과적 변화관리 모형이 매우 중요하다. 이러한 모형을 구현하기위하여 이 연구는 객체지향 데이타베이스 시스템을 기반으로 하는 객체지향 변화관리 모형을 제시했다. 변화관리 모형의 핵심적인 내용은 종속관계 유지 및 변화통보 작업으로 구성되는 데, 이 논문에서 이들은 후원자의 지속성 여부에 따라 2단계로 구분하여 제시한다. 먼저, 일시적 후원자들만을 전제하는 1단계에서는 후원자와 종속자간의 종속관계를 전적으로 기록하는 도구로서 종속성 사전이 도입된다. 종속관계를 위하여 사용자들이 한 후원자에 대한 뷰를 새로 만들면, 그 뷰는 자동적으로 후원자의 종속자로 사전에 기록되어 유지된다. 만일 어떤 다른 사용자에 의하여 그 후원자가 수정되면, 그 변화내역이 후원자의 모든 종속자들에게 통보되고, 각 종속자는 변화에 상응하는 수정작업을 후속적으로

할 수 있게한다. 2단계는 후원자가 데이타베이스 시스템에 존재하는 지속적 후원자를 전제한다. 지속적 후원자는 1단계에서 정의된 모든 기능을 소유하며, 다중 사용자가 데이타베이스 트랜잭션 메카니즘하에서 후원자를 수정하는 경우까지를 고려하며, 객체의 무결성을 유지하기 위하여 후원자의 변화내용을 일시 유보지연하는 기능을 갖도록 하였다.

이 연구에서 제안되는 객체지향 변화관리 모형은 다음의 두가지 특징으로 요약할 수 있다.

첫째, 이 연구에서는 객체지향 패러다임을 도입함으로서, 종속자 사전, 후원자, 종속자, 지연된 변화등을 포함하는 주요 구성체들이 독립적이고 모듈화된 객체로서 구현된다. 또한 객체 추상화를 통하여, 개별 객체들의 복잡한 데이타 속성과 처리 알고리즘을 숨김으로서, 복잡한 내부의 메카니즘을 다 이해하지 않고도 쉽게 변화관리 객체의 기능들을 활용할 수 있게 했다. 이 같은 변화통보기법이 내재된 객체들은 추후 건축용 블럭처럼 다른 객체에 재사용되는 것을 목표로 개발되었는 데, 곧 후원자와 종속자들을 기초로 파생 후원자를 만듦으로서 다양한 응용 시스템에 변화관리 메카니즘이 내재화(embedded)되게 된다.

둘째, 시스템의 하부 구조로서 객체지향 데이타베이스 시스템을 채용함으로써 일시적 후원자, 지속적 후원자, 및 분산된 종속자 모두를 단일 형식론하에서 수용하였다. 이 데이타베이스 시스템을 이용한 접근방법은 특히 다수 사용자 컴퓨팅 환경에서 중요한 병행성 기능(Concurrency Control)을 향상시켰으며 트랜잭션관리 기능과 변화관리 기법을 연계함으로서, 객체 무결성 측면에서 기존의 협동 컴퓨팅 구조에서 많이 사용되고 있는 파일 시스템 접근방법에서보다 높은 신뢰도를 유지할 수 있게 해 주었다.

변화관리 기법의 원형(prototype) 시스템은 SUN-4 시스템에서 객체지향 데이타베이스 시스템인 OBJECTSTORE[18] 위에 C++를 이용하여 개발되었으며, 클라이언트-서버형 분산 컴퓨팅 환경까지 지원하도록 되어있다. 현재, 분산 컴퓨팅 환경하에서의 변화 관리 기법에 대해 다른 논문이 작성되고 있으며, 향후의 연구과제로서, 그룹 협동 컴퓨팅 시스템의 응용분야로 사무 서류처리, CAD/CAM 엔지니어링 디자인, 모형관리 영역, 그룹 의사결정 지원시스템(GDSS), 또는 컴퓨터 지원 협동작업 시스템등과 같은 분야를 연구하고 있다.

참고 문헌

[1] Booch, G., *Object-Oriented Design with Applications*, CA:Benjamin/Cummings, 1990.

[2] Chin, and S. Chanson, "Distributed Object-Based Programming Systems," *ACM Computing Surveys*, vol. 23, no. 1, pp. 91-124, 1991.

[3] Coad, P. and E. Yourdon, *Object-Oriented Analysis*. Yourdon Press, NJ: Prentice-Hall, 1990.

[4] Cox, B., *Object-Oriented Programming:An Evolutionary Approach*, Reading, MA:Addison-Wesley, 1986.

[5] CSCW 88 *Proceedings of the Conference on Computer-Supported Cooperative Work* (Portland, Or., Sept. 26-29) ACM, New York, 1988.

[6] CSCW 90 *Proceedings of the Conference on Computer-Supported Cooperative Work* (LOs Angeles, CA., Oct.. 7-10) ACM, New YOrk, 1990.

[7] Ellis, C. and S. Gibbs, "Concurrency Control in Groupware Systems," *Proceedings of the International Conference on the Management of Data*, ACM SIGMOD, pp.399-407, 1989.

[8] Gorlen, K., "An Object-Oriented Class Library for C++ Programs," *Software Practice and Experience*, vol. 17, no. 12, pp. 899-922, 1987.

[9] Goldberg, A. and D. Robson, *Smalltalk-80 The Language and its Implementation*, Reading, MA:Addison-Wesley, 1983.

[10] Greif, I. and Sarin, Sunil, "Data Sharing in Group Work," *Conference on Computer-Supported Cooperative Work:A Book of Readings*, CA:Morgan Kaufmann, California, 1988.

[11] Huh, S.-Y., "An Object-Oriented Model for a Change Management Framework in Workgroup Computing Systems," submitted to Information Systems, 1995.

[12] Huh, S.-Y., and Dave Rosenberg, "A Change Management Framework: Dependency Maintenance and Change Notification," forthcoming in Journal of Systems and Software, 1995.

[13] Jul, E., H. Levy, N. Hutchinson, and A. Black, "Fine-Grained Mobility in the Emerald System," *ACM Trans. Comut. Syst.*, vol. 6, no. 1, pp.109-133, 1988.

[14] Kim, W. and F. H. Lochovsky, *Object-Oriented Concepts, Databases, and Applications*. New York:ACM, 1989.

[15] Krasner, G. and S. Pope, "Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80," *Journal of Object-Oriented Programming*, vol. 1, no. 3, pp.26-49, 1988.

[16] Krakowiak, S. M. Meysembourg, H. Nguyen Van, M. Riveill, C. Roisin, and X. Rousset de Pina, "Design and Implementation of an Object-Oriented, Strongly Typed Language for Distributed Applications," *Journal of Object-Oriented Programming*, pp.11-22, 1990.

[17] Meyer, B., *Object-Oriented Software Construction*, NJ:Prentice-Hall, 1988.

[18] Lamb, C. G. Landis, J. Orenstein, and D. Weinreb, "The ObjectStore Database System," *Commun. ACM*, vol. 34, no. 10, 50-63, 1991.

[19] Liskov, B. "Distributed Programming in Argus," *Commun. ACM*, vol. 31, no. 3, pp.300-312, 1988.

[20] Nunamaker, J., A. Dennis, H. Valacich, and D. Voge, "Information Technology for Negotiating Groups: Generating Options for Mutual Gain," *Management Science*, vol. 37, no. 10, pp.1325-1346, 1991.

[21] Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, NJ: Prentice-Hall, 1991.

[22] Shan, Y.-P., "An Event-Driven Model-View-Controller Gramework for Smalltalk," *Proceddings of Object-Oriented Pro-*

gramming Systems, Languages, and Applications:OOPSLA, October 1-6, New Orleans, Louisiana, pp.347-352, 1989.

[23] Stevens, W., *UNIX Network Programming*, NJ:Prentice-Hall, 1990.

[24] Stroustrup, B., *The C++ programming language*, Reading, MA:Addison-Wesley, 1986.

[25] Zdonik, S., and D. Maier, *Readings in Object-Oriented Database Systems*, Morgan Kaufman, 1990.