경영정보학연구
제8권 제3호
1998년 12월

분산 환경에서 그룹시스템에서의 객체 일관성 유지를 위한 체계의 개발

허 순 영*, 김 형 민*

Development of an Object Consistency Maintenance Framework for Group Systems in Distributed Computing Environments

Huh, Soon-Young, Kim, Hyung-Min

Group collaborative systems are recently emerging to support a group of users engaged in common tasks such as group decision making, engineering design, or collaborative writing. Simultaneously, as communications networks and distributed database systems become core underlying architecture of the organization, the need of collaborative systems are gaining more attentions from industry. In such collaborative systems, as the shared objects may evolve constantly or change for operational purposes, providing the users with synchronized and consistent views of the shared object and maintaining the consistency between shared object and replicated objects are important to improve the overall productivity. This paper provides an change management framework for the group collaborative systems to facilitate managing dependency relationships between shared objects and dependents, and coordinating change and propagation activities 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. And the proposed framework accommodates both persistent dependents such as replicated data and transient dependents such as various user views in a single formalism. A prototype system is developed on a commercial object-oriented database management system called OBJECTSTORE using the C++ programming language.

* 한국과학기술원 테크노경영대학원

Ⅰ. 서 론

최근 들어 그룹의사결정, 그룹 엔지니어링 디자인, 문서 공동 작성과 같이 하나의 공동 작업에 여러 사람이 동시에 일하는 그룹 업무들의 생산성 제고를 위하여 그룹 협동 시스템 또는 그룹웨어 시스템으로 불리워지는 소프트웨어 시스템들에 관한 연구가 활발하게 진행되고 있다. 또한 네트워크환경이 발달함에 따라 이러한 그룹 협동 시스템들은 사용하는 데이터베이스가 지역적으로 여러 곳에 위치하는 분산 컴퓨팅 환경으로 나아가고 있다. 이러한 분산 협동 시스템하에서 네트워크로 연결되어 있는 분산된 시스템들이 모두 자신의 데이터베이스를 가질 필요는 없지만 적어도 두 곳 이상에 데이터베이스가 위치하게 되며, 이러한 데이터베이스 안에는 중복된 데이터 (replicated data)가 존재할 수 있다. 데이터를 중복시키는 데에 따른 장점은 지역적 자율성 (local autonomy)과 데이터의 가용성 (availability) 및 신뢰성 (reliability)의 향상 그리고 통신비용의 절감 등이 있으며, 이에 따른 문제점은 데이터의 변화시 이에 대한 즉각적인 반영과 이에 대한 구현 및 관리의 복잡성에 있다.

그룹 협동 시스템하에서는 여러 명의 사용자들이 하나의 객체를 공유하고 동시에 고칠 수 있으므로, 공유 객체의 변화결과가 동시에 작업하고 있는 여러 명의 사용자들에게 적절히 알려져야 할 필요가 있으며 특히, 공유 되고 있는 객체가 다른 데이터베이스에 중복되어 관리되고 있는 경우에는 그 중복된 객체에도 변화가 적절히 반영되어야 한다. 변화관리 기법이란 공유된 객체가 수정되었을 때, 발생된 변화를 관련된 다른 객체 및 관련된 사용자들에게 제때에 알리게 하는 기법으로 그룹 협동 시스템과 분산 컴퓨팅 환경의 부상과 더불어 그 중요성이 점점 부각되고 있다.

구조적으로 볼 때 분산 환경에서의 그룹 협동 시스템은 대화식 사용자 인터페이스, 분산 컴퓨팅, 실시간 시스템, 객체저장 기법 등 여러 기술분야의 혼합이 요구된다. 그러나 개별적인 기술의 발달에 비교해 볼 때 그룹 협동 시스템 관점에서 변화관리를 지원해 주기 위한 개별기술의 통합은 아직 초보적 단계에 머물고 있다. 예를 들어, Smalltalk환경의 표준 사용자 인터페이스로 제공되는 Model-View-Controller (MVC) 체계[12, 22]의 경우 변화관리 기능을 지원하기는 하지만, 일시적 (transient) 메모리 상에 존재하는 일시적 공유객체에 관련한 종속관계는 지원하나 영속적 (persistent) 공유객체나 여러 명이 동시에 사용할 수 있는 분산시스템으로까지는 아직 발전되지 못하고 있다. 한편, 객체에 기반을 두고 대규모의 복잡한 응용시스템을 구현하기 위하여 개발된 Argus[16], Emerald[11], GUIDE[13] 같은 기존의 분산 프로그래밍 체제[4]의 경우, 분산환경에서 여러 프로세스에 의해 관리되는 객체들이 자율적으로 서로 메시지를 주고 받고, 객체를 영속적으로 저장할 수 있는 기능을 제공하나 변화관리에 필요한 종속관계 유지와 변화 통보 기능은 없는 상태이며, 다중 사용자 지원등에 한계를 가지고 있다. 한편, 기존의 변화관리 체계로서 Gamma[8]는 객체들간의 종속관계를 관리하고 한 객체에 변화가 발생한 경우 다른 관련된 객체로의 변화통보를 위한 객체지향 구성체를 제안하였다. 그러나 이 체제 안에서는 모든 객체가 일시적 메모리 상에 있어야 한다는 한계를 가지고 있으며 따라서, 다중 사용자를 지원하지 못한다. 또한, Diaz[6]와 본 논문의 선행 연구[10]에서는 데이터베이스 내에 저장되어 있는 객체가 변화한 경우 이에 대한 사용자 뷰로의 변화통보를 위한 메커니즘을 제시하였다. Diaz는 Event-Condition-Action 규칙을 이용하여 데이터베이스에 저장되어 있는 영속적 객체가 변화한 경우 이에 대한 사용자 뷰가 이러한 변화를 반영할 수 있는 메커니즘을 제시하였고, 본 논문의 선행 연구에서

는 클라이언트/서버 환경에서 객체간의 종속관계 관리와 변화 통보를 위한 객체지향 데이터베이스에 기반을 둔 모형을 제시하였다. 그러나 이들 두 가지 메커니즘들은 데이터베이스 상에 존재하는 영속적 객체에 대한 변화를 관리하고는 있지만 이러한 변화에 영향을 받는 종속적인 객체가 다른 데이터베이스 상에 존재하는 경우, 다시 말해서 종속적인 객체가 영속적인 경우에 대해서는 변화관리 기능을 지원하지 못한다. 한편, 기존의 데이터베이스 시스템에서 데이터에 변화가 발생한 경우, 이에 대한 관리에 대한 연구[1, 3, 20, 21]들은 데이터베이스 상에 존재하는 영속적 객체의 변화를 데이터베이스 상에 존재하는 다른 영속적 객체로 통보할 수 있는 메커니즘은 제공하고 있으나 이러한 객체들을 공유하고 있는 사용자들에게 통보할 수 있는 체계, 다시 말해서, 사용자 뷰로의 변화통보 기능을 제공해 주고 있지 못하다.

분산 환경에서 그룹 협동 시스템을 위한 변화관리 체계는 공유 객체를 여러 사용자가 동시에 접근하여 사용할 수 있도록 공유 객체가 데이터베이스 상에 영속적으로 저장될 수 있어야 하며, 이에 대한 변화가 사용자 뷰와 같은 일시적인 메모리 상에 존재하는 객체로 뿐만 아니라 다른 데이터베이스에 존재하는 중복 객체와 같은 영속적인 객체로도 통보 및 반영이 되어야 한다. 이를 위해서는 일시적 객체와 영속적 객체를 다루기 위한 메커니즘을 하나의 패러다임 안에서 통합하는 것이 필요하다. 이러한 통합용 패러다임은 앞에서 기술한 바와 같은 대화식 사용자 인터페이스, 분산 컴퓨팅, 실시간 시스템, 객체저장 기법 등과 같은 변화관리를 위한 개별적 기술들을 가능한 한 하나의 개념틀안에서 포용할 수 있어야 하며, 변화관리 기법을 이해하기 쉽고 모듈화가 용이하도록 모형화하며, 일반적이고 유연한 모델링 체계를 가져서, 그룹 협동 시스템이 보다 쉽게 개발되고 다양한 응용시스템으로 확장될 수 있도록 지원해야 한다.

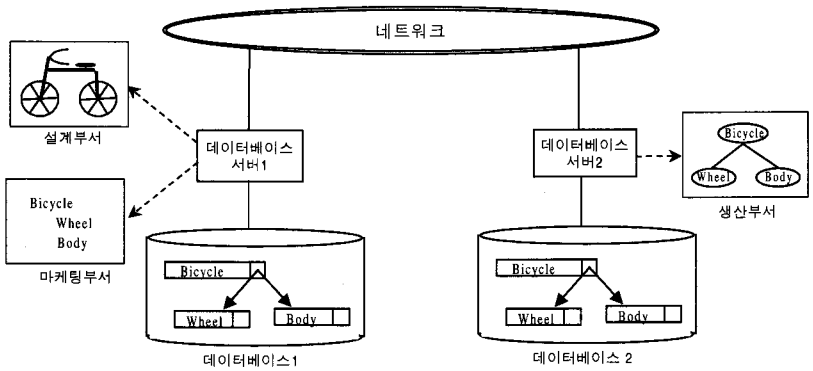
본 논문에서는 분산 그룹 협동 시스템에서의 변화관리를 위한 객체지향모델을 제시한다. 모델의 기반으로서 특히 객체지향 데이터베이스 시스템을 채택하였는데, 이것은 객체지향 데이터베이스 시스템이 일시적 종속객체와 영속적 종속객체를 하나의 형식론 (formalism)안에서 자연스럽게 모형화할 수 있게 해주며 또한, 일반적으로 객체 지향 모델이 분산 시스템이나 구조적으로 복잡한 시스템에 효과적으로 사용될 수 있도록 모듈화가 용이하며, 여러 가지 응용 시스템으로의 확장을 용이하도록 하는 상속(inheritance)이나 다형성 (polymorphism)과 같은 메커니즘을 제공하기 때문이다[2, 4, 19].

본 논문의 구성은 다음과 같다. 먼저, 2장에서 분산환경에서의 변화관리의 개념과 기본적 구성체에 대하여 소개하고자 한다. 3장에서는 변화의 주체가 되는 객체와 그 변화에 영향을 받는 객체간의 종속관계를 관리하기 위한 메커니즘에 대하여 설명하며, 4장에서는 변화의 주체가 되는 객체에 변화가 발생한 경우 그 변화에 영향을 받는 객체로의 통보를 위한 메커니즘을 설명한다. 그리고, 5장에서 본 논문의 결론을 정리하고자 한다.

Ⅱ. 분산환경에서 변화관리의 개념과 기본적 구성체

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

<그림 1>의 데이터베이스에 있는 자전거 설계도는 한 회사의 설계, 생산, 마케팅 부서의 담당자들이 각 부서에서 동시에 참조하고 수정하는 대상이며, 이러한 데이터는 분산환경에서 다른 데이터베이스에 중복관리 될 수 있다. 이와 같은 그룹 협동 컴퓨팅 환경하에서, CAD

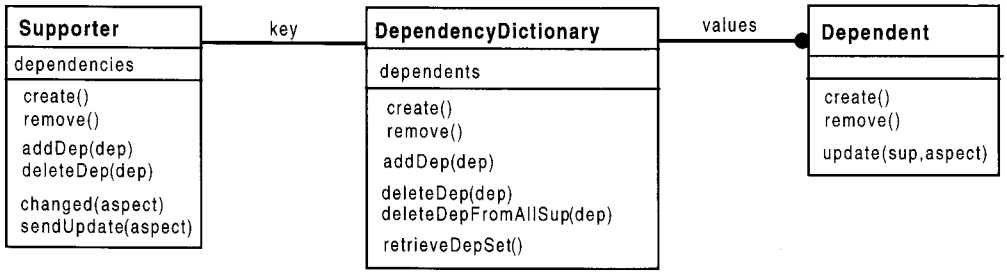


<그림 1> 분산된 공유객체와 여러 가지 사용자 뷰

설계 부서에서는 이 도면을 구성하는 트리구조의 객체를 <그림 1>에서와 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 <그림 1>에서와 같이 계층적인 구조를 보여주는 뷰를 통하여 참조하게 된다. 또한 마케팅 부서에서는 그들의 필요에 맞게, 자전거의 설계에 따른 원가 구조를 정확히 파악하기 위해 <그림 1>과 같이 리스트 형태의 화면을 만들어 참조하게 된다. 그리고 데이터베이스에 저장되어 있는 제품에 대한 정보는 <그림 1>에서와 같이 데이터베이스 1과 데이터베이스 2에 중복되어 관리될 수 있으며 부서간에 서로 다른 데이터베이스를 참조할 수 있다 (<그림 1>의 예에서는 설계부서와 마케팅부서는 데이터베이스 1을 참조하고 생산부서는 데이터베이스 2를 참조).

이처럼 하나의 복잡한 객체를 중복관리하면서 여러 가지 다른 뷰로 나타내 줄때 가장 기본적으로 요구되는 기능 하나는 중복된 객체간 그리고 공유 객체와 이를 참조하는 뷰간의 상호 일관성이다. 중복된 객체 또는 사용자 뷰들이 공유 객체를 항시 반영하고 있으려면, 공유 객체가 어떤 한 뷰를 통하여 수정되었을때 중복된 객체 또는 다른 뷰들은 이러한 변화를 자동적으로 통보 받아 수정되어야 한다. 이러한 유기적 관계를 유지하기 위해서는 종속관계와 변화통보로 구성되는 두 가지 형태의 연결관계가 필요하게 된다. 먼저 종속관계는 사용자 뷰들이 공유 객체에 종속적이며 이 공유 객체에 변화가 생긴 경우 이를 반영하여야 함을 나타낸다. 다음으로 변화통보는 중복된 객체나 사용자 뷰들이 공유 객체의 상태변화를 계속적으로 주시함으로써 공유 객체의 현재 상태을 반영해야 하며 반대로 공유 객체는 그것에 종속된 객체들에게 항상 자신의 변화를 통보해 주어야 하는 의무가 있음을 나타낸다. <그림 1>의 예에 나타난 개념은 일반적으로 다음과 같이 정의될 수 있다.

- 후원자 (supporter)는 가장 중요한 자료원으로서 참고 되고 수정되는 공유 객체이다. 이것은 데이터베이스에 저장된 영속적 객체이며, <그림 1>의 예에서는 데이터베이스에 저장되어 있는 자전거의 구조를 나타낸 트리가 후원자가 된다.
- 종속자 (dependent)는 후원자의 구조를 사용자의 관점이나 필요에 맞게 표현하는 객체이거나 후원자와 관련이 있는 다른 영속적 메모리에 저장되어 있는 객체이다. <그림 1>의 예에서 설계부서, 마케팅부서 및 생산부서에서 사용하는 뷰나 데이터베이스 2에 저장되어 있는 자전거에 대한



<그림 2> 변화관리를 위한 기본적 구성체 (클래스)의 정의

데이터가 종속자가 된다.

- 종속성 사전 (dependency dictionary)은 후원자와 종속자간의 종속관계를 관리하기 위한 관리도구 객체로서 한 후원자에 대한 종속자의 생성과 소멸을 동적으로 관리한다.

이와 같은 기본적 구성체간의 관계와 내부구조를 단순화시켜서 Rumbaugh의 객체모형기법 (OMT : Object Model Technique[19])을 이용하여 그려보면 <그림 2>와 같으며, 각 구성체는 본 연구에서 시스템의 하부구조로 C++프로그래밍 언어를 지원하는 객체지향 데이터베이스 시스템을 사용하였기 때문에 C++로 구현되었다.

<그림 2>에서 후원자 (Supporter) 클래스와 종속자 (Dependent) 클래스는 앞에서 기술한 바와 같이 변화의 주체가 되는 공유객체와 이러한 변화에 영향을 받은 객체를 각각 표현한다. 그리고 종속성 사전 (DependencyDictionary) 클래스는 이러한 후원자와 종속자 간의 종속성 관계를 관리한다. 후원자 클래스의 create()와 remove()는 해당 클래스의 생성자와 소멸자이며, 특히 후원자 클래스가 소멸될 때 만일 이에 종속자가 있다면 종속성 관계의 참조적 무결성을 유지하기 위하여 종속자들도 함께 소멸된다. 종속자 클래스의 create()와 remove() 역시 해당 클래스의 생성자와 소멸자 이며 특히 종속자 클래스가 생성될 때 해당 후원자의 addDep()를 통하여 종속성 관계를 관리하기 위한 종속성 사전에 등록이 되게 된다. 후원자와 이에 대한 종속성 사전은 후원자의 dependencies 속성을 통하여 연결이 되며, 종속성 사전과 종속자는 종속성 사전의 dependents 속성을 통하여 연결이 된다.

종속성 사전은 후원자와 종속자 간의 종속관계를 관리하기 위하여 다음과 같은 세가지 부류의 데이터 조작기능 - 해당 후원자에 종속자의 추가 (addDep()), 제거 (deleteDep(), deleteDepFromAllSup()), 검색 (retrieveDepSet()) - 을 제공한다. 그리고 후원자는 자신의 종속자를 추가, 제거하기 위하여 자신의 addDep(), deleteDep()를 사용하며 이러한 함수들은 내부적으로 종속성 사전의 데이터 조작기능을 이용하게 된다.

후원자의 sendUpdate()와 종속자의 update()는 후원자에 변화가 발생한 경우 이를 후원자가 통보하고 종속자가 반영하기 위한 기능을 나타내며, 이러한 변화통보 기법을 분산환경에 적용하기 위해서는 적절한 관리 메커니즘을 필요로 한다. 이에 대한 설명은 뒤에서 자세히 다루고자 한다.

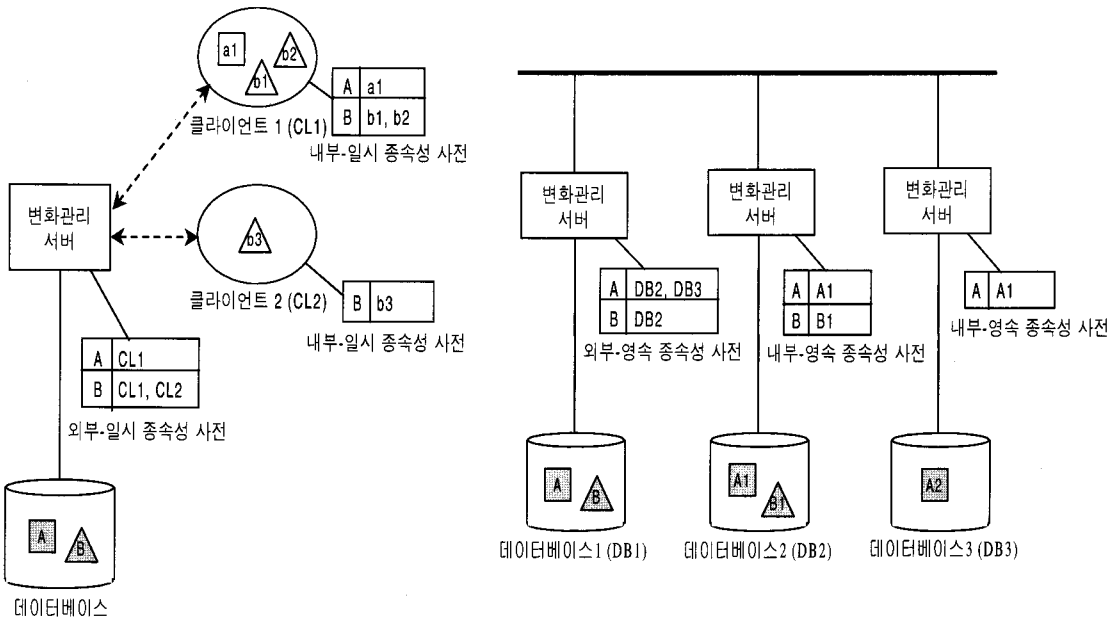
Ⅲ. 분산환경에서 종속관계 관리

앞에서 제시된 변화관리를 위한 기본적인 구성체들을 분산환경, 특히 중복된 객체가 존재하는 상황에 적용하기 위해서는 보다 정교하고 복잡한 메커니즘이 필요하게 되며, 이를 위해서는 앞 절에서 제시한 기본적인 체계의 확장이

필요하다. 앞서 언급된 바와 같이, 변화관리 기법은 종속관계의 관리와 후원자와 종속자간의 변화통보로 구성되며, 이 절에서는 이 두 가지 기능 중 종속관계 관리에 대하여 설명하고자 하며, 다음 장에서 분산환경에서의 변화통보에 대하여 설명하고자 한다.

분산환경에서 데이터베이스에 저장되어 있는 후원자에 대하여 이에 대한 종속자는 일시적 (transient) 종속자와 영속적 (persistent) 종속자로 나눌 수 있다. 일시적 종속자란 사용자 뷰와 같이 일시적 메모리 내에서 생성되며, 이를 생성한 프로그램 세션이 끝나면 함께 소멸된다. 반면에 영속적 종속자는 다른 데이터베이스 시스템 내에 존재하는 중복된 객체와 같이 이를 생성한 프로그램 세션이 끝나도 그 내용이 영속적으로 보존된다. 앞 절에서 제시한 종속성 사전을 이와 같은 분산환경에 적용하기 위해서는 다음과 같이 그 용도에 따라 4가지로 분류하는 것이 필요하며, 먼저 이에 대한 용어를 정리해 보면 다음과 같다.

- 내부-일시 종속성 사전 (ITDD: Internal Transient Dependency Dictionary)은 한 후원자에 대하여 프로세스 내부에 존재하는 일시적 종속자들이 무엇인지를 관리한다.
- 외부-일시 종속성 사전 (ETDD: External Transient Dependency Dictionary)은 한 후원자에 대하여 프로세스 외부에 존재하는 일시적 종속자들을 관리하는 다른 프로세스들이 무엇인지를 관리한다.
- 내부-영속 종속성 사전 (IPDD: Internal Persistent Dependency Dictionary)은 한 후원자에 대하여 프로세스 내부에 존재하는 영속적 종속자들이 무엇인지를 관리한다.
- 외부-영속 종속성 사전 (EPDD: External Persistent Dependency Dictionary)은 한 후원자에 대하여 프로세스 외부에 존재하는 영속적 종속자들을 관리하는 다른 프로세스들이 무엇인지를 관리한다.



(a) 일시적 종속자의 관리 (b) 영속적 종속자의 관리

<그림 3> 종속관계 관리를 위한 종속성 사전

종속성 사전의 내부와 외부의 구분은 내부 종속성 사전은 자신의 프로세스 안에 존재하는 구체적인 종속자를 관리하며, 외부 종속성 사전은 자신의 프로세스 밖에 존재하는 종속자를 관리하는 프로세스가 무엇인가를 관리한다. 한편, 내부-일시 종속성 사전과 외부-일시 종속성 사전은 일시적 종속자를 관리하기 위한 종속성 사전이고 내부-영속 종속성 사전과 외부-영속 종속성 사전은 영속적 종속자를 관리하기 위한 종속성 사전이다. 위에서 제시한 4가지 종속성 사전을 개념적으로 예를 들어 설명하면 <그림 3>과 같다.

<그림 3>에서 데이터베이스내의 회색으로 표시된 삼각형과 사각형은 데이터베이스 내에서 관리되는 영속적 객체를 나타내며 클라이언트 내의 흰색의 삼각형과 사각형은 같은 모양의 영속적 객체의 사용자 뷰를 나타낸다. 먼저, (a) 일시적 종속자의 관리의 예를 보면, 데이터베이스 내에 A라는 공유객체가 있고 이에 대한 뷰가 클라이언트1 (CL1)에 있다. 이때, 외부-일시 종속성 사전은 A라는 공유객체의 뷰가 CL1이라는 클라이언트에 있다는 사실만을 관리하고, 클라이언트1에 있는 내부-일시 종속성 사전이 공유객체 A에 대한 구체적인 뷰가 a1이라는 사실을 관리한다. 이와 같이, 외부-일시 종속성 사전은 해당 변화관리 서버가 관리하는 후원자에 대하여 어느 클라이언트에 뷰가 생성되어 있는가를 관리하고, 내부-일시 종속성 사전은 클라이언트 내에 구체적으로 어떤 뷰들이 생성되어 있는가를 관리한다. 종속성 사전을 변화관리 서버가 관리하는 외부-일시 종속성 사전과 클라이언트가 관리하는 내부-일시 종속성 사전으로 구분함으로써 서버는 구체적인 종속자 (사용자 뷰)들을 모두 관리할 필요 없이 종속자를 가지는 클라이언트가 무엇인가 만을 관리하고 실제로 해당 클라이언트 내에서 구체적인 종속자들을 관리하게 된다. 이렇게 함으로써, 클라이언트가 많아지는 경우 종속관계에 대한 관리가 서버에 집중되어 성능 (performance)을 저하시키는 것을 방지할 수 있다.

한편, (b) 영속적 종속자의 관리의 예를 보면, 데이터베이스 1 안에 B라는 공유객체가 있고 이에 대한 중복된 객체가 DB2라는 데이터베이스 내에 존재하게 된다. 이때, 앞에서 설명한 일시적 종속자의 관리의 경우와 비슷한 방법으로 외부-영속 종속성 사전은 B라는 공유객체의 중복된 객체가 DB2라는 데이터베이스에 있다는 사실만을 관리하고, 데이터베이스 2에 있는 내부-영속 종속성 사전이 공유객체 B에 대한 구체적인 중복된 객체가 B1이라는 사실을 관리한다. 이와 같이, 외부-영속 종속성 사전은 해당 변화관리 서버가 관리하는 후원자에 대하여 어느 데이터베이스에 종속자가 있는가를 관리하고, 내부-영속 종속성 사전은 데이터베이스 내에 구체적으로 어떤 종속자들이 있는가를 관리한다.

Ⅳ. 분산환경에서의 변화통보

앞 절에서 제시한 4가지 종속성 사전을 이용하여 종속관계를 관리하는 상황에서 후원자에 변화가 발생한 경우, 이를 종속자에게로 통보해 주어야 한다. 이러한 변화통보의 관점에서 일시적 객체들은 단일 사용자에 의해서 사용되는 객체들로서 이들 간의 변화관리에서는 시스템적인 장애가 발생하지 않는 한 비일관성 (inconsistency)이 나타나지 않는다. 그러나 데이터베이스 내에 저장되는 영속적 객체들은 다중 사용자에 의한 병행 접근이 허용되기 때문에 이에 따른 서로 다른 작업 간의 충돌이 발생할 수 있다. 따라서 영속적 객체가 후원자 또는 종속자로서 역할을 하는 경우, 변화통보시 서로 다른 객체 간에 데이터의 일관성 (consistency)을 보장해 주기 위한 별도의 메커니즘이 필요하다. 이 절에서는 이러한 문제들을 해결하기 위하여 변화 통보시 필요한 메커니즘에 대하여 설명하고자 하며, 실제로 변화 통보를 실행하게 되는 변화관리 서버

(ChangeManagerServer)와 변화관리 클라이언트 (ChangeManagerClient)라는 프로세스에 대하여 설명하고자 한다.

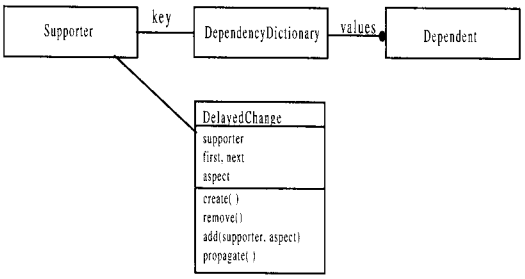
4.1 영속적 후원자와 변화의 유보 지연

본 논문에서 대상으로 하고 있는 후원자는 영속적 후원자로서 후원자가 데이터베이스 시스템에 저장되어서 프로그램 세션이 끝나도 계속 그 내용이 보존된다. 일반적으로 후원자의 영속성은 파일시스템에서 단순히 후원자의 스냅샷 (snapshot)을 제공하는 기능만으로도 지원이 된다 할 수 있으나, 본 논문에서는 다중 사용자의 동시 병행접근까지 허용해야 한다는 점에서 이보다는 좀더 복잡한 기능을 전제한다. 즉, 영속적 후원자를 지원하려면 다중 사용자에 의해 후원자가 공유될 수 있어야 하며, 이러한 상황에서 변화관리기법이 지원되어야 한다. 이것은 곧 후원자가 데이터베이스 시스템에 저장되어 있더라도 변화관리 메커니즘이 후원자의 상태변화를 계속 살펴서 변화발생시 이 변화를 즉시 종속자에게 알려주어야 함을 의미한다. 그러나 다중 사용자의 접근을 허용하는 데이터베이스에 저장되어 있는 영속적 후원자의 경우는 변화에 대해 즉각적인 변화통보는 바람직하지 않다. 왜냐하면, 다중 사용자가 데이터베이스 시스템에 저장된 후원자를 동시에 병행 접근하여 수정하다 보면 그 사용자들의 작업간에 상호 충돌이 발생하여 짧은 순간이나마 후원자 상태를 오류의 상태로 만들 수 있기 때문이다. 이러한 오류발생시에는 후원자가 변화된 것 같아도 그 변화 상태가 유효한 것으로 판정되어질 때까지 변화통보를 연기할 필요가 있다. 변화 상태의 유효성을 판단하는 근거로 본 논문에서는 데이터베이스 시스템 내에서 개체 무결성 (object integrity), 순차성 (serializability), 영속성 (permanence) 등을 보장하기 위한 메커니즘으로 사용되는 트랜잭션 관리 기법을 이용한다. 트랜잭션 관리는 시작점과 종료점의 경계를 가지고 있으며, 경계내의 모든 변화는 개별적으로 보호되어 트랜잭션이 끝날 때까지 다른 사용자는 볼 수 없다. 트랜잭션이 실행되는 동안 한 경계 내에서 트랜잭션은 보통 여러 단계를 거친다. 우리는 네가지의 대표적인 단계들 - 시작 (begin), 시완료 (precommit), 완료 (commit), 중지 (abort) - 에 초점을 맞추는데 이러한 단계들이 변화통보 기법과 관련하여 활용될 수 있기 때문이다. 시작단계는 트랜잭션이 시작되는 시점이다. 트랜잭션 경계내의 일련의 수정작업이 모두 끝나면 다음의 시완료 단계에 도달한다. 이 단계에서는 현재의 트랜잭션이 실행중인 다른 트랜잭션을 간섭하지 않았는지 확인하기 위해서 몇 가지 병행성 검사를 한다. 검사가 성공적이면 완료단계에 도달하고 트랜잭션은 성공적으로 끝난다. 그러나 병행성 검사에서 문제가 발생하면 중지단계로 가서 트랜스액션이 끝날 때 모든 수정작업을 무효화한다. 이상의 각 단계들은 사용자 프로시저와 함께 정의 될 수 있는데, 각 프로시저 내에 트랜잭션 관리기법을 이용한 변화관리 메커니즘을 삽입하여서, 후원자의 저장, 제거, 변화통보 등의 변화관리 기법이 각 단계에 효과적으로 실현되게 할 수 있다.

트랜잭션 관리의 관점에서 보면 종속자는 트랜잭션 경계 내에서 작업이 완료된 수정작업에 대해서만 변화를 통보 받을 필요가 있다. 반대로 트랜잭션이 중지된 경우는 중간에 후원자가 부분적으로 수정이 되었더라도 아직 유효한 변화가 아니므로 종속자가 알 필요가 없다. 따라서 일련의 여러 작업 중 각 개별 작업의 수행결과는 종속자에게 즉시 통보되기 보다는 하나씩 저장되어서 전체가 끝날 때까지 통보를 유보하는 것이 바람직하다. 그리고 이 변화의 통보는 그 트랜잭션 안의 모든 단위 작업이 끝나서 트랜잭션이 완료단계에 이를 때에 시작되도록 한다.

트랜잭션 경계 내에서 변화 통보를 용이하게

하고 변화의 유보 지연을 관리하기 위해 지연된 변화 (DelayedChange) 클래스가 제공된다 (<그림 4> 참조).



<그림 4> 지연된 변화의 객체 정의

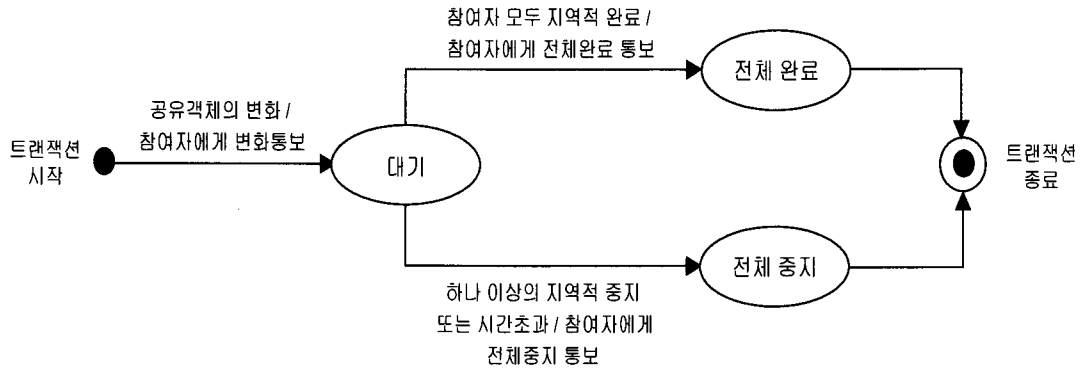
지연된 변화는 후원자의 추상화된 전달자 객체로서 종속자에게 무엇을 어떻게 변화시킬 것인가를 전달하는 역할과 트랜잭션의 중지가 발생하여 데이터를 초기상태로 복귀 (rollback) 시킬 때 초기상태를 알 수 있도록 하는 역할을 한다. 지연된 변화는 변화작업이 트랜잭션 경계 내에서 수행될 때 만들어 진다. 실제 수행과정을 상술하면, 후원자의 수정함수가 수행되어 후원자의 changed() 함수가 작동되면, changed() 함수 내에서 지연된 변화 객체 (DelayedChange)가 자동적으로 만들어 진다. 이때 지연된 변화 객체의 속성 중에서 supporter는 변화된 후원자의 값을 가지며, aspect는 변화내용을 기록한다. 또한 지연된 변화 객체는 first, next 속성을 통해 연결 리스트로 계속 이어지게 되므로, 한 트랜잭션 내에 여러 개의 지연된 변화 객체가 생겨도 이를 시간적 발생순서에 의거하여 순차적으로 보관할 수 있게 된다. 후에 수정 트랜잭션이 성공적으로 완료되면, 하나의 객체 리스트로 연결된 유보 지연된 변화 리스트가 일시에 종속자들에게 통보되게 된다.

4.2 영속적 종속자와 2단계 완료

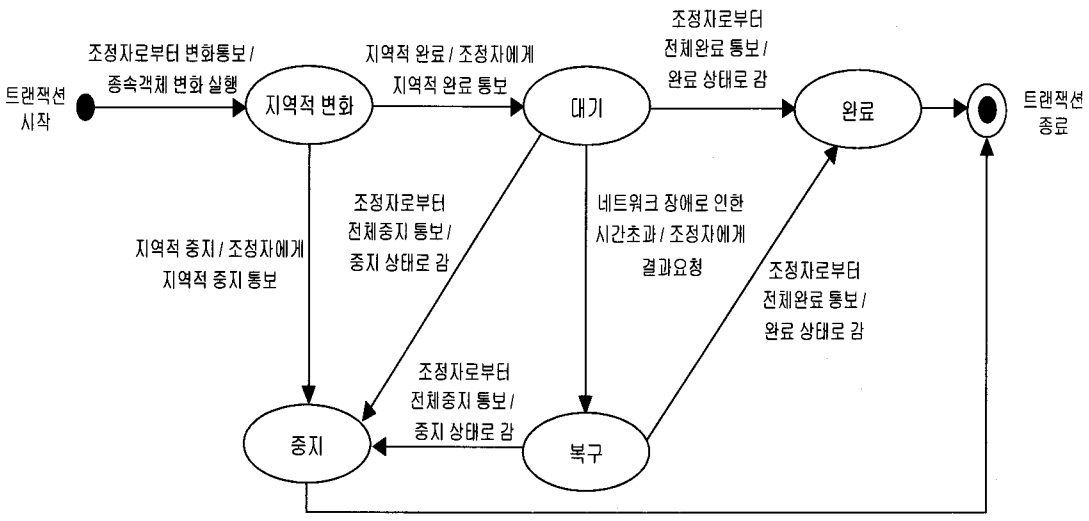
종속자가 일시적인 경우에는 변화된 후원자를 관리하는 변화관리 서버가 종속자를 관리하는 프로세스로 변화를 통보하게 되면 변화관리 서버의 역할은 그것으로 끝나게 된다. 그러나 종속자가 영속적인 경우에는 종속자를 수정하는 과정에서 다중 사용자의 여러 작업간에 충돌이 발생할 수 있고, 따라서 후원자는 수정이 되었으나 종속자는 수정이 되지 않아 비일관성이 발생할 가능성이 있다. 이러한 문제를 해결하기 위하여 본 논문에서는 분산환경에서의 2단계 완료 규약 (two-phase commit protocol)을 보장하기 위한 한 메커니즘을 사용한다. 2단계 완료를 위해서는 변화된 영속적 후원자를 관리하는 프로세스가 조정자 (coordinator)의 역할을 하게 되며, 이에 대한 영속적 종속자를 관리하는 프로세스가 참여자 (participant)의 역할을 하게 된다. 영속적 종속자를 가지는 후원자에 변화가 발생한 경우 이에 대한 변화통보에 따른 조정자와 참여자의 상태 (state)의 변화를 객체모형기법 (OMT: Object Model Technique)의 상태변화도 (state transition diagram)로 나타내보면 <그림 5>와 같다. <그림 5>에서 타원은 조정자와 참여자 각각의 상태를 나타내며, 화살표는 상태의 변화를 나타낸다. 화살표 위에 있는 주석은 두 가지 구성요소를 가지는데 사선('/')앞에 있는 부분은 해당 상태 변화를 일으키는 사건 (event)를 나타내며, 뒤에 있는 부분은 이러한 사건이 일어났을 때 취하게 되는 작용 (action)을 나타낸다.

후원자에 변화가 발생한 경우 조정자의 역할을 하는 변화관리 서버는 먼저 후원자에 영속적 종속자가 있는가를 조사한다. 영속적 종속자가 없는 경우에는 후원자의 수정 트랜잭션이 완료로 끝나는 경우, 앞에서 설명한 지연된 변화 객체의 내용을 일시적 종속자에게 통보함으로써 변화관리 서버의 역할은 끝나게 된다. 반면에 영속적 종속자가 있는 경우에는 변화관리 서버는 영속적 종속자를 관리하는 프로세스에 대하여 2단계 완료를 보장하기 위한 조정자의

역할을 하게 된다. 먼저, 후원자의 변화 트랜잭션이 정상적으로 종료된 경우, 변화관리 서버는 후원자의 수정 트랜잭션을 지역적 완료(local-commit)로 끝내도록 한다. 지역적 완료란 한 데이터베이스 내에서는 트랜잭션이 정상적으로 종료되었으나 만일, 이 트랜잭션과 관련된 다른 데이터베이스 내에서의 트랜잭션이 중지되는 경우 복귀될 수 있는 상태를 의미한다. 트랜잭션의 복귀를 위해서는 변화된 객체의 초기 상태를 알아야 하는데 초기상태 정보는 앞에서 설명한 지연된 변화 객체에 의해 관리된다. 후원자의 수정 트랜잭션이 지역적 완료로 끝난 경우 영속적 종속자로의 변화통보 프로세스가 시작되는데 이에 따른 조정자 (후원자를 관리하는 변화관리 서버)와 참여자 (영속적 종속자를 관리하는 변화관리 서버)의 상태의 변화는 다음과 같다.



(a) 조정자



(b) 참여자

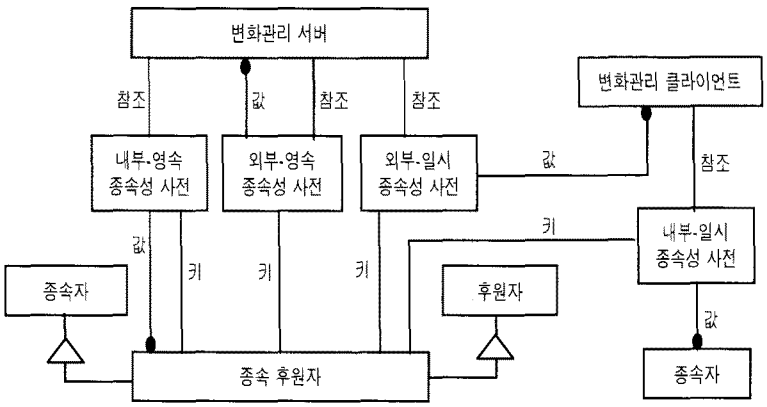
<그림 5> 2 단계 완료에 따른 조정자와 참여자의 상태의 변화

먼저, <그림 5>에서 (a) 조정자의 경우를 보면, 변화통보 프로세스가 시작되면 조정자는 영속적 종속자를 관리하는 모든 참여자에게 후원자의 변화를 통보하고 자신은 대기 상태가 되어 참여자들로부터 지역적 완료 또는 중지의 회답을 기다리게 된다. 참여자들로부터 모두 지역적 완료의 회답을 받은 경우에는 조정자는 참여자들에게 전체 완료 (global-commit)를 통보하게 된다. 한편, 하나 이상의 참여자로부터 중지의 회답을 받거나 일정 시간동안 회답을 받지 못한 경우 전체 트랜잭션은 중지로 끝나게 되며, 조정자는 참여자들에게 전체 중지 (global-abort)를 통보한다. 전체중지를 통보받은 참여자들은 자신의 지연된 변화 객체를 이용하여 초기상태로 복귀하게 된다.

한편, <그림 5>에서 (b) 참여자의 경우를 보면, 조정자로부터 변화를 통보 받으면서부터 참여자의 지역적 트랜잭션은 시작된다. 조정자로부터 변화를 통보 받으면 참여자는 지연된 변화 객체에 변화내역을 저장하면서 자신이 관리하는 영속적 종속자를 변화시킨다. 변화시키는 과정에서 정상적인 완료로 끝날 수도 있고 중지로 끝날 수도 있다. 참여자에서의 수정 트랜잭션이 완료로 끝난 경우, 참여자는 조정자에게 지역적 완료를 통보하고 대기상태로 들어가 조정자로부터 전체완료 또는 전체중지가 오기를 기다린다. 만일, 참여자의 수정 트랜잭션이 중지로 끝난 경우에는 조정자에게 지역적 중지를 통보하고 자신은 중지상태로 끝나게 된다. 한편, 대기상태로 들어간 참여자가 전체 완료를 통보받으면 자신의 상태를 지역적 완료에서 전체 완료로 바꾸고 끝내게 되며, 전체 중지를 통보받으며 자신의 변화를 초기 상태로 복귀하고 중지 상태로 끝내게 된다. 만일, 대기상태로 들어간 참여자가 조정자로부터 미리 정한 일정시간 동안 아무런 회답을 받지 못하면 회답을 받지 못한 이유 (예를 들어, 네트워크 장애 등)를 찾아 이를 복구해 주어야 하며, 그 이후, 조정자에게 회답을 받지 못한 수정 트랜잭션의 최종 결과를 요청하게 된다. 그리고 그 결과에 따라 전체 완료 또는 전체 중지로 끝을 내게 된다. 이와 같이 2단계 완료 규약을 변화관리 서버 간에 구현함으로서 영속적 객체간의 변화통보 과정에서 발생할 수 있는 후원자와 영속적 종속자간의 비일관성 문제를 해결할 수 있다.

4.3 변화관리 서버와 클라이언트간의 통신

앞에서 설명한 변화통보 프로세스는 실제로 데이터베이스를 관리하는 시스템에 존재하는 변화관리 서버라는 프로세스와 클라이언트에 존재하는 변화관리 클라이언트라는 프로세스간

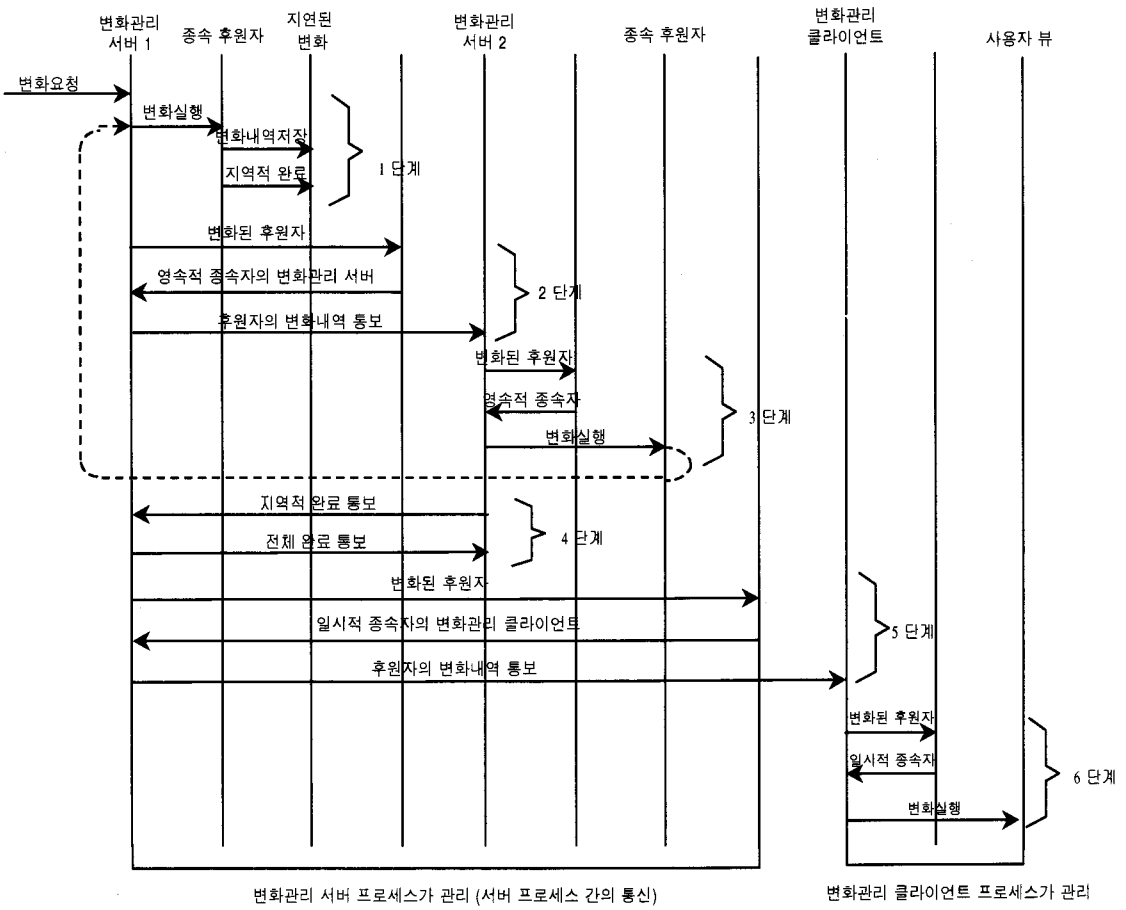


<그림 6> 분산환경에서 변화관리를 위한 객체데이터 모델

의 통신을 통해서 이루어진다. 각각의 프로세스는 서버와 서버간 또는 서버와 클라이언트간의 통신을 위한 오퍼레이션들을 가진다. 변화관리 서버와 변화관리 클라이언트 그리고 4가지 종속성 사전 간의 관계를 객체모형기법 (OMT) [19]을 이용하여 표현하면 <그림 6>과 같다.

<그림 6>에서 종속 후원자 (dependent supporter)는 종속자와 후원자로부터 동시에 계승을 받은 클래스로서 데이터베이스에 저장되는 영속적 객체가 이에 해당한다. 이러한 영속적 객체는 중복된 객체인 경우에는 다른 영속적 객체에 대하여 종속자의 성격을 가지는 동시에 이에 대한 사용자 뷰에 대해서는 후원자의 성격을 가진다. 따라서 데이터베이스 내에 존재하는 영속적 객체는 실제로 종속 후원자라는 클래스의 인스턴스가 된다.

또한, <그림 6>에서와 같이 변화관리 서버는 외부-영속 종속성 사전, 외부-일시 종속성 사전 그리고 내부-영속 종속성 사전을 참조하며, 변화관리 클라이언트는 내부-일시 종속성 사전을 참조한다. 다시 말해서, 후원자 (실제로는 종속 후원자)에 변화가 발생한 경우 외부에 있는 영속적 종속자 (실제로는 종속 후원자)를 관리하는 변화관리 서버를 찾기 위하여 외부-영속 종속성 사전을 참조하며, 내부에 있는 영속적 종속자를 찾기 위하여 내부-영속 종속성 사전을 참조한다. 또한 외부에 존재하는 일시적 종속자를 가지는



<그림 7> 변화통보의 사건추적도

변화관리 클라이언트를 찾기 위하여 외부-일시 종속성 사전을 참조한다. 한편, 변화관리 클라이언트는 자신이 가지는 구체적인 일시적 종속자를 찾기 위하여 내부-일시 종속성 사전을 참조한다.

변화관리 서버와 변화관리 클라이언트 프로세스 간의 통신을 통한 변화통보 과정의 예를 사건추적도 (event trace diagram)[17, 19]를 통하여 그려보면 <그림 7>과 같다. 사건추적도는 사건의 발생 순서와 관련된 객체간의 관계를 표현해 주는데 효과적이다. 사건추적도에서 객체는 수직선으로 표시되며 사건은 수평선으로 표시되는데 사건의 발생 순서는 위에서부터 아래방향이다.

<그림 7>에서 1단계는 변화관리 서버 1이 관리하는 후원자 (실제로는 종속 후원자)에 변화가 요청되면서 시작된다. 변화를 요청 받은 변화관리 서버는 변화내역을 지연된 변화 객체에 저장하며 변화를 실행하고 변화가 정상적으로 종료된 경우 이를 지역적 완료로 끝내게 된다. 2단계는 변화관리 서버 1이 자신이 관리하는 외부-영속 종속성 사전을 참조하여 영속적 종속자를 가지는 변화관리 서버 (<그림 7>의 예에서는 변화관리 서버 2)를 찾아내어 후원자의 변화내역을 통보하게 된다. 3단계에서는 변화를 통보 받은 변화관리 서버 2는 자신이 관리하는 내부-영속 종속성 사전을 참조하여 영속적 종속자를 찾아내게 되며, 찾아진 종속자들에게 변화를 실행하게 된다. 이때, 영속적 종속자들이 자신의 또 다른 영속적 종속자를 가지는 경우 자신의 외부-영속 종속성 사전을 참조하여 변화를 통보하는 과정이 재귀적으로 되풀이 되게 된다. 이러한 상황을 <그림 7>에서는 3단계에서 점선으로 표시하였다. 4단계에서는 영속적 종속자를 가지는 변화관리 서버 2가 수정 트랜잭션을 정상적으로 종료한 경우 후원자를 가지는 변화관리 서버 1에게 지역적 완료를 통보하게 되고 이를 통보 받은 변화관리 서버 1은 변화관리 서버 2에게로 전체 완료를 통보하게 된다. 5단계부터는 일시적 종속자에게로의 변화통보과정이며, 5단계에서는 변화관리 서버 1은 자신의 외부-일시 종속성 사전을 참조하여 일시적 종속자들을 가지는 변화관리 클라이언트를 찾아내어 후원자의 변화내역을 클라이언트에게로 통보해 주게 된다. 마지막으로 6단계에서는 클라이언트 내에서 자신의 내부-일시 종속성 사전을 참조하여 자신이 가지는 일시적 종속자 (사용자 뷰)에 후원자의 변화를 반영하게 된다.

V. 결 론

그룹 협동 시스템 환경하에서 공유 객체가 다중 사용자에 의한 접근 및 수정이 허용될 때, 각 사용자는 자신의 필요에 따라 서로 다른 여러 가지 뷰를 가지게 되며 이러한 뷰들은 공유 객체가 수정이 될 때 마다 이에 대한 변화를 즉각적으로 반영해야 할 필요가 있다. 특히, 분산 환경하에서의 그룹 협동 시스템에서는 공유 객체를 저장 관리하는 데이터베이스 시스템이 지역적으로 두 곳 이상에 분산되어 있을 수 있고 이들 간에는 공유 객체가 중복 관리될 수 있다. 이러한 분산환경에서 하나의 공유 객체가 수정된 경우 다른 중복된 공유 객체로 변화에 대한 통보가 이루어져야 한다. 이러한 필요성에 대하여 본 논문에서는 공유 객체에 대한 변화를 관리하기 위한 객체지향모델을 제시하였다. 특히, 본 논문에서는 공유 객체의 변화가 통보되어야 하는 대상인 종속자가 일시적인 메모리 상에 존재하는 경우 (예를 들어, 사용자 뷰)와 영속적인 메모리 상에 존재하는 경우 (예를 들어, 다른 데이터베이스 상에 존재하는 중복된 객체)를 하나의 형식론 (formalism)안에서 해결할 수 있도록 하였다.

변화관리 모형의 핵심적인 내용은 변화의 주체인 후원자와 변화의 영향을 받는 객체인 종속자 간의 종속관계 관리와 후원자에 변화가

발생한 경우 종속자로의 변화통보이다. 본 논문에서는 종속관계 관리를 위한 도구로서 종속성 사전이라는 추상화된 객체를 사용하는데 이는 일시적 종속자를 관리하기 위한 내부-일시 종속성 사전과 외부-일시 종속성 사전 그리고 영속적 종속자를 관리하기 위한 내부-영속 종속성사전과 외부-영속 종속성 사전으로 나누어 진다. 종속성 사전의 내부와 외부의 구분은 내부 종속성 사전은 자신의 프로세스 안에 존재하는 구체적인 종속자를 관리하며, 외부 종속성 사전은 자신의 프로세스 밖에 존재하는 종속자를 관리하는 프로세스가 무엇인가를 관리한다. 이와 같이 종속관계에 대한 관리를 서버와 클라이언트 간에 자신이 필요로 하는 부분으로 한정함으로써 종속관계를 한곳에서 집중하여 관리할 때 서버 및 클라이언트의 수가 늘어남에 따라 나타날 수 있는 성능저하를 막을 수 있다. 한편, 분산환경에서의 변화통보 과정에서 영속적 객체가 개입되는 경우에는 일시적 객체간의 변화통보의 경우 보다 정교한 관리 메커니즘을 필요로 한다. 왜냐하면, 데이터베이스에 저장되어 있는 영속적 객체의 경우는 다중 사용자의 병행접근을 허용하므로 이에 따른 서로 다른 작업 간에 상호충돌이 발생할 수 있다. 만일, 충돌이 발생하여 영속적 객체에 대한 수정 작업이 중지된 경우에는 수정 작업을 초기 상태로 되돌려야 하며 종속자로의 변화통보는 되지 말아야 한다. 이러한 문제를 해결하기 위하여 본 논문에서는 지연된 변화 객체를 사용한다. 또한 영속적 종속자로의 변화통보의 경우에는 후원자를 수정한 이후 종속자를 수정하는 과정에서 중지가 일이 날 수 있으며, 이러한 경우에는 이미 진행된 후원자의 수정 사항이 초기 상태로 복귀되어야 한다. 이러한 문제를 해결하기 위하여 본 논문에서는 분산환경에서의 2단계 완료 규약을 보장하기 위한 변화관리 서버간의 관리 메커니즘을 사용한다. 2단계 완료를 보장하기 위하여 후원자를 관리하는 변화관리 서버가 조정자의 역할을 하게 되며, 영속적 종속자를 관리하는 변화관리 서버가 참여자의 역할을 하게 된다.

본 논문에서 제시하는 분산환경에서의 변화관리를 위한 객체지향 모형은 추후 건축용 블록처럼 다른 객체에 재사용 되는 것을 목표로 개발되었다. 다시 말해서, 후원자와 종속자 클래스들을 기초로 하여 이 클래스를 계승 받는 파생 후원자 및 파생 종속자를 만듦으로서 다양한 응용 시스템에 분산환경에서의 변화관리 메커니즘이 내재화 (embedded)되게 할 수 있다.

본 논문에서 제시하는 기법의 원형 시스템은 Windows NT시스템에서 객체지향 데이터베이스 시스템인 ObjectStore위에 C++를 이용하여 개발되었으며, 향후 연구과제로서, CAD/CAM 엔지니어링 디자인, 모형관리 영역, 그룹 의사결정 지원시스템 등과 같은 분야에의 적용을 연구하고 있다.

〈참 고 문 헌〉

[1] Awerbuch, B. and L. J. Schulman, "The Maintenance of Common Data in a Distributed System," *Journal of the ACM*, Vol. 44, No. 1, Jan. 1997, pp. 86-103.

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

[3] Chakravarthy, S., "Rule Management and Evaluation: An Active DBMS Perspective," *SIGMOD Record*, Vol. 18, No. 3, Sep. 1989, pp. 20-28.

[4] Chin, R. and S. Chanson, "Distributed Object-Based Programming Systems," *ACM Computing Surveys*, Vol. 23, No. 1, 1991, pp. 91-124.

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

[6] Diaz, O., A. Jaime, N. W. Paton, and G. al-Qaimari, "Supporting Dynamic Displays Using Active Rules," *SIGMOD Record*, Vol. 23, No. 1, March 1994, pp. 21-26.

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

[8] Gamma, E., R. Helm, R. Johnson, and J. Vlissides, *Design Patterns : Elements of Reusable Object-Oriented Software*, Addison-Wesly, 1994.

[9] 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.

[10] Huh, S.-Y. and Dave Rosenberg, "A Change Management Framework: Dependency Maintenance and Change Notification," *Journal of Systems and Software*, 1995, pp. 1-16.

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

[12] 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, 1988, pp. 26-49.

[13] 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*, 1990, pp. 11-22.

[14] D. McGoveran, "Two-Phased Commit or Replication," *Database Programming and Design*, May 1993, pp. 35-44.

[15] Lamb, C. G. Landis, J. Orenstein, and D. Weinreb, "The ObjectStore Database System," *Communications on ACM*, Vol. 34, No. 10, 1991, pp. 50-63.

[16] Liskov, B. "Distributed Programming in Argus," *Commun. ACM*, Vol. 31, No. 3, 1988, pp. 300-312.

[17] Mullin, M, *Object-Oriented Program Design with Examples in C++*, Addison-Wsley, 1989.

[18] Nunamaker, J., A. Dennis, J. Valacich, and D. Vogel, "Information Technology for Negotiating Groups: Generating Options for Mutual Gain," *Management Science*, Vol. 37, No. 10, 1991, pp. 1325-1346.

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

[20] Rusinkiewicz, M., A. Sheth, and G. Karabatis, "Specifying Interdatabase Dependencies in a Multidatabase Environment," *IEEE Computer*, 1991, pp. 46-52.

[21] Segev, A. and J.-S. Park, "Updating Distributed Materialized Views," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 1, No. 2, June 1989, pp. 173-184.

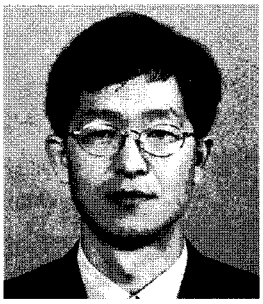
[22] Shan, Y.-P., "An Event-Driven Model-View-Controller Framework for Smalltalk,"

Proceedings of Object-Oriented Programming Systems, Languages, and Applications : OOPSLA, October 1-6, New Orleans, Louisiana, 1989, pp. 347-352.

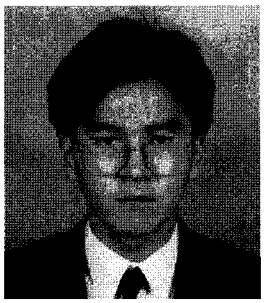
[23] Theel, O. E. and B. D. Fleisch, "A Dynamic Coherence Protocol for Distributed Shared Memory enforcing High Data Availability at Low Costs," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 9, Sep. 1996, pp. 915-930.

◈ 이 논문은 1998년 접수하여 5월 1일 1차 수정을 거쳐 1998년 9월 21일 게재확정되었습니다.

◆ 저자소개 ◆



허순영(Huh, Soon-Young)
현재 한국과학기술원 테크노 경영대학원 조교수로 재직중이다. 서울대학교 전자공학과에서 공학사 (1981), 한국과학기술원 경영과학과에서 공학석사 (1983), 미국 UCLA 대학교 경영대학원에서 경영정보시스템 분야로 경영학 박사학위 (1992)를 취득하였다. 주요 관심분야는 객체지향 기술을 토대로 한 의사결정지원 시스템, 지능형 금융 트레이딩 시스템, 컴퓨터 기반 그룹웨어 등이다.



김형민(Kim, Hyung-Min)
현재 한국과학기술원 테크노 경영대학원 박사과정에 재학중이다. 한국과학기술원 경영과학과에서 이학사 (1993) 및 공학석사 (1995)를 취득하였다. 주요 관심분야는 모형관리 시스템, 지식관리 데이터베이스 시스템, 객체지향 데이터베이스 기반 그룹협동 시스템 등이다.