UML 모델링과 COM을 기반으로 한 컴포넌트의 런타임 갱신

(UML Modeling and COM based Runtime Updating of Component)

정 한 식 † 김 일 곤 †† 김 황 수 †††
(Han Shik Jung) (Il Kon Kim) (Hwang-Soo Kim)

요 약 컴포넌트의 런타임 갱신은 컴포넌트를 사용하는 프로그램을 종료하지 않고, 기존에 사용하는 컴포넌트를 그 내부가 새로 변경된 컴포넌트로 교환하는 기술이다. 최근 소프트웨어 개발에서는 기존 컴포넌트를 재활용하여 새로운 소프트웨어를 개발하는 추세이다. 그러나 컴포넌트가 시간 측면에서 연속적인 서비스를 제공해야하는 소프트웨어에 사용될 때 컴포넌트를 새롭게 기능이 바뀐 것으로 변경하려면 기존 소프트웨어를 종료해야한다. 이 경우 기존 소프트웨어의 서비스를 제공받는 사용자는 서비스를 다시 요구해야 되며 때로는 자신이 사용하는 정보를 손실하는 경우도 발생한다. 따라서, 본 논문에서는 기존 컴포넌트 갱신 방법의 문제점을 해결하기 위하여 소프트웨어를 종료하지 않고 그 소프트웨어를 구성하는 컴포넌트를 새로운 컴포넌트로 갱신하기 위한 방법을 제시한다.

Abstract Runtime updating of component is a technology which is replacing component used in program with newly implemented component without terminating running program using old one. Recently, it is trend that softwares are developed by reusing the existing component. When we use these components to softwares which have to offer consecutive services, they must be terminated or shutdown to update component with newly implemented one. In this case, users who are offered services of software have to request them one more time and sometimes lose informations which they use. Therefore, this paper suggests how to update components with newly implemented one without terminating the running software which is composed with the component for the purpose of solving problems which are occurred when existing components are being updated.

1. 서 론

본 논문에서는 UML(Unified Modeling Language)[1, 2]을 이용하여 컴포넌트를 모델링(Modeling)하는 방법을 제시하고, UML로 표현된 모델을 바탕으로 런타임(Runtime)에 갱신할 수 있는 기능을 가진 컴포넌트를 생성하는 방법을 제안하며, 그 동작 원리를 기술한다.

† 비 회 원 : (주)오픈테크 연구원
HSJUNG@apollo.otech.co.kr
†† 정 회 원 : 경북대학교 컴퓨터과학과 교수
ikkim@knu.ac.kr
††† 종신회원 : 경북대학교 컴퓨터과학과 교수
hsk@knu.ac.kr
논문접수 : 1999년 12월 29일
심사완료 : 2000년 10월 30일

런타임에 갱신 가능한 컴포넌트를 런타임 갱신 컴포넌트(Runtime Update Component)라 하며, 그 컴포넌트는 COM(Component Object Model)[3, 4]을 기반으로 한 인프락 서버(In-proc server) 컴포넌트로 만들어진다.

기존 컴포넌트는 그 개발단계에서부터 클라이언트 프로그램에 의한 컴포넌트의 사용 단계까지 런타임에 컴포넌트를 갱신하기 위한 방법으로 몇 가지 제약을 가지고 있다. 본 논문에서는 그 중 세 가지 제약을 보완하여 컴포넌트를 런타임에 갱신할 수 있는 특성을 가지게 하는 방법을 제시한다.

첫째, 기존 컴포넌트는 인터페이스(Interface) 부분과 내부구현(Implementation)으로 구성되어, 인터페이스와 내부구현을 분리시켜 놓았다. 이러한 모델로 인해 내부

구현에 필요한 모든 데이타의 접근이 불가능하며, 컴포넌트의 일부 데이타만 그 값을 변경할 수 있도록 설계된다. 이러한 컴포넌트의 구성은 컴포넌트의 런타임 갱신에 필요한 컴포넌트의 상태변경 지원을 불가능하게 만든다. 본 논문에서 컴포넌트의 상태변경은 기존 컴포넌트의 인스턴스(Instance)를 새로운 컴포넌트의 인스턴스로 변경하는 것을 의미한다.

둘째, 기존 컴포넌트는 클라이언트 프로그램이 컴포넌트의 기능을 사용할 수 있는 인터페이스만을 제공한다. 즉, 컴포넌트의 런타임 갱신을 위한 인터페이스를 지원하지 않으며, 기존 컴포넌트의 인터페이스와 새로운 컴포넌트의 인터페이스를 일치시키지 못 한다. 클라이언트 프로그램이 기존 컴포넌트의 기능을 사용하면서, 그 컴포넌트를 다른 컴포넌트로 대치할 경우, 기존 컴포넌트가 사용하고 있는 정보들을 다른 컴포넌트에서 다시 사용할 수 있게 만드는 인터페이스가 제공되어야한다.

셋째, 기존 컴포넌트를 이용한 클라이언트 프로그램은 자신이 사용하는 컴포넌트의 종류가 미리 결정되어있다. 즉, 클라이언트 프로그램이 사용하고 있는 컴포넌트와 이질성을 지니고 있는 컴포넌트로 변경이 불가능하다. 이점은 컴포넌트의 재사용성(Reusability)이 클라이언트 프로그램을 개발단계에서만 적용되며, 클라이언트 프로그램이 실행단계에서는 적용되지 않음을 보여준다.

본 논문에서는 앞에서 기술한 기존 컴포넌트의 세 가지 제약을 극복하여 컴포넌트의 런타임 갱신을 지원을 위한 컴포넌트 생성의 새로운 정형화 방법을 제시한다. 컴포넌트 개발기술은 COM[5]을 기반으로 하며, 컴포넌트 생성에 필요한 모델링 도구는 Rational 사에서 제공하는 Rose[6]를 사용한다.

컴포넌트의 런타임 갱신에 관한 연구로는 Java를 이용한 SOFA/DCUP가 있다.[7] SOFA는 기존 컴포넌트가 사용하는 IDL(Interface Definition Language)대신에 자체적으로 사용하는 CDL(Component Description Language)을 사용한다. CDL은 컴포넌트의 인터페이스와 함께 컴포넌트의 내부구조를 표현한다. 컴포넌트의 내부 구조를 표현함으로써 컴포넌트의 런타임 갱신을 위한 구조 정보를 프레임워크(Framework)를 제공한다. DCUP는 컴포넌트의 런타임 갱신을 위한 컴포넌트의 내부구조를 정의한 것이다. 컴포넌트의 구조는 변경 가능한 부분과 변경할 수 없는 부분으로 나누어 놓았으며, 이러한 구조를 이용하여 컴포넌트 생성을 위한 프레임워크를 제공한다. SOFA/DCUP와 본 논문의 차이는 CDL 대신 UML을 사용하고, COM을 기반으로 하며, 기존 컴포넌트 갱신 방법보다 수행속도를 향상시킨 점이 있다.

본 논문의 구성은 2장에서 컴포넌트의 런타임 갱신 지원을 위해 필요한 구성요소를 제시하고, 3장에서는 런타임 갱신 컴포넌트의 구조 및 알고리즘을 기술한다. 4장은 런타임 갱신 컴포넌트를 이용하는 클라이언트 프로그램의 예를 들고, 성능을 분석하며, 5장에서 결론을 맺는다.

2. 구성요소

본 논문에서는 컴포넌트의 런타임 갱신 지원을 위해 필요한 두 가지 요소를 제시한다. 하나는 런타임 갱신의 기능을 가진 컴포넌트를 생성하기 위한 컴포넌트 생성 도구의 지원이고, 다른 하나는 런타임 갱신의 기능을 가진 컴포넌트를 관리하기 위한 컴포넌트 관리 도구이다.

컴포넌트 생성부는 런타임 갱신 컴포넌트를 생성하기 위한 프레임워크[8, 9]를 구성하고, 개발자는 프레임워크를 바탕으로 컴포넌트의 내부를 구현한다. 런타임 갱신 컴포넌트는 클라이언트 프로그램에 의해 사용되면, 컴포넌트 관리 도구에 등록된다. 그림 1은 런타임 갱신 컴포넌트를 위한 시스템의 전체 구조이다.

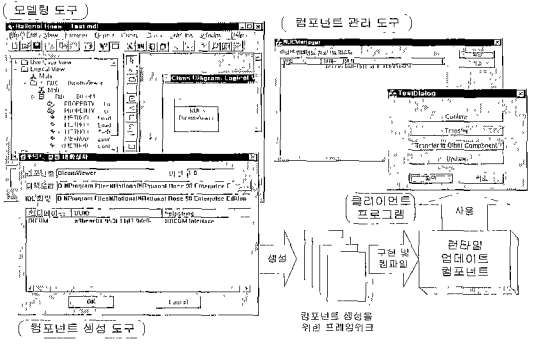

그림 1 런타임 갱신 컴포넌트를 위한 시스템 구조

컴포넌트 개발자는 런타임 갱신 컴포넌트 생성을 위한 프레임워크를 이용하여 기존 컴포넌트 생성 방법과 똑같은 방법으로 컴포넌트를 만들 수 있다. 이렇게 만들어진 컴포넌트 관리 도구와 연계되어 사용된다. 클라이언트 프로그램도 기존 컴포넌트를 사용하는 방법과 같은 방법으로 사용하고, 컴포넌트에서 런타임 갱신을 위해 제공되는 메소드(Method)를 부가적으로 사용할 수 있다.

컴포넌트 생성도구를 사용하여 생성된 컴포넌트 프레임워크로 개발된 컴포넌트는 클라이언트 프로그램에 의

해 사용되는 시점에 컴포넌트 관리도구에 등록되며, 관리자는 등록된 컴포넌트를 컴포넌트 관리도구를 이용하여 다른 컴포넌트로 변경할 수 있다. 본 논문에서 제시하는 컴포넌트 관리도구는 클라이언트 프로그램이 사용하는 컴포넌트 및 컴포넌트가 생성한 인스턴스들을 클라이언트 프로그램의 수행에 영향을 주지 않고 변경시킨다.

3. 런타임 갱신 컴포넌트

런타임 갱신 컴포넌트는 자신이 가지고 있는 정보를 다른 컴포넌트에 전달할 수 있고, 다른 컴포넌트의 함수 구현부분을 자신의 것으로 만들 수 있다.[10] 컴포넌트의 갱신은 독립적인 기능을 가진 컴포넌트 단위로 이루어진다.

3.1 컴포넌트 모델링

런타임 갱신 컴포넌트를 만들 때, 중복되는 작업이 많고, 객체의 상태정보 교환을 위한 정형화된 자료형태를 만들어야 한다. 이러한 작업을 간편하게 하기 위해, 컴포넌트 모델링을 통해 컴포넌트 생성을 위한 프레임워크를 제공한다.

컴포넌트 모델링은 UML의 패키지 다이어그램(Package Diagram)과 클래스 다이어그램(Class Diagram)을 사용한다. 클래스 다이어그램은 객체의 자료형과 이들간의 관계를 기술함으로써 표현된다. 컴포넌트 모델링은 하나의 패키지 내에 포함된 클래스들로 표현되며, 각 클래스들은 하나의 객체 클래스를 정의하고 객체 클래스가 인터페이스에서 지원하는 메소드의 종류 및 속성(Property)을 결정한다. 컴포넌트 모델링에서 패키지는 하나의 컴포넌트와 대응된다.

컴포넌트 모델링은 런타임 갱신 컴포넌트의 생성방법을 정형화하여 이질성을 지닌 컴포넌트간에도 호환성을 가질 수 있게 만들어 기존 컴포넌트가 가진 이질성을 지닌 컴포넌트간에 호환될 수 없는 제약을 극복한다.

컴포넌트 모델을 컴포넌트 생성 도구에 입력으로 주면, 컴포넌트 생성 도구는 모델을 바탕으로 런타임 갱신을 지원하는 컴포넌트의 코드를 생성한다. 코드 생성은 크게 세 가지로 나눌 수 있다.

첫째는 인터페이스의 생성이다. 인터페이스는 컴포넌트 모델에서 스테레오타입(Stereotype)이 정의된 클래스 다이어그램에서만 생성된다. 둘째는 내부구현 코드의 생성이다. 내부 구현 코드는 그림 2의 구조로 만들어지며, 구현 클래스의 인터페이스는 컴포넌트 생성 도구를 통해 생성된 것을 사용한다. 셋째는 DLL(Dynamic Linking Library) 코드의 생성이다. DLL 코드는 객체 생성 랩퍼(Wrapper) 클래스를 전역으로 선언하여 만들어지며, 레지스트리에 컴포넌트를 등록하거나 등록된 내용을 삭제하는 함수를 제공한다.

그림 2는 컴포넌트 생성도구가 만들어내는 코드 파일의 종류를 나타낸 것이다.

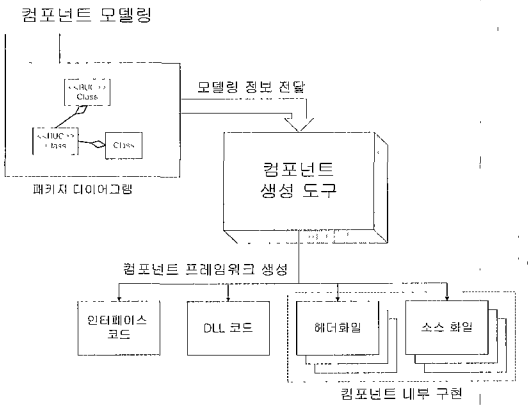

그림 2 컴포넌트 프레임워크 생성

인터페이스 파일과 DLL 코드 파일은 컴포넌트의 이름을 사용하여 만든다. 그리고, 내부 구현 코드는 컴포넌트 모델에서 정의된 클래스의 이름을 이용하여 만들어진다. 내부 구현 코드는 객체 클래스, 객체 랩퍼 클래스, 객체생성 클래스, 객체생성 랩퍼 클래스 네 개로 구성되어 있으며, 각 클래스에 대해 헤더파일과 소스파일을 생성한다.

3.2 컴포넌트 구조

런타임 갱신 컴포넌트는 컴포넌트가 가진 모든 클래스에 런타임 갱신을 위한 일반화된 메소드들을 가진다. 본 논문에서는 그 방법으로 컴포넌트의 런타임 갱신을 위한 메소드들을 기본 인터페이스에 정의한 IRUC 인터페이스를 제공한다. 런타임 갱신 컴포넌트에서 제공하는 모든 인터페이스는 IRUC 인터페이스를 상속받는다. IRUC에서 제공되는 메소드는 기존 컴포넌트에서는 제공되지 않는 것으로 컴포넌트의 내부 상태 정보 접근을 가능하게 한다.

IRUC 인터페이스는 컴포넌트의 내부 상태를 교환하기 위한 방법을 제공하여, 컴포넌트의 블랙박스의 특성으로 인한 데이타 교환이 어려운 문제를 해결한다. 그리고, 컴포넌트가 가진 내부 구현부분을 서로 주고받을 수 있는 방법을 제공하여, 클라이언트 프로그램이 사용하는 컴포넌트의 인터페이스에서 제공되는 메소드들의 구현을 런타임에 변경할 수 있게 한다.

런타임 갱신 컴포넌트는 기존 객체생성 클래스와 다르게 객체 클래스가 생성한 구현 객체를 관리하기 위한 기능을 가지고 있다. 객체생성 클래스의 인터페이스는 IClassFactory 인터페이스를 상속받아 만들어지며, 자신이 생성한 객체 인스턴스를 관리하기 위한 메소드를 제공한다.

런타임 갱신 컴포넌트의 내부구조[7]는 기존 컴포넌트의 내부구조[3]를 변경하여 런타임에 컴포넌트를 변경할 수 있는 구조로 만들어져 있다. 기존 컴포넌트의 내부구조는 컴포넌트에서 제공하는 각 인터페이스를 상속받아 구현한 객체 클래스와 객체 클래스의 인스턴스를 생성하기 위한 메소드를 제공하는 객체생성 클래스로 구성되어 있다. 기존 COM에서 객체생성 클래스는 IClassFactory 인터페이스를 상속받아 만들어지는 클래스이다. 런타임 갱신 컴포넌트는 런타임 갱신 지원을 위해, 기존 컴포넌트의 구조에서 랩퍼를 사용한 구조로 변경한다. 즉, 런타임 갱신 컴포넌트는 랩퍼를 이용하여 컴포넌트간의 구현 부분을 서로 교환하는 구조로 만들어지게 된다. 또한 랩퍼는 컴포넌트의 런타임 갱신을 위한 컴포넌트의 관리 기능을 제공한다. 그림 3은 런타임 갱신 컴포넌트의 내부구조이다.

런타임 갱신 컴포넌트는 하나의 객체 클래스마다 하나의 객체생성 클래스를 가진다. 객체생성 클래스는 컴포넌트의 사용자의 요청에 따라 컴포넌트가 제공하는 인터페이스를 상속받아 만든 객체 클래스의 인스턴스를 생성하는 역할을 한다. 런타임 갱신 컴포넌트는 컴포넌트의 실제 구현 부분을 랩퍼로 감싸서 랩퍼 내의 구현 객체를 상호 교환할 수 있게 만든다. 클라이언트 프로그램은 컴포넌트의 인터페이스만을 이용하여 컴포넌트를 사용하기 때문에, 컴포넌트의 내부구조가 어떠한 방법으로 만들어져 있는지는 중요하지 않다.

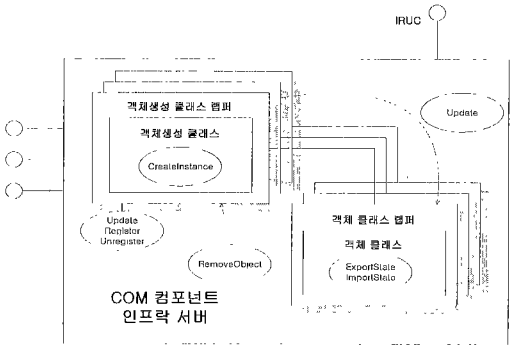

그림 3 런타임 갱신 컴포넌트의 내부구조

런타임 갱신 컴포넌트는 COM에서 제공하는 방식 중에서 인프락 서버로 만들어진다. 인프락 서버는 DLL을 기반으로 만들어지며, 클라이언트의 프로세스 내에서 독립적인 기능을 수행한다. 런타임 갱신 컴포넌트의 런타임 갱신 기능은 클라이언트 프로그램이 사용하는 컴포넌트를 런타임에 변경하여 클라이언트 프로그램의 실행 코드를 프로그램의 실행 중에 변경시킨다.

객체 클래스는 컴포넌트의 인터페이스를 상속받아 만들어지는 인터페이스의 구현 클래스이다. 객체 클래스는 객체 랩퍼 클래스에 포함되어 있으며, 그 클래스의 구현 객체는 랩퍼 객체 사이의 이동이 가능하다. 객체 클래스의 이러한 특징은 인터페이스가 가진 구현 객체의 변경을 가능하게 한다. 객체 클래스의 인터페이스는 런타임에 갱신을 위한 메소드를 가진 IRUC 인터페이스를 상속받는다.

객체 클래스는 IRUC를 상속받은 인터페이스를 사용하기 때문에, 이들 인터페이스에 대한 구현 메소드를 지원해야한다. 그러나, IRUC의 모든 메소드를 객체 클래스가 사용하는 것은 아니다. 이는 IRUC 인터페이스를 객체 랩퍼 클래스에서도 동일하게 사용되고 있기 때문이며, 메소드 중에는 랩퍼에서만 사용되는 것들이 있기 때문이다.

컴포넌트 모델은 컴포넌트 내의 자료 표현에 중점을 두고 있기 때문에, 이 자료를 정형화된 스트림(Stream)으로 변경하는 작업을 런타임 갱신 컴포넌트가 지원한다.

그림 4는 객체 인스턴스의 상태정보를 스트림으로 저장하는 형식을 나타낸 것이다.[11, 12]

상태정보는 상태정보에 대한 정보를 가진 해더정보와 객체의 상태를 저장하는 내용정보로 구성된다. 내용정보는 객체 클래스의 각 멤버들에 대한 정보를 저장한다. 그림 4에서 각 필드의 아래에는 각 필드가 스트림에서 차지하는 길이 정보를 표현하였다.

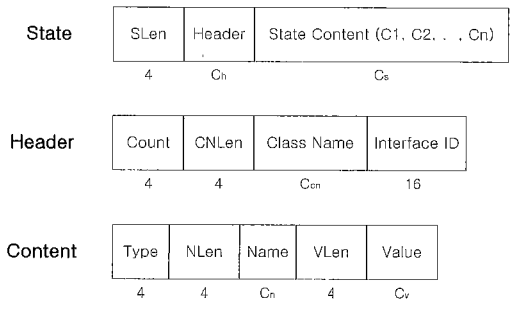

그림 4 상태정보 표현 방법

SLen은 객체 인스턴스의 상태정보를 저장하기 위한 스트림의 공간을 메모리에 확보하기 위해 사용된다. Export에서는 메모리에 할당된 스트림에 헤더정보와 내용정보를 저장한다. Import는 스트림에 저장된 상태 정보를 바탕으로 구현 객체의 상태를 변경한다. 상태변경은 멤버 데이타의 자료형과 이름을 이용하여 일치하는 내용이 있으면, 해당하는 필드의 값을 변경한다.

객체 랩퍼 클래스는 객체를 갱신하는 경우, 외부에서 객체를 참조하고 있는 레퍼런스를 그대로 사용하기 위한 것이다. 객체의 갱신은 메모리 상에서 새로운 객체로 만들어지기 때문에, 기존의 객체가 가지고 있는 자신의 레퍼런스가 다른 객체의 레퍼런스로 변경되어야 한다. 그러나 객체를 참조하고 있는 외부의 객체를 모두 저장하고 있는 경우, 객체의 관리를 위한 공간상의 오버헤드가 커진다. 이를 해소하기 위해 객체에 랩퍼를 두어 메모리의 공간을 낭비하는 오버헤드를 해결한다.

객체생성 클래스는 클라이언트 프로그램이 사용하고자 하는 인터페이스의 구현 객체를 생성해 주는 역할을 한다. 객체생성 클래스는 IClassFactory 인터페이스를 상속받은 IRUCClassFactory 인터페이스를 사용한다. 객체 생성 클래스가 생성하는 객체는 랩퍼 클래스의 객체이며, 클라이언트 프로그램은 이 객체를 인터페이스를 이용하여 접근한다. 객체 랩퍼는 서비스의 요청에 따라 구현 객체의 메소드를 수행한다.

객체생성 랩퍼 클래스는 객체생성 클래스가 생성한 객체를 관리하고, 객체생성 객체의 갱신을 위한 메소드를 제공한다. 객체생성 랩퍼 클래스는 내부의 객체생성 클래스의 인스턴스 종류를 변경하여, 새로운 구현 객체를 생성할 수 있다. 객체생성 랩퍼 클래스는 객체생성 클래스와 같이 IRUCClassFactory 인터페이스를 사용한다.

3.3 컴포넌트 갱신 알고리즘

클라이언트 프로그램이 객체생성 랩퍼에 구현객체의 생성을 요구하면, 객체생성 랩퍼는 객체생성 객체에게 새로운 객체 랩퍼를 생성한다. 객체 랩퍼는 클라이언트 프로그램이 요구하는 구현 객체를 생성하여 내부에 가지고 있다. 클라이언트 프로그램은 객체 랩퍼를 통해 구현 객체에 접근한다.

객체생성 랩퍼는 객체생성 객체가 생성한 객체 랩퍼의 레퍼런스를 배열에 저장한다. 이 배열은 클라이언트가 컴포넌트 인스턴스의 갱신을 요구할 경우, 객체생성 랩퍼에서 기존에 생성된 구현 객체의 종류를 변경하기 위해 객체 랩퍼를 배열에 저장한다.

컴포넌트 갱신은 컴포넌트 인스턴스의 갱신과 레지스트리 등록정보의 변경으로 진행된다.

컴포넌트 인스턴스의 갱신은 구현 객체의 변경과 객체생성 객체의 변경으로 구분되며, 구현 객체의 변경은 클라이언트 프로그램이 사용하는 인터페이스에서 지원하는 메소드의 구현이 변경됨을 의미하고, 객체생성 객체의 변경은 클라이언트 프로그램이 새로 생성하는 구현 객체의 종류를 변경시킨다. 그림 5는 컴포넌트를 갱신하는 알고리즘이다.

```
OCR.Update(NCID);
for each implementation objects
        IORL.Get(i).Update(NCID);

CreateInstance(NCID, &NIOR);
OIO.Export(&state);
NIOR.Import(state);
NIOR.GetInnerObject(&OIO);
NIOR.Close();
```

그림 5 컴포넌트 갱신 알고리즘

OCR은 기존 컴포넌트의 객체 생성 랩퍼이고, IORL은 구현 객체 랩퍼의 리스트이다. NCID는 새로운 컴포넌트의 클래스 식별자(Identifier)이다. NIOR은 새로운 구현 객체 랩퍼이고, OIO는 기존 구현 객체이다.

기존 구현 객체의 상태 정보는 새로운 구현 객체에 전달되며, 객체 랩퍼는 새로운 구현 객체를 가지게된다. 이전 구현 객체는 참조개수가 감소하며, 외부에서 참조되고 있지 않으면, 메모리에서 삭제된다. 이전 객체생성 객체도 구현 객체와 같이 외부에서 더 이상 참조하고 있지 않으면, 메모리에서 삭제된다. 새로 생성된 구현 객체의 랩퍼는 클라이언트 프로그램에 의해 사용되지 않기 때문에, 이전 랩퍼가 새로운 구현객체를 가지고 난 후에, 메모리에서 삭제된다. 그러나 새로운 객체생성 랩퍼는 이전 객체생성 랩퍼가 새로운 객체생성 인스턴스를 가지고 난 후에도 메모리에서 삭제되지 않는다. 객체생성 랩퍼는 DLL에서 전역으로 만들어지기 때문에, 객체생성 랩퍼는 DLL이 메모리에서 사라져야 삭제된다.

컴포넌트 인스턴스 갱신이 끝나면, 컴포넌트 관리도구는 레지스트리에 등록된 기존 컴포넌트에 대한 경로를 새로운 컴포넌트에 대한 경로로 변경한다. 이로서 컴포넌트 갱신은 작업은 종료된다.

4. 적용 및 성능

실질적인 적용 예로는 자동현금지급기 서버를 구성하

여 보았다. 자동현금지급기 서버는 ATM 컴포넌트를 사용하여 구성되어 있으며, ATM 컴포넌트는 런타임 갱신 컴포넌트이다. 그림 6은 수행 중인 자동현금지급기 서버의 ATM 컴포넌트를 컴포넌트 관리 도구를 이용하여 기능이 추가된 컴포넌트로 갱신한 예를 보여준다.

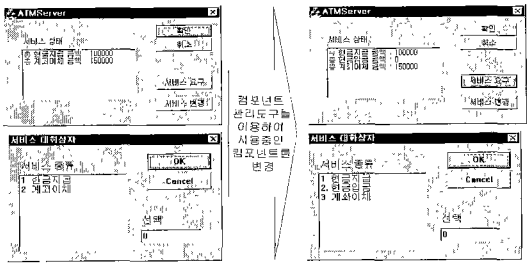

그림 6 실제 프로그램 적용 예

처음에 자동현금지급기 서버는 현금지급과 계좌이체 서비스만을 제공하고, 당일 현금 출입현황을 통계내는 것이었다. 이러한 기능은 내부에 런타임 갱신 컴포넌트에 의해 제공된다. 이 서버가 사용하는 컴포넌트를 컴포넌트 관리 도구를 이용하여 새로운 컴포넌트로 갱신하면, 그림 7의 오른쪽 프로그램과 같이 갱신된다. 새로운 자동현금지급기 서버는 현금입금의 서비스가 추가되었고, 자료 통계에서는 총 현금입금 금액이 추가되었다. 이것은 ATM 컴포넌트의 상태정보가 변경되었음을 보여준다.

예에서 소요시간을 측정한 결과 컴포넌트가 서비스를 수행하는 시간이 컴포넌트를 갱신하는데 걸리는 시간보다 짧다. 컴포넌트 갱신 주기가 짧은 응용영역에서는 실제 컴포넌트 갱신에 걸리는 시간이 더 많이 소요되기 때문에, 적용하기에는 적절하지 못 하다는 것을 보여준다. 그러나, 일반적인 컴포넌트의 갱신은 갱신주기가 길기 때문에, 위의 예와 같은 응용영역에 적절하다고 볼 수 있다. 또한 컴포넌트의 갱신을 일괄적으로 처리하는 SOFA/DCUP의 갱신 방법보다 갱신 속도가 향상됨을 보여준다.

본 논문에서 기술하는 런타임 갱신 컴포넌트는 컴포넌트가 생성한 구현 객체를 일괄적으로 갱신하는 것이 아니라, 지연 갱신의 정책에 따라 컴포넌트의 갱신 시간을 분산시킨다. 클라이언트 프로그램의 사용자가 컴포넌트 갱신을 위해 대기하는 시간이 짧아진다. 일괄적인 갱신 방법에 비한 지연 갱신의 사용자 대기 시간은 다음과 같다.

컴포넌트가 생성한 구현 객체의 개수를 N, 구현 객체의 상태 정보를 스트림으로 저장하는 시간을 T_e^{impl}, 스트림에 저장된 상태 정보를 구현 객체의 상태 정보로 만드는 시간을 T_i^{impl}, 구현 객체를 지연 갱신의 상태로 만드는 시간을 T_d^{impl}, 사용자가 컴포넌트 갱신을 대기하는 시간을 T_w라고 하면,

$$T_w = \sum_{impl=0}^{N-1} \{ T_e^{impl} + T_i^{impl} \} \quad (1)$$

$$= \sum_{impl=0}^{N-1} T_d^{impl} \quad (2)$$

(1)은 일괄적으로 컴포넌트를 갱신할 경우를 나타내고, (2)는 지연 갱신의 경우르 나타낸다. 구현 객체의 상태 정보를 스트림으로 저장하는 시간과 스트림에 저장된 상태 정보를 구현 객체의 상태로 만드는 과정은 구현 객체가 가지고 있는 상태 정보에서 내용 정보의 개수에 의해 결정된다. 내용 정보는 인터페이스를 구현한 객체 클래스의 멤버 데이타의 개수와 같다.

내용 정보의 개수를 M이라 하면 T_e^{impl}와 T_i^{impl}의 시간 복잡도(time complexity)는 각각 $O(M)$이다. 클라이언트 프로그램이 그 컴포넌트의 구현 객체를 N개 생성하면, 일괄적인 컴포넌트 갱신의 경우 사용자 대기시간은 $O(NM)$이고, 지연 갱신의 경우 구현 객체의 지연 갱신 상태 설정은 $O(1)$이 소요됨으로 사용자 대기시간은 $O(N)$이다.

클라이언트 프로그램이 컴포넌트에 서비스를 요청하면, 컴포넌트는 해당하는 구현 객체의 서비스를 수행한다. 구현 객체가 지연 갱신의 상태에 있는 경우, 서비스를 지원하는 객체를 갱신하고 클라이언트 프로그램이 요구한 서비스를 제공한다. 그러므로, 컴포넌트의 서비스 수행 시간은 다음과 같이 된다.

구현 객체의 서비스 수행 시간을 T_f, 컴포넌트 서비스 수행 시간을 T_s라 하면,

$$T_s = T_e^{impl} + T_i^{impl} + T_f \quad (3)$$

$$= T_f \quad (4)$$

(3)은 구현 객체가 지연 갱신의 상태에 있는 경우이고, (4)는 일반적인 서비스의 수행의 경우이다. 구현 객체가 지연 갱신의 상태에 있으면, 서비스 수행 시간은 $O(M)$이 증가된다.

컴포넌트의 지연 갱신은 컴포넌트가 갱신되는 동안 대기 시간을 감소시키나 서비스의 수행 시간은 증가시킨다. 그러나, 전체적인 서비스 요구시간을 분석하면 컴포넌트가 갱신되는 동안 사용자가 대기해야 되는 시간은 $O(N)+O(M)$이 되어 보다 빠른 서비스를 사용자에게 제공할 수 있다.

5. 결 론

본 논문에서는 런타임에 갱신할 수 있는 컴포넌트에 대해 기술하였다. 런타임 갱신 컴포넌트를 만들기 위한 모델링 방법 및 런타임 갱신 구조에 대해 설명하고 이를 바탕으로 컴포넌트 생성 도구를 구현하였다. 또한, 런타임 갱신 컴포넌트의 동작 원리를 기술하고 이를 관리하기 위한 컴포넌트 관리도구를 구현하였다.

런타임 갱신 컴포넌트는 컴포넌트의 상태정보 교환을 통해 컴포넌트의 갱신을 지원한다. 그러므로, 클라이언트 프로그램은 기존 컴포넌트의 상태정보를 이용하여 새로운 컴포넌트의 서비스를 제공받을 수 있다. 또한 컴포넌트의 상태교환을 위한 메소드는 컴포넌트 모델링을 통해 생성되기 때문에, 컴포넌트 개발자는 컴포넌트의 런타임 갱신을 위한 정형화되고, 성능이 향상된 내부구현을 가진 메소드를 제공받는다.

런타임 갱신 컴포넌트는 기존 컴포넌트가 제공하지 않는 상태교환 방법을 지원함으로써 클라이언트 프로그램이 컴포넌트를 새로운 컴포넌트로 갱신할 수 있도록 지원한다. 또한 컴포넌트의 갱신을 런타임에 제공함으로써 클라이언트 프로그램이 컴포넌트를 더욱 유연하고 효과적으로 사용할 수 있도록 한다.

참 고 문 헌

[1] Fowler, M., Scott, K., *UML DISTILLED APPLYING THE STANDARD OBJECT MODELIING LANGUAGE*, Addison Wesley Longman, Inc., 1997.

[2] OMG, *UML Semantics*, version 1.1, 1997. 9.

[3] Box, D., *Essential COM*, Addison Wesley Longman, Inc., 1998.

[4] Chappell, D., *Understanding ActiveX and OLE*, Microsoft Press, 1996.

[5] 전병선, *Inside ATL/COM Programming with Visual C++*, 삼양출판사, 1998.

[6] Rational Rose, Rational Rose 98 Extensibility reference manual, 1998. 2.

[7] Plasil, F., Balek, D., Janecek, R., "SOFA/DCUP: Architecture for Component Trading and Dynamic Updating," Proc. of ICCDS'98, 1998. 5.

[8] Little, M. C., Wheater, S. M., "Building Configurable Applications in Java," Proc. of the 4th IEEE International Conference on Configurable Distributed Systems, 1998. 5.

[9] Oreizy, P., Medvidovic, N., Taylor, R. N., "Architecture-Based Runtime Software Evolution," Proc. of ICSE'98, 1998. 4.

[10] Ridgway, J. V. E., Wileden, J. C., "Toward Class Evolution in Persistent Java™," PJW3 on Persistence and Java, 1998. 9.

[11] Goudarzi, K. M., Kramer, J., "Maintaining Node Consistency in the Face of Dynamic Change," Proc. of 3rd ICCDS, 1996. 5.

[12] OMG, *The Common Object Request Broker: Architecture and Specification*, Revision 2.1, 1997. 8.

정 한 식

1998년 경북대학교 컴퓨터과학과 졸업(학사). 2000년 경북대학교 컴퓨터과학과 석사학위 취득. 2000년 ~ 현재 (주) 오픈테크 연구원. 관심분야는 자동화 시스템, 실시간 OS, 병렬처리, 컴포넌트 모델링.

김 일 곤

1980년 서울대학교 수학교육과 졸업(학사). 1988년 서울대학교 전산과학과 석사학위 취득. 1991년 서울대학교 전산과학과 박사학위 취득. 1992년 ~ 현재 경북대학교 컴퓨터과학과 부교수로 근무. 관심분야는 지능형 에이전트, 분산시스템, 의료정보학.

김 황 수

1975년 서울대학교 전기공학과 졸업(학사). 1982년 Univ. of Michigan 석사학위 취득. 1988년 Univ. of Michigan 박사학위 취득. 1989년 ~ 현재 경북대학교 컴퓨터과학과 부교수로 근무. 관심분야는 computer vision, neural networks, AI임.