공정 제어 응용을 위한 분산 능동 객체 시스템(DAOS)의 설계 및 구현

(Design and Implementation of Distributed Active Object System(DAOS) for Manufacturing Control Applications)

음 두 헌 † 유 은 자 ††
(Doohun Eum) (Eunja Yoo)

요 약 낙공정 제어 응용은 로봇, AGV(Automatic Guided Vehicle), 컨베이어 등과 같이 능동적이면서 병행적으로 동작하는 컴포넌트들로 구성되며 이들간의 상호작용으로 수행이 이루어진다. 객체지향 기술은, 이러한 컴포넌트들을 재사용이 가능한 객체로 모델링하여, 공정 제어 응용의 생산성 및 확장성을 향상시킬 수 있다. 그러나 기존 객체지향 기술의 객체는 상태와 행위 정보만을 캡슐화하여 실세계의 객체를 표현하며, 메시지가 전달되어야만 반응하는 수동 객체(passive object)이다. 본 논문에서 소개하는 분산 능동 객체 시스템(Distributed Active Object System: DAOS) 방식은 Java/CORBA 기반의 분산 환경에서, 객체의 상태와 행위 정보 뿐 아니라 객체 자신의 제어(control) 부분까지 캡슐화한 능동 객체(active object)를 지원하여, 공정 제어 컴포넌트들을 더욱 완전하게 모델링할 수 있다. 여기서, 자신의 제어란 자신의 상태뿐 아니라 인터페이스 변수(interface variable)로 연결된 타 객체의 상태까지 모니터링하고 그 상태 변화에 따라 스스로 행위를 수행할 수 있는 기능을 말한다[1, 2]. 따라서 DAOS 방식은, 메시지 전달을 이용해 각 분산 객체들의 제어를 기술하지 않고, 인터페이스 변수들을 사용하여, 스스로 기동할 수 있는 객체들을 구성적으로 조립하여 시스템을 구축한다. 즉, DAOS 방식은 객체 조립성을 지원하여 기존 객체지향 기술보다 분산 공정 제어 소프트웨어의 생산성 및 확장성을 개선하고, 제어까지 캡슐화된 능동 객체를 지원하여 컴포넌트의 재사용성을 향상시킨다.

Abstract Manufacturing control applications consist of concurrent active components such as robots, AGV's (Automatic Guided Vehicles), and conveyors. Running of manufacturing control programs is interactions among those components. We can enhance the productivity and extendability of manufacturing control applications by using the object-oriented technology that models those components as reusable objects. But the objects in current object-oriented technology that encapsulate state and behavior information are passive in a sense that those respond only when messages are sent to them. In this paper, we introduce the Distributed Active Object Systems (DAOS) approach that supports active objects. Since active objects encapsulate control information in addition to state and behavior information under CORBA/Java-based distributed environment, they can represent manufacturing control components better than the objects in ordinary object-oriented technology. This control information provides an object with a feature that can monitor its own status as well as other object's status connected by interface variables. Active objects can initiate a behavior according to the change of those status. Therefore, we can structurally assemble self-initiating active objects by using interface variables to construct a system without describing how to control distributed objects by using message passing. As the DAOS approach supports object composability, we can enhance the productivity and extendability of distributed manufacturing control applications even better than the ordinary object-oriented approach. Also, the DAOS approach supports better component reusability with active objects that encapsulate control information.

· 본 논문은 2000년도 KISTEP 연구비 지원으로 수행되었음.
· 본 논문은 덕성여자대학교 교내 연구비 지원으로 수행되었음.

† 종신회원 : 덕성여자대학교 전산및정보통신대학원 교수
dheum@namhae.duksung.ac.kr

†† 학생회원 : 덕성여자대학교 전산및정보통신대학원
bittyy@namhae.duksung.ac.kr

논문접수 : 2000년 5월 4일
심사완료 : 2000년 12월 10일

1. 서 론

최근 공정 제어 시스템의 개념이 단일 시스템에서 인터넷/인트라넷으로 연결된 분산 공정 제어 시스템으로 확장되고 있다. 또한, 이러한 소프트웨어의 수요는 나날이 증가하고 있고 그들의 효율적인 관리 및 확장성이 요구되고 있다. 공정 제어 응용은 로봇, AGV(Automatic Guided Vehicle), 컨베이어, 탱크, 밸브 등과 같이 능동적이면서 병행적으로 동작하는 컴포넌트들로 구성되며 이들간의 상호작용으로 수행이 이루어진다[3]. 객체지향 기술은, 이러한 컴포넌트들을 재사용이 가능한 객체로 모델링하여, 공정 제어 응용의 생산성 및 확장성을 향상시킬 수 있다.

현재, 객체지향 기술을 기본으로 하는, 분산 기술로는 EJB(Enterprise JavaBeans)[4, 5], Microsoft의 COM (Component Object Model)/COM+[6] 등이 대표적이고, 컴포넌트 기반의 응용 프로그램 작성 기술로는 Microsoft의 Visual Basic, Sun의 JavaBeans[7] 등이 대표적이다. E-Commerce 응용을 위한 EJB 기술은, 운영체제 상에서 응용 프로그램이 필요로 하는 트랜잭션(transaction), 보안(security), 지속성(persistency) 등의 미들웨어 기능을 제공하는 컨테이너(container) 및 그 안에서 동작하는 컴포넌트들에 대한 표준이다. 그러나 EJB 프로그래밍은 설정된 엄격한 표준을 따라야 하고 컴포넌트들을 이용하여 분산 응용 프로그램을 작성하나 컴포넌트간의 조립성이 미약하다. COM/COM+는, Microsoft 제품들이 기반으로 하는 기술이기 때문에 높은 시장 점유율을 전망하고 있다. 그러나 COM/COM+는 완전한 객체지향 기술이 아니기 때문에 그 프로그래밍이 매우 복잡하다. 컴포넌트 기반 소프트웨어 구축 도구로서 가장 많이 사용되고 있는 것은 Visual Basic이며 최근 Java의 활발한 사용과 함께 JavaBeans 기술이 소개되었다. 그러나 Visual Basic 및 JavaBeans 기술은 단일 시스템 환경에서 주로 GUI 위주의 컴포넌트 기반 기술이다[8].

기존 객체지향 기술의 객체는 상태와 행위 정보를 캡슐화하여 실세계의 객체를 표현하지만, 메시지가 전달되어야만 반응하는 수동 객체(passive object)이다. 본 논문에서 소개하는 분산 능동 객체 시스템(Distributed Active Object System: DAOS) 방식은 Java/CORBA[9, 10, 11] 기반의 분산 환경에서, 객체의 상태와 행위 정보 뿐 아니라 객체 자신의 제어(control) 부분까지 캡슐화한 능동 객체(active object)를 지원하여, 능동적이고 병행적으로 동작하는 공정 제어 컴포넌트들을 더욱 완전하게 표현할 수 있다[2, 12, 13, 14].

그림 1의 DAOS 개념도에서, 능동 객체는 Java/CORBA 객체에 능동 행위 인터페이스를 포장한(wrapping) 객체이다. 능동 행위 인터페이스는 인터페이스 변수(interface variable)를 통해 자신 및 타 객체의 상태를 모니터링하고 그 상태 변화에 의해 지정된 행위를 수행하는 기능을 제공한다. DAOS 방식의 분산 응용 프로그램은 Java/CORBA 객체와 능동 객체, 그리고 그들간의 인터페이스 변수를 통한 상호연결로 구성된다. 그림에서 화살표는 객체들 간의 논리적 연결을 나타낸다. 실선으로 표시한 화살표는 네트워크 상에서 같은 사이트 객체간의 연결을 나타내고, 점선으로 표시한 화살표는 서로 다른 사이트 객체간의 연결로써, 그 물리적 연결은 ORB를 통해 이루어진다.

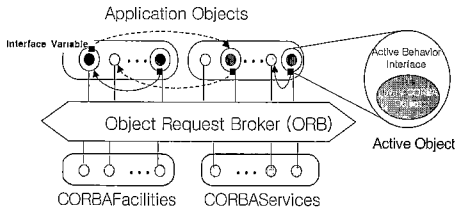

그림 1 DAOS의 개념도

DAOS 방식에 기반한 응용 프로그램은, 메인에서 각 분산 객체들의 제어를 기술하지 않고, DAOSMain 클래스에서 인터페이스 변수들을 사용하여 객체들을 구성적[1])으로 조립하여 구축할 수 있다[1]. 또한, 이렇게 구성된 응용 프로그램을 다른 응용 프로그램과 연결하여 작동시킬 때에도 프로그램간의 인터페이스 변수의 연결로 구성적으로 조립하여 확장할 수 있다. 즉, DAOS 방식은 객체간이나 이들로 구성된 응용 프로그램간의 구성적 조립을 지원하여 공정 제어 소프트웨어의 생산성 및 확장성을 기존 객체지향 기술보다 개선하고, 객체나 응용 프로그램 단위의 컴포넌트 기반 기술이므로 컴포넌트의 재사용성을 향상시킬 수 있으며, 소프트웨어의 유지보수 비용을 절감할 수 있다[15].

2절에서는 간단한 분산 Tank 제어 시스템을 이용해, DAOS 방식의 개요를 소개한다. 3절에서는 DAOS 방식을 기반으로 한 간단한 공정 제어 응용 프로그램의 동작 원리를 소개하고 DAOS 라이브러리의 주요한 구현 사항들에 대해 설명한다. 마지막 절에서는 DAOS 방식으로 공정 제어 응용 프로그램을 작성할 때의 장점을 요약한다.

2. 분산 능동 객체 시스템(Distributed Active Object System) 방식

1) '전체'를 창조해 내기 위해 여러 요소를 결합·배치하여 하나의 예술 작품을 성립시키는 방법. 모아서 조립한다는 의미.

본 절에서는 DAOS 방식의 개요를, 간단한 공정 제어 응용인 분산 Tank 제어 시스템의 작성 예를 이용하여, 소개한다. 먼저, DAOS 방식이 지원하는 구성적 인터페이스(structural interface)의 개념을 설명한 후, 응용 프로그램의 조립식 작성 과정을 보인다. 마지막으로, DAOS 정의 언어(DAOS Description Language: DAOSDL)를 소개한다.

2.1 구성적 인터페이스(Structural Interface)

능동 객체는 스스로의 제어를 캡슐화한 모듈성이 뛰어난 소프트웨어 *IC(Integrated Circuit)*이며, 능동 객체의 인터페이스 변수는 하드웨어 IC의 핀과 같은 역할을 한다. 사운드 카드, 컴퓨터 등의 하드웨어 제품들은, 설계도를 따라 표준화된 하드웨어 IC들을 구성적으로 조립, 연결함으로써 신속하게 설계 및 생산할 수 있다. 같은 원리로 DAOS 방식에서는, 설계한 객체 다이어그램대로 능동 객체들을 배치한 후, 그들의 인터페이스 변수를 이용하여 객체들을 연결만 함으로써, 쉽고 신속하게 공정 제어 응용 프로그램을 작성할 수 있다. DAOS 방식이 지원하는 이러한 능동 객체간의 인터페이스를 본 논문에서는 구성적 인터페이스라 한다[1]. 구성적 인터페이스는, 기존 객체지향 기술이 지원하는 메시지 호출 기반의 절차적 인터페이스(procedural interface)에 비해, 설계하고자 하는 시스템을 보다 간단하고 정확하게 표현할 수 있다.

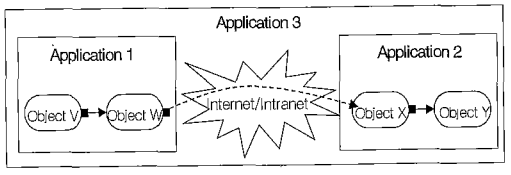

그림 2 DAOS 방식의 구성적 인터페이스

그림 2는 DAOS 방식으로 응용 프로그램을 작성할 때, 객체 단위의 구성적 조립과 응용 프로그램 단위의 구성적 조립의 예를 보인 것이다. 화살표는 객체간의 인터페이스 변수를 통한 연결을 나타낸다. 응용 프로그램 1은 객체 V와 W를 인터페이스 변수를 이용하여 구성적으로 연결하여 완성되고, 응용 프로그램 2도 객체 X와 Y를 인터페이스 변수를 이용하여 구성적으로 연결함으로써 완성된다. 또한, 분산된 응용 프로그램 1과 응용 프로그램 2를 다시 인터페이스 변수를 통해 구성적으로 연결함으로써, 확장된 기능의 분산 응용 프로그램 3을 만들 수 있다.

2.2 응용 프로그램의 조립식 작성

그림 3은 간단한 분산 Tank 제어 시스템의 구성도로서, 2개의 valve 객체와 1개의 tank 객체를 가지는 프로그램 1과, 2개의 valve 객체와 2개의 tank 객체를 가지는 프로그램 2로 구성된다. 화살표는 인터페이스 변수를 통한 객체 연결을 나타내고 탱크 안의 점선은 안전 수위(safety level)를 나타낸다. DAOS 방식으로 이 시스템을 모델링하기 위해, tank 및 valve 컴포넌트들을 인터페이스 변수를 갖는 능동 객체 소프트웨어 IC로 표현한 후, 이들을 그림 3의 다이어그램과 같이 연결만 시켜주면 능동적이면서 병행적으로 동작하는 분산 Tank 제어 시스템을 얻을 수 있다.

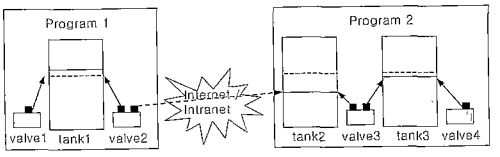

그림 3 분산 Tank 제어 시스템의 구성도

valve는 인터페이스 변수를 통해 tank의 상태 값인 level과 자신의 상태 값인 state를 모니터링하는 능동 객체이다. 이때, tank의 level이 안전 수위 이상이 되면, valve는 능동적으로 자신의 상태 값인 state를 closed에서 open으로 변화시키고 자신을 열어 *tank*의 level을 조절하게 된다. tank도 자기 자신의 상태 값을 관찰하는 능동 객체이다. valve의 열림과 닫힘에 의해 level 값에 변화가 오면, 능동적으로 자신의 디스플레이를 갱신한다. valve2와 tank2의 네트워크를 통한 연결은 프로그래머에게는 논리적인 직접 연결이다.

분산 Tank 제어 시스템은 능동 객체인 tank 및 valve들과 이들간의 연결로 구성되며 작성 과정은 메인에서, 다이어그램을 그대로 반영하는 능동 객체들을 선언하고 이들을 인터페이스 변수들로 연결시켜 주기만 하면 된다. 메인에서, 각 객체들을 제어하는 문장(메시지 전달 문장)들이 없어도, 각 구성 객체들이 능동 객체들이기 때문에 적절히 동작하는 시스템을 얻게 된다.

그림 3의 분산 Tank 제어 시스템의 메인인 DAOS Main은 그림 4와 같다. DAOSMain은, 능동 객체의 행위 및 그들간의 연결을 명시하기 위한 DAOSDL로 작성한다.

```
// 프로그램 1의 DAOSMain
public class DAOSMain TankSystem1 {
  DAOSbind("128.134.11.226");
  // 분산 프로세스를 관리하는 분산 스케쥴러에 연결
  Valve valve1 with {
    inTank = null;
    // 인터페이스 변수로 valve1에서 tank1으로 연결
    outTank = tank1;
  }
  Tank   tank1;              // tank1 선언
  DValve valve2 with {
    inTank = tank1;
```

```
    // 인터페이스 변수로 valve2에서 tank1으로 연결
    outDtank = tank2;
    // 분산 인터페이스 변수로 vavle2에서 tank2로 연결
  }
}

public class DAOSMain TankSystem2 {
  DAOSbind("128.134.11.226");
  // 분산 프로세스를 관리하는 분산 스케줄러에 연결
  DTank tank2;                    // tank2 선언
  Valve  valve3 with {
      inDtank = tank2;
      // 분산 인터페이스 변수로 valve3에서 tank2로 연결
      outTank = tank3;
      // 인터페이스 변수로 valve3에서 tank2로 연결
    }
    Tank   tank3;                 // tank3 선언
    Valve  valve4 with {
      inTank = tank3;
      // 인터페이스 변수로 valve4에서 tank3로 연결
       outTank = null;
    }
}
```

그림 4 분산 Tank 제어 시스템의 DAOSMain

그림 3의 프로그램 1과 프로그램 2는 각각 TankSystem1과 TankSystem2에 해당한다.

TankSystem1의 메인에서는, 분산 객체들의 능동 행위를 지원하기 위한 DAOS의 스케줄러가 위치할 사이트에 함수 DAOSbind를 이용해 연결한다. 능동 객체 valve1과 valve2는 선언과 함께, with절로 인터페이스 변수인 inTank와 outTank를 통해 tank1과 연결된다.

TankSystem2의 메인도 비슷하게 완성된다. 각 메인은 객체 선언과 그들간의 연결로 완성되며 실행 문장이 없어도, 구성 객체들이 능동 객체들이기 때문에 수행 가능한 시스템을 얻게 된다. 이와 같이, DAOS 방식의 코드인 그림 4는 그림 3의 시스템 다이어그램을 그대로 반영한다. 즉, 이러한 인터페이스 변수를 통한 객체 및 응용 프로그램간의 조립식 연결이 구성적 인터페이스를 이용한 객체 및 응용 프로그램간의 구성적 조립이다.

2.3 분산 능동 객체 시스템 정의 언어(DAOS Description Language)

본 절에서는 DAOS가 지원하는 세 가지 객체 트리거 방식과, 각각의 방식을 명시할 수 있는, 능동 객체 시스템 정의 언어(DAOS Description Language: DAOSDL)와 능동 객체 시스템 인터페이스 정의 언어 (DAOS/Interface Definition Language: DAOS/IDL)를 소개한다.

표 1에 기존의 객체지향 기술과, 이에 객체의 능동성과 조립성 기능을 추가해서 확장한 DAOS의 객체 트리거 방식을 비교하였다. 기존의 객체지향 기술에서는, 동기적인 (synchronous) 이벤트 호출(메시지 전달, event- driven)에 의해 타 객체를 기동시키는 동기적 이벤트 호출만을 지원한다. 그러나 DAOS 방식은 동기적 이벤트 호출뿐만

표 1 기존 객체지향 기술과 DAOS의 객체 트리거 방식

	기존 객체지향	DAOS 방식
Event-Driven Synchronous Call	O	O
Event-Driven AsynchronousCall	×	O
State-Driven Synchronous Call	×	O

아니라, 비동기적(asynchronous) 이벤트 호출도 지원한다. 비동기적 이벤트 호출이란 일정 시간 후, 원하는 객체를 기동시키는 기능을 말한다. 또한, DAOS 방식은 관찰 대상 객체인 타 객체 또는 자신의 상태 변화에 의해 스스로 기동할 수 있는 상태 기반(state-driven) 전이문도 지원한다[12].

객체의 능동성과 객체간의 구성적 조립을 명시하기 위해, Java를 확장한 DAOSDL의 확장된 문장들은 표 2와 같다. DAOSDL은 상태 기반 전이 문장과 이벤트 기반 전이 문장으로 구성된다. 표 2에 객체의 선언 및 조립을 명시하기 위한 DAOSMain 클래스의 키워드들도 함께 보였다.

표 2 분산 능동 객체 시스템 정의 언어(DAOSDL)

기반상태	전이 규칙	transition *name* when(*condition-part*) {*execution-part*}	*conditon-part*에 condition variable을 포함한 조건을 명시.
		조건이 만족되면 *execution-part* 수행	
	동등 배정	always *name* {*always-execution*}	*always-execution*에 condition variable을 포함한 배정문을 명시
		ondition variable 값의 변화에무조건 배정문 수행	
이벤트	지연 호출	*function-call* after *delay*	*delay*만큼 지연 후, *function-call* 수행
	지연 배정	*expression* after *delay*	*delay*만큼 지연 후, *expression* 수행
DAOSMain 클래스		DAOSMain{ (*Dname* with {*connections*} }	분산객체의 인터페이스 변수 연결
		name with {*connections*})+ }	로컬객체의 인터페이스 변수 연결
		분산 객체 및 로컬 객체 선언. with 정의 *conditions*에서 인터페이스 변수를 통한 연결 명시	

능동 객체는 관찰 대상 객체의 상태 변화에 따라 능동적으로 정해진 일을 수행한다. 이를 명시하기 위한 상태 기반 전이 문장은 전이 규칙문과 동등 배정문으로 나누어진다. 전이 규칙문에서, 이름을 *name*에 명시하고 조건 변수(condition variable)를 포함한 조건을 *condition -part*에 명시한다. 조건이 만족될 때, 수행할 행위는 실행문들로 구성되는 *execution-part*에 기술한다. 타 객체

의 상태 변화(조건 변수 값의 변화)에 의해 무조건 실행되는 동등 배정문은 전이 규칙문의 간략형이다. 동등 배정문에서, 이름을 *name*에 명시하고 조건 변수를 포함하는 배정문을 *always-execution*에 기술한다. 배정문 내의 조건 변수의 값이 변하면 무조건 그 문장이 수행된다.

이벤트 기반 전이 문장의 지연 호출문과 지연 배정문은 비동기적 이벤트 호출을 지원하며, 동기적 이벤트 호출문은 Java의 메시지 전달 문법과 같다. 지연 호출문은 *delay*에 원하는 지연 시간을 명시하고, 호출할 메소드를 *function-call*에 명시한다. 지연 배정문은 *delay*에 지연 시간을 설정하고, 수행 할 배정을 *expression*에 명시한다.

DAOSMain문에서는 필요한 능동 객체들을 생성하고, 이들을 인터페이스 변수들을 이용하여 **with**절 내에서 연결하여 원하는 시스템을 구성한다. 리모트 객체인 경우, 능동 객체의 이름(*name*)앞에 **D** 키워드를 붙여주고, 로컬 객체인 경우, 능동 객체의 이름을 그대로 명시한다. **with**절 내의 *connections*에서, 생성된 능동 객체가 모니터링할 타 객체 및 자신의 조건 변수를 인터페이스 변수로 연결한다.

분산된 능동 객체인 경우, 그 인터페이스를 IDL(Interface Definition Language)로 선언해야 한다. DAOS 방식에서도, 모니터링하고 모니터링 되는 객체들이 분산되어 있을 때, CORBA/IDL을 확장한 DAOS/IDL로 그 인터페이스를 선언해야 한다. DAOS/IDL의 확장된 문장들은 표 3과 같다.

DAOSDL로 구현될 전이 규칙문과 동등 배정문의 이름을 *name*에, 조건 변수를 *cond-var*에 명시한다. 지연 호출은 메소드 이름만 **daosFA** 키워드와 함께 *function-name*에 명시한 후, 그 메소드에 대한 CORBA IDL 정의를 한다. 지연 배정문은 분산 객체간의 수행이 아니므로 상응하는 DAOS/IDL문이 없다.

표 3 분산 능동 객체 시스템 인터페이스 정의 언어 (DAOS/IDL)

전이 규칙	**daosT** *name* (*cond-var*)	*cond-var*에 조건 변수를 명시
동등 배정	**daosA** *name* (*cond-var*)	*cond-var*에 조건 변수를 명시
지연 호출	**daosFA** *function-name*	*function-name*에 함수 이름을 명시

3. DAOS의 구현

본 절에서는 2.2절에서 소개한 분산 Tank 제어 시스템을 DAOS 방식으로 작성하는 과정을 보이고 그 동작 원리를 설명함으로써, DAOS 방식의 능동 객체 및 구성적 인터페이스에 대한 구현을 설명한다. 또한, DAOS 라이브러리의 주요 클래스들을 소개한다.

3.1 응용 프로그램의 작성 과정

DAOS 방식으로 그림 3의 분산 Tank 제어 시스템을 구성적으로 작성하는 과정은 그림 5와 같다[2]. 그림에서, 순차적인 각 작성 과정에 번호를 부여했으며 각 번호에 해당하는 설명은 다음과 같다.

(1) 그림 3에서, valve2와 tank2는 분산 환경에서 인터페이스 변수로 연결될 객체들이므로, 그 인터페이스 정의인 DTank와 DValve를 DAOS/IDL 파일인 tanksys.daos에 그림 6과 같이 정의한다.

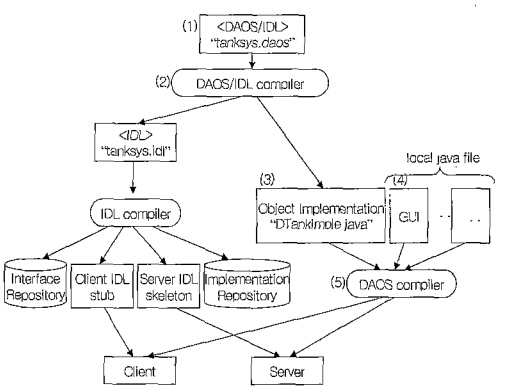

그림 5 DAOS 방식을 이용한 분산 응용 프로그램의 작성

DTank와 DValve는 DAOS가 제공하는 DAO 인터페이스를 상속받아 정의한다. DTank의 정의에서, DA Double형의 변수 dLevel은, 리모트 상에 존재하는 능동 객체에 의해서 모니터링 되는 조건 변수이다. 이 조건 변수 값이 변하면 인터페이스 변수를 통해 이 값을 모니터링하는 객체가 그 변화를 감지한다. safetyLevel은 CORBA IDL의 변수이다. setLevel은 과정 (3)에서, tank의 수위가 변하면 능동적으로 수위를 스스로 조절하는 전이 규칙문으로 구현될 메소드 시그니춰이다. **daosT** 키워드와 함께 선언되며 수위를 나타내는 상태 변수 dLevel을 파라미터로 명시한다.

DValve의 정의에서, ABoolean형의 변수 state는, 같은 사이트 상에 존재하는 능동 객체에 의해 모니터링 되는 조건 변수이다. opening과 closed는 모니터링 할 tank의 상태가 조건을 만족하면 valve를 열고 닫는 전이 규칙문으로 구현될 메소드 시그니춰들이다.

(2) 그림 6의 tanksys.daos를 DAOS/IDL 컴파일러로 처리하면, 그림 7의 tanksys.idl 파일과 그림 8과 같이, 인터페이스에 대해 자동 생성된 클래스인 DTankImpl.java 및 DVavleImpl.java 파일을 얻는다. tanksys.idl 파일의 DTank 인터페이스와 DValve 인터페이스는, tanksys.daos 파일에서 선언한 조건 변수와 CORBA IDL 변수 및 메소드만을 포함하고, 이들은 CORBA IDL 컴파일러에 의해 처리되어 Stub과 Skeleton 등을 생성한다. tanksys.daos에서 선언한 전이 문장들의 시그니춰는, DAOSDL의 문법에 따라 변환되어, 그 골격이 해당 클래스인 DTankImpl.java와 DValveImpl.java에 포함된다.

```
// TankSystem.daos
interface DTank : DAO {                 // DAO인 tank2를 정의할 인터페이스
  attribute DADouble dLevel;            // 분산 조건변수 선언
  attribute double safetyLevel;         // CORBA IDL 변수
  daosT setLevel (DADoble dLevel);      // 전이 규칙문으로 구현될 메소드
};

interface DValve : DAO {                // DAO인 valve2를 정의할 인터페이스
  attribute ABoolean state;             // 로컬 조건변수 선언
  daosT opening (ADouble level);        // 전이 규칙문으로 구현될 메소드
  daosT closed (ADouble level);         // 전이 규칙문으로 구현될 메소드
};
```

그림 6 tanksys.daos

```
//Tanksystem.idl
#include"c:₩DAOSystem₩DAOS₩daos.idl"
// DAOS의 DAO클래스를 상속받기 위해 daos.idl을 include함
#include ···
interface DTank : DAOSystem::DAOS::DAO {
  attribute DAOSystem::DAOS::DADouble dLevel;
  attribute double safetyLevel;
};
interface DValve : DAOSystem::DAOS::DAO {
     attribute DAOSystem::DAOS::ABoolean state;
};
```

그림 7 tanksys.idl

(3) 그림 8의 DTankImpl.java와 DValveImpl.java 파일의 setLevel, opening, closed의 전이 문장 골격을 완전한 DAOSDL 문장으로 그림 9와 같이 구현한다. setLevel은 자신의 상태 값인 dLevel이 이전 값과 다르면 트리거되는 전이 규칙문으로서, dLevel 값을 새로운 값으로 바꾸고 repaint 메소드를 호출해서 자신의 디스플레이를 갱신한다. 전이 규칙 opening은 인터페이스 변수 inTank와 outTank를 통해 valve의 왼쪽과 오른쪽에 위치한 tank들의 조건 변수 level과 dLevel의 값을 모니터링한다. 이때, 왼쪽 tank의 수위가 safetyLevel 이상이고 오른쪽 tank의 수위가 그 이하면, valve는 state를 false에서 true로 변환하고 왼쪽 tank의 액체를 빼서 오른쪽 tank로 보낼 수 있

```
// DTankImpl.java
class DTankImpl extends DAO
                implements _DTankOperations {
  transition setLevel when ( ) { }   // 전이 규칙문의 시그니춰
}

// DValveImpl.java
class DValveImpl extends DAO
                implements _DValveOperations {
  transition opening when ( ) { }    // 전이 규칙문의 시그니춰
  transition closed when ( ) { }     // 전이 규칙문의 시그니춰
}
```

그림 8 DTankImpl.java와 DValveImpl.java

```
// DTankimpl.java
class DTankImpl extends DAO
                implements _DTankOperations {
  . . .
  transition setLevel
    when (dLevel().val() != newLevel) {
      dLevel().val(newLevel);
      repaint( );
    }                          // 전이문장 구현
  DADouble dLevel()       { return this.dLevel; }
  void dLevel(double dLevel) { this.dLevel(dLevel); }
  DABoolean dState()      { return this.dState; }
  void dState(boolean dState) { this.dState(dState); }
};
// DValveImpl.java
class DValveImpl extends DAO
              implements _DValveOperations {
  transition opening
    when ((inTank.level.val()>= inTank.safetyLevel.val())
    &&(outTank.dLevel.val()<=out Tank.saftyLevel. val()) {
      state().reverse();
      inTank level.decrease();
      outTank.dLevel.increase();
    }                          // 전이 문장 구현
  transition closed
    when(outTank.dLevel.val()>=outTank.safetyLevel.val()) {
      state().reverse();
    }                          // 전이 문장 구현
};
```

그림 9 DTankImpl.java와 DVavleImpl.java의 구현

도록 level 값을 감소시키고 dLevel 값을 증가시킨다. 전이 규칙문 closed는 인터페이스 변수 outTank를 통해 오른쪽에 있는 tank의 수위가 안전 수위 이상이면 valve의 state를 true에서 false로 변환한다.

(4) 그림 3의 valve1, tank1, valve3, tank3, valve4와 같이 로컬 상의 능동 객체들에 대한 클래스 정의는 Tank.java와 Vavle.java 파일에 그림 10과 같이 SAO 클래스를 상속받아 구현한다. Tank와 Valve 클래스의 전이 문장 정의는 그림 9의 DTankImpl과 DValveImpl 클래스의 전이 문장 정의와 같다. 또한, 그림 4에 보인 DAOSMain.java 파일과 GUI 등에 필요한 파일들을 구현한다.

(5) 구현된 모든 Java 파일을 DAOS 컴파일러로 컴파일한다. 이때, 전이 문장들은 CORBA의 일반 메소드들로

변환된다.

```
// Tank.java
class Tank extends SAO {
  ADouble level;
  transition setLevel
    when (level.val() != newLevel) {
       level.val(newLevel);
       repaint();
    }                                  // 전이 규칙문 구현
}
// Valve.java
class Valve extends SAO {
  ABoolean state;
  transition opening
    when ((inTank.level.val()>inTank.safetyLevel.val())
    &&(outTank.level.val()<outTank.saftyLevel. val()) {
       inTank.level.decrease();
       outTank.level.increase();
       state.reverse();
    }                                  // 전이 규칙문 구현
    transition closed
      when (outTank.level.val()>outTank.safetyLevel.val()) {
         state.reverse();
    }                                  // 전이 규칙문 구현
}
```

그림 10 Tank.java와 Valve의 구현

3.2 응용 프로그램의 동작 원리

DAOS 방식으로 구현된 그림 3의 분산 Tank 제어 시스템의 동작 원리는 그림 11과 같다. 능동 객체에 선언된 각 조건 변수마다, 그 조건 변수를 사용하는 전이 문장들을 지정하는(포인팅하는) TE(Trigger Element)라는 자료구조가 생성된다. 또한, 각 응용 프로그램은 상태 기반 전이 문장의 처리를 위한 ATL(Action Trigger List)과 지연 호출 및 지연 배정문의 처리를 위한 FEL(Future Event List)이라는 큐를 하나씩 가진다.

ATL은, 조건 변수 값이 변하고 그 값이 전이 문장의 조건에 만족되면, 그 조건 변수의 TE가 ATE(Activated Trigger Element)를 생성하여 삽입하는 큐이고, FEL은 시간의 변화에 따라 TE가 FEE(Future Event Element)를 생성하여 삽입하는 큐이다. 삽입된 ATE 및 FEE들은 로컬 Scheduler에 의해 순차적으로 처리된다.

분산 객체들 간의 능동 행위는 DATL(Distributed ATL)과 DFEL(Distributed FEL)에 의해 지원된다. DATL은 모니터링 되는 객체의 조건 변수의 값이 변하고 그 값이 전이 문장의 조건에 맞으면, 그 조건 변수의 TE가, ATL을 생성하여 속한 응용 프로그램의 ATL에 삽입함과 동시에, 그 응용 프로그램에 대한 정보를 가진

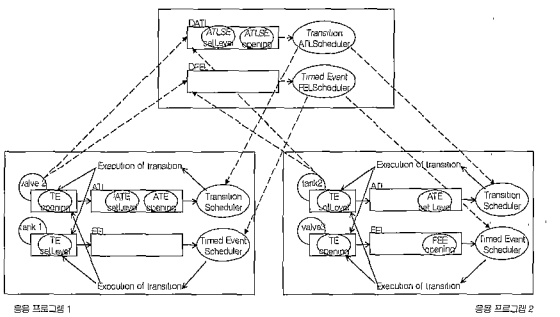

그림 11 DAOS 기반 분산 Tank 제어 시스템의 동작 원리

ATLSE(ATL Scheduler Element)를 생성하여 삽입하는 큐이다. 삽입된 ATLSE들은 응용 프로그램의 로컬 Scheduler에 의해 해당 ATE가 처리될 때, ATLScheduler에 의해 연계되어 순차적으로 처리된다. DFEL도 DATL과 비슷하게 동작한다.

즉, 전체 응용 프로그램이 실행될 때, 로컬상의 객체를 모니터링 하는 능동 객체의 ATE들은 로컬상의 ATL에서 즉시 실행되고, 리모트상의 객체를 모니터링 하는 능동 객체의 ATE들은 해당 ATL에서 대기하다가 DATL의 실행 명령에 의해 수행된다.

그림 3의 분산 Tank 제어 시스템인 경우, 각 tank의 조건 변수 level의 TE는 그림 9의 전이 문장 opening과 closed를 지정하며, 또한 그림 10의 전이 문장 setLevel도 지정한다. 설명을 위해, tank1의 level은 safetyLevel 보다 높고, tank2의 level은 낮다고 가정한다. 능동 객체 valve2는 인터페이스 변수인 inTank와 outTank를 통해 모니터링하는 tank1과 tank2의 level을 체크한다. 이때, 조건 변수 level의 값이 전이 규칙문 opening의 조건에 만족되므로 level의 TE는 ATE opening을 생성하여 응용 프로그램 1의 ATL에 삽입과 동시에 ATLSE opening을 생성하여 DATL에도 삽입한다.

ATLScheduler와 응용 프로그램 1의 Scheduler는 삽입된 ATLSE opening과 ATE opening을 DATL과 ATL에서 가져다 지정된 전이 규칙문인 opening을 연계하여 수행한다. 이 전이 규칙문 opening에 의해 valve의 상태가 closed에서 open으로 변화하고 이로 인해 양쪽 tank의 level이 변하게 된다. 이때, level의 변화에 의해 level의 TE는 ATE setLevel을 생성하여 응용 프로그램 1의 ATL에 삽입한다. tank1은 응용 프로그램 1 내에서 스스로를 모니터링 하는 능동 객체이므로, ATLSE를 생성하여 DATL에 삽입하지는 않는다.

tank2도 tank1과 같이 자신의 level 값의 변화를 감지

하게 되고 tank2의 level의 TE가 ATE setLevel를 생성하여 응용 프로그램 2의 ATL에 삽입함과 동시에 ATLSE setLevel을 생성하여 DATL에도 삽입한다. 삽입된 ATE 및 ATLSE들은 리모트 상의 ATLScheduler와 각 응용 프로그램의 로컬 Scheduler들에 의해 연계 처리된다. 처리 과정에서 발생되는 조건 변수의 변화에 따라 TE들은 다시 ATE, ATLSE들을 생성하여 ATL, DATL에 삽입하고 이와 같은 처리 과정이 반복되어 valve와 tank의 능동 행위가 지원된다.

네트워크 상의 2대의 펜티엄(Windows98, NT)과 OrbixWeb 3.1 및 Java 1.2 환경에서 실행한 분산 Tank 제어 시스템의 실행 화면은 그림 12와 같다. tank의 전체

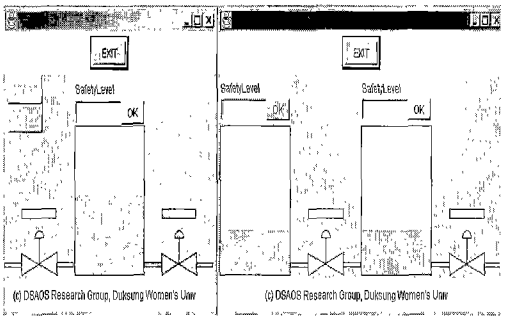

그림 12 분산 Tank 시스템의 실행 화면

level을 1, safetyLevel을 0.5로 가정했다. tank1의 level은 0.5 이상이고 tank2는 여유가 있기 때문에 valve2는 이를 감지하고 스스로의 상태를 closed에서 open으로 바꾸어, 두 tank의 수위를 조정한다. 각 tank는 자신의 수위 변화를 감지하고 이를 반영하는 그래픽 작업을 수행한다.

3.3 DAOS 라이브러리

그림 13은 DAOS 라이브러리의 계층도이다.

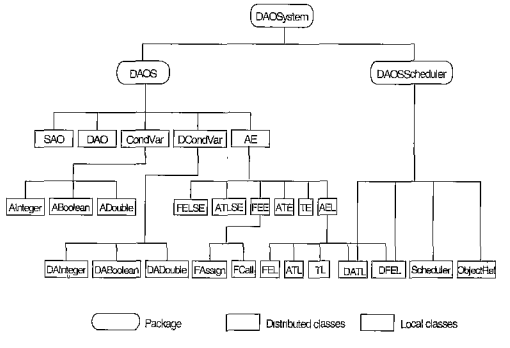

그림 13 DAOS 라이브러리의 계층도

객체들의 능동 행위를 지원하는 DAOSystem 패키지는 DAOS 패키지와 DAOSScheduler 패키지로 구성된다. 그림에서, 분산 및 로컬 클래스는 각각 리모트 및 로컬 상의 타 객체를 모니터링하고 능동 행위를 수행하는 기능을 구현한 클래스들이다.

3.3.1 DAOS 패키지

DAOS 패키지는 표 1의 세 가지 객체 트리거 방식을 지원하고, 시스템을 구성하는 객체에게 능동성을 부여하며, SAO(Structural Active Object), DAO(Distributed Active Object), CondVar(Condition Variable), DCondVar(Distributed CondVar), AE(Action Element) 등의 클래스들로 구성된다.

SAO 및 DAO는 로컬 및 리모트 상의 타 객체를 모니터링하고 그에 따라 행위를 수행하는 기능을 구현한 클래스이다. 예를 들어, 그림 3의 분산 Tank 제어 시스템에서, 인터페이스 변수를 통해 자신이나 타 객체를 모니터링 하는 객체인 tank1, tank2, tank3, valve1, valve2, valve3, valve4는 모니터링 하는 대상이 로컬 또는 리모트 상의 객체인 지에 따라 SAO 또는 DAO 클래스의 하위 클래스 객체들로 정의된다.

CondVar과 DCondVar은 조건 변수의 행위를 구현한 클래스들이다. 저장하고 있는 값이 변화하고 그 값이 전이 문장의 조건에 만족되면 배정된 TE로부터 ATE나 ATLSE를 생성한 후, ATL이나 DATL에 삽입하는 기능을 수행한다. 그림 6의 tanksys.daos 파일에서 DTank 인터페이스의 DADouble형과 DVavle 인터페이스의 ADouble형은 각각 DCondVar과 CondVar 클래스의 하위 클래스이다.

AE(Active Element)는 기동될 전이 문장을 지정하는 기능을 구현한 클래스이다. ATE는 전이 규칙문이나 동등 배정문에 해당하는 함수를 지정하고, FEE는 지연 호출문과 지연 배정문에 해당하는 함수를 지정한다. ATLSE, FELSE는 리모트상에 존재하는 객체의 상태 변화를 모니터링 할 때, 생성된 ATE나 FEE가 어느 응용 프로그램에서 트리거 되었는지에 대한 정보를 추가하는 기능을 구현한 클래스이다.

TL(Trigger List)은 생성된 ATE가 삽입되어, Scheduler에 의한 처리를 대기하는 기능을 제공하는 클래스이다. ATL은 TE 중 전이 규칙문이나 동등 배정문을 지정하는 것들이 ATE로 변환되어 삽입되는 기능을 구현한 클래스이다. FEL은 지연 호출문과 지연 배정문을 지정하는 것들이 FEE로 변환되어 삽입되는 기능을 제공하는 클래스이다.

3.3.2 DAOSScheduler 패키지

DAOSScheduler는 ATL과 FEL에 저장된 ATE 또는 FEE들이 처리될 때, 리모트 상의 객체와 연계되어 처리되는 기능을 구현한 패키지이다. 즉, 각 응용 프로그램마다 로컬 Scheduler가 한 개씩 생성되며, 여러 응용 프로그램에 분산된 능동 객체가 시간 흐름에 맞게 순차적인 작업의 처리는 ATLScheduler와 FELScheduler에 의해 이루어진다.

분산 Tank 제어 시스템의 valve2와 같이, 리모트 상의 객체인 tank2를 모니터링 하는 객체의 능동적인 행위는 ATLScheduler, FELScheduler와 DATL, DFEL에 의해 지원된다. 즉, valve2와 같은 객체는 분산된 두 객체 사이에 연관된 수행을 필요로 하므로, TE가 ATE를 생성하여 ATL에 삽입할 때, 이와 동시에 ATLSE을 생성하여 DATL에도 삽입한다. 리모트 상의 tank2를 모니터링 하는 valve2의 ATE가 수행될 차례가 되면, DATL의 ATLSE 정보를 이용하여 valve2와 연관된 응용 프로그램 2의 ATL에 있는 ATE들이 따라서 처리된다. FELScheduler는 TE가 FEE로 변환되어 FEL에 삽입될 때, FELSE로 변환되어 DFEL에 삽입되는 것을 모델링한 클래스이다.

4. 결 론

본 논문에서는 분산 공정 제어 응용의 생산성 및 확장성 향상을 위해, 객체에 능동성을 부여함으로써, 시스템을 구성적으로 쉽고 빠르게 작성할 수 있는 분산 능동 객체 시스템(Distributed Active Object System: DAOS) 방식과 그 주요 구현 사항을 소개한다. DAOS 방식으로 구축된 공정 제어 시스템은 능동 객체들과 그들의 인터페이스 변수를 통한 상호 연결로 구성되며, 각 능동 객체는 자신의 상태 뿐 아니라 인터페이스 변수를 통해 타 객체의 상태까지 모니터링하고 그 상태 변화에 따라 스스로 행위를 수행할 수 있는 객체이다. 따라서, DAOS 방식은 능동적이면서 병행적으로 동작해야 하는 공정 제어 컴포넌트들을, 기존의 객체지향 기술에 비해 보다 완전하게 모델링할 수 있다. 또한, 이러한 능동 객체들 간의 상호작용을 위한 구성적 인터페이스는 기존 객체지향의 절차적 인터페이스에 비해 객체 및 응용 프로그램 단위의 조립성을 제공한다[1]. 공정 제어 응용을 위한 DAOS 방식과 기존 분산 객체 지향 기술과의 비교는 표 4와 같다.

즉, 구성적 인터페이스로 객체 조립성을 지원하여 기존 객체지향 기술에 비해 공정 제어 응용의 생산성 및 확장성을 개선하고, 제어까지 포함하는 능동 객체를 지원하여 소프트웨어 컴포넌트화를 촉진시키며 따라서 소프트웨어의 유지보수 비용을 절감할 수 있는 장점이 있다.

향후, 로봇, 컨베이어, 밸브, 탱크 등의 공정 제어 컴포넌트들을 DAOS 방식으로 모델링한 라이브러리를 구축하고 이들을 조립식으로 연결해서 시스템을 작성할 수 있는 그래픽 에디터를 구현한다면, 공정 제어 응용 프로그램의 개발 및 유지에 드는 비용을 크게 삭감할 수 있다. 또한, 공정 제어 응용과 유사한 공급망 관리 등과 같은 분산 병행 시스템(distributed concurrent systems)의 개발에도 DAOS 방식이 잘 적용될 것이다.

표 4 공정 제어 응용을 위한 기존 객체지향 기술과 DAOS 방식의 비교

	기존 분산 객체 지향	DAOS 방식
응용 분야	범용	공정 제어 및 병행 시스템
인터페이스	절차적	구성적
객체 조립성	+	+++
확장성	+	++

참 고 문 헌

[1] 음두헌, 분산 객체의 구성적 인터페이스, 한국정보과학회 소프트웨어공학회지, 12권 2호, pp. 15-24, 1999. 6.

[2] 음두헌외 3인, 분산 능동 객체 시스템(DAOS)의 설계, 한국정보과학회 춘계학술발표논문집, 26권 1호, pp. 551-553, 1999.

[3] Thomas E. Vollman and David C. Whybark, Manufacturing Planning and Control Systems, McGraw-Hill, 1997.

[4] Richard Monson-Haefel, Enterprise JavaBeans, O'Reilly & Associates, 1999.6.

[5] Ed Roman, Mastering Enterprise JavaBeans, Wiley, 1999.

[6] Don Box, Keith Brown, Tim Ewald, and Chris Sells, Effective COM, Addison-Wesley, 1999.

[7] Reaz Hoque, Programming JavaBeans 1.1, McGraw-Hill, 1998.

[8] Robert Orfali, Dan Harkey, and Jeri Edwards, The Essential Client/Server Survival Guide, John Wiley & Sons, 1998.

[9] Budd,T.Understanding object-oriented programming with JAVA, Addison-Wesley, 2000.

[10] Sean Baker, CORBA Distributed Objects Using Orbix, Addison-Wesley, 1998.

[11] Robert Orfali and Dan Harkey, Client/Server programming with JAVA and CORBA, Wiley, 1997.

[12] Eum, D. and Minoura, T. Structural active object systems for mixed-mode simulation, IEICE Trans. on Information and Systems, vol. E79-D, no. 6, pp. 855-865, June 1996.

[13] Eum, D and Minoura .T., Structural and Hierarchical Composition of Interactive Multimedia

Scenarios : 4th Interbational Conference on Multi-Media Modeling (MMM97), Singapore, 1997.11. pp. 297-302.
[14] S.Choi and T.Minoura, "User interface system based on active objects," Proc. 2nd Symp. on Environments and Tools for Ada, Jan. 1992.
[15] 음두헌, 분산 객체 조립기를 이용한 MVC 응용의 구성적 작성, 정보과학회 논문지(B), 26권 11호, pp. 1298-1305, 1999.11.

음 두 헌
1984년 2월 서강대학교 전자공학과(학사). 1987년 10월 오레곤 주립대학교 컴퓨터 공학과(석사). 1990년 10월 오레곤 대학교 컴퓨터 공학과(박사). 1991년 1월 ~ 1992년 2월 전자통신연구소 인공자지능연구실. 1992년 3월 ~ 현재 덕성여자대학교 전산학과 부교수. 1999년 9월 ~ 2000년 8월 오레곤 주립대학 전산학과. 관심분야는 객체지향 시스템, 컴포넌트 기반 시스템.

유 은 자
2000년 2월 덕성여자대학교 전산학과(학사). 현재 덕성여자대학교 전산·정보통신 대학원 재학중. 관심분야는 객체지향 시스템, 컴포넌트 기반 시스템.