분산 능동 객체 시스템(DAOS)의 구현

이미은, 유은자, °양유진, 음두헌
덕성여자대학교

Implementation of Distributed Active Object System(DAOS)

Mieun Lee, Eunja Yoo, °Yoojin Yang, Doohun Eum
Duksung Women's Univ

요 약

기존 객체지향 기술의 객체는 상태(data) 정보와 행위(behavior) 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 CORBA 기반의 분산 환경에서, 객체의 상태 정보와 행위 정보 뿐 아니라 객체 자신의 제어(control) 부분까지 캡슐화한 능동 객체(active object)를 지원하여 실객체를 더욱 자연스럽게 표현할 수 있다. 여기서, 자신의 제어란 자신의 상태뿐 아니라 인터페이스 변수(interface variable)로 연결된 타 객체의 상태까지 모니터링하고 그 상태 변화에 따라 스스로 행위를 수행하는 것을 말한다. 따라서 DAOS 방식은 프로그램의 메인에서 메시지 전달을 통한 각 분산 객체들의 제어를 기술하지 않고, DAOSMain 클래스에서 인터페이스 변수들을 사용하여 객체들을 구성적으로 조립하여 시스템을 구축한다. 즉, DAOS 방식은 객체 조립성을 지원하여 분산 소프트웨어의 생산성을 향상시키고, 제어까지 캡슐화된 능동 객체를 지원하여 컴포넌트의 재사용성을 개선한다.

1. 서론

최근 인터넷 이용의 증가에 따라, 서비스 제공 시스템의 개념이 단일 시스템에서 네트워크로 연결된 분산 시스템으로 확장되고 있다. 현재, 분산 시스템에 대한 생산성 향상이 요구되고 있고, 분산 환경의 모델링에 적합하고 재사용성이 뛰어난 객체지향 패러다임이 분산 시스템 개발에 활발히 적용되고 있다.

기존 객체지향 기술의 객체는 상태(data) 정보와 행위(behavior) 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) 방식[3]은 CORBA[2] 기반의 분산 환경에서, 객체의 상태 정보와 행위 정보 뿐 아니라 객체 자신의 제어(control) 부분까지 캡슐화한 능동 객체(active object)를 지원하여 실객체를 더욱 자연스럽게 표현할 수 있다. 능동 객체는 로컬 객체인 구성적[1] 능동 객체(Structural Active Object: SAO)[1]와 분산 객체인 분산 능동 객체(Distributed Active Object: DAO)로 구성된다. DAOS 방식을 기반으로 하는 분산 응용 프로그램은 능동 객체와 일반 객체로 구성되며 능동 객체는 자신의 상태 뿐 아니라 인터페이스 변수(interface variable)로 연결된 타 객체의 상태까지 모니터링하고 그 상태 변화에 따라 행위를 수행한다. 따라서 프로그램의 메인에서 각 분산 객체들의 제어를 기술하지 않고, DAOSMain 클래스에서 인터페이스 변수들을 사용하여 객체들을 구성적으로 조립하여 시스템을 구축한다. 즉, DAOS 방식은 객체 조립성을 지원하여 분산 소프트웨어의 생산성을 향상시키고, 제어까지 캡슐화된 능동 객체를 지원하여 컴포넌트의 재사용성을 개선한다.

이러한 DAOS 방식[3]은 실시간 모니터링 및 콘트롤 응용에 잘 적용될 수 있다. 본 논문에서는 분산 Tank 제어 시스템에 DAOS 방식을 적용한 예를 소개한다. 그림 1의 분산 Tank 제어 시스템은 2개의 valve 객체와 1개의 tank 객체를 가지는 분산 프로세스 1과, 2개의 valve 객체와 2개의 tank 객체를 가지는 분산 프로세스 2로 구성된다. valve는 화살표로 나타낸 인터페이스 변수를 통해 tank의 상태값(level)을 모니터링하다가 일정 수위가 되면 능동적으로 자신의 상태값(state)과 tank의 상태값(level)을 변화시키고, tank는 valve에 의해 변화된 상태값(level)을 반영하므로, 이들을 능동 객체로 선언한 후, 인터페이스 변수를 이용하여 연결한다. 이때, 로컬상에서 연결되는 객체는 SAO로, 분산으로 연결되는 객체는 DAO로 선언한다. valve와 tank는 인터페이스 변수로 연결된 객체를 모니터링하다가 상태값의 변화에 의해 능동적으로 행위하는 능동 객체이므로 전체 시스템에서 객체의 행위에 대한 제어를 메시지 전달을 통해 하지 않아도 각자의 제어에 의해 전체 시스템이 동작한다

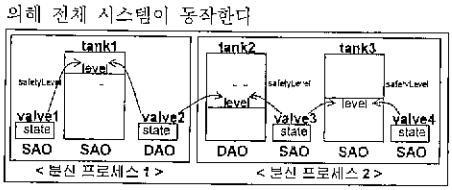

그림 1. 분산 Tank 제어 시스템 구성도

2절에서는 DAOS 방식의 개요와 DAOS 방식이 지원하는 구성적

1) '전체'를 창조해 내기 위해 여러 요소를 결합, 배치하여 하나의 예술작품을 성립시키는 방법. 모아서 조립한다는 의미

인터페이스, 그리고 분산 Tank 제어 시스템 응용 프로그램을 구성적으로 작성하는 과정을 설명한다. 3절에서는 DAOS 방식의 클래스 계층도와 동작 원리에 대해 설명하고 마지막 절에서는 DAOS 방식으로 분산 응용 프로그램을 작성할 때의 장점을 요약한다

2. 분산 능동 객체 시스템

본 절에서는 객체의 능동성과 분산 응용 프로그램의 구성적 작성을 지원하는 DAOS 방식과 이를 분산 응용 프로그램의 작성에 적용하는 과정을 설명한다.

2.1 DAOS(Distributed Active Object System) 방식

DAOS 방식이 지원하는 객체는 그림 2와 같이 상태 정보와 행위 정보 뿐만 아니라, 자신의 제어까지 포함하여 외부 조건이 만족되면 스스로 기동하는 능동 객체이다. SAO는 로컬상의 일반 Java 객체[4]에 제어에 해당하는 인터페이스를 포장한 능동 객체이고, DAO는 일반 CORBA[5] 객체에 제어에 해당하는 인터페이스를 포장한 분산 능동 객체이다 이러한 능동 객체는 인터페이스 변수를 통해 자신의 상태뿐만 아니라 타 객체의 상태를 모니터링하다가 조건을 판단하여 스스로 지정된 행위를 수행하는 능동성을 가진다

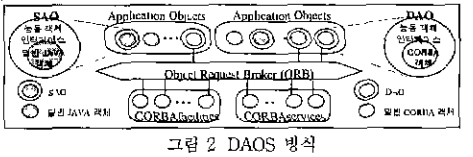

그림 2 DAOS 방식

기존의 객체지향 기술과 이를 개선한 DAOS 방식의 객체 트리거 방식은 표 1과 같다. DAOS 방식은 동기적인 이벤트 호출에 의해 타 객체에게 메시지를 전달하는 기존의 방법 뿐만 아니라, 비동기적 이벤트 호출에 의한 메시지 전달, 객체의 상태 변화에 의한 동기적·비동기적 메시지 전달도 지원한다 분산 응용 프로그램을 작성할 때, 기존의 객체 지향 방식대로 동기적 이벤트 호출만을 이용할 수도 있으나, DAOS 방식에서는 원하는 트리거 방식을 선택할 수 있으므로 객체들이 능동적으로 행위하는 실세계를 더욱 자연스럽게 모델링할 수 있다.

표 1. 기존 객체지향 기술과 DAOS 방식의 객체 트리거 방식

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

DAOS 방식에서, 객체의 능동성을 명시하기 위한 분산 능동 객체 시스템 정의 언어(DAOS Description Language : DAOSDL)의 전이 문장을 구성하는 상대 기반 전이 문장, 이벤트 기반 전이 문장과 시스템을 구성할 때 사용하는 키워드는 표 2와 같다.

표 2. 분산 능동 객체 시스템 정의 언어(DAOSDL)

상태 기반	전이규칙	transition *name* when *(조건부)* *{동작부}*	조건부에 명시한 상태정보 변경시, 조건부를 판단, 만족하면 동작부 실행
	동등매칭	Always *{동작부}*	조건부에 명시한 상태정보 변경시, 항상 동작부 실행
비 기 반 트	지연호출 지연매칭	*<function-call> after delay expression after delay*	delay 만큼 지연 후, *function-call* delay 만큼 지연 후, *expression* 수행
	일반호출	*object.method()*	일반 메소드 호출
시스템구성 키워드		DAOSMain{ *Dname* with *{인터페이스변수로 연결}* *name* with *{인터페이스변수로연결}*}	메인 키워드 분산 객체 선언, with절을 통한 연결 = DAO와 SAO와의 연결 = DAO와 DAO와의 연결 로컬 객체 선언, with절을 통한 연결 = SAO와 SAO, DAO와의 연결

DAOS 방식이 지원하는 SAO와 DAO는 스스로의 제어를 캡슐화한 모듈성이 뛰어난 소프트웨어 부품(IC)이며, 이는 설계도를 따라 구성적으로 조립하여 연결함으로써 신속하게 하드웨어 제품을 생산하는 규격화된 하드웨어 부품과 같은 개념이다. 인터페이스 변수는 하드웨어 IC의 핀과 같은 역할을 한다. 예를 들어, 프로그래머는 그림 1과 같이 설계한 객체 다이어그램대로 SAO와 DAO를 배치한 후, 인터페이스 변수를 이용하여 연결만 함으로써 손쉽게 분산 응용 프로그램을 작성할 수 있다. 이러한 능동 객체가 지원하는 인터페이스를 본 논문에서는 구성적 인터페이스(Structural Interface)[6]라 한다

2.2 DAOS 방식을 이용한 분산 응용 프로그램의 작성

DAOS 방식[3]을 이용하여 그림 1의 분산 Tank 시스템을 구성적으로 작성하는 과정은 그림 3과 같다

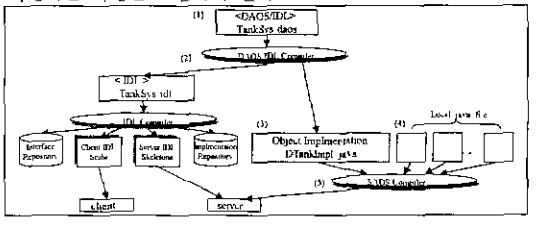

그림 3 DAOS 방식을 이용한 분산 응용 프로그램의 작성

(1) 그림 1의 각 프로세스를 구성하는 객체 중, 분산 환경에서 인터페이스 변수로 연결되는 두 객체인 valve2와 tank2는 DAOS/IDL인 TankSystem.daos에 그림 4와 같이 DAO로 정의한다.

```
interface DTank : DAO {
  attribute DADouble dLevel;
  attribute DABoolean dStatus;
  attribute double safetyLevel;
  oneway void setStatus(in boolean status);
  oneway void setLevel(in string name, in double newLevel) };
interface DValve : DAO { };
```

그림 4 Tanksystem.daos

(2) 그림 4의 TankSystem.daos를 DAOS/IDL 컴파일러로 컴파일하면, 그림 5와 같이 Tanksystem.idl 파일로 변환되고 정의한 interface는 ~Impl.java 파일로 자동 생성된다.

```
#include "c:\DAOSystem\DAOS\daos.idl"
#include
interface DTank : DAOSystem::DAOS::DAO {
  attribute DAOSystem::DAOS::DADouble dLevel;
  attribute DAOSystem::DAOS::DABoolean dStatus;
  attribute double safetyLevel;
  oneway void setStatus(  ),
};
interface DValve : DAOSystem::DAOS::DAO { };
```

그림 5 Tanksystem.idl

(3) 컴파일되어 자동 생성된 파일에 그림 6과 같이 DAOS/IDL에서 정의한 일반 메소드와 상대 변수에 따라 능동적으로 수행되는 행위를 건이 문장으로 구현한다

```
// DTankImpl.java
class DTankImpl extends DAO
      implements  DTankOperations {
  transition redraw
    when (dLevel.val( ) != oldLevel)
      { repaint( ); }
}     // 일반메소드 전이문장 구현

// DValveImpl.java
class DValveImpl extends DAO
      implements _DValveOperations {
  transition openingCheck
    ; when ( · ) { · }
    when ( · ) { · },
};    // 일반메소드, 전이문장 구현
```

그림 6 자동 생성 파일의 구현

(4) 로컬상에서 필요한 Java 파일을 구현하고, 그림 7과 같이 TankSystem1과 TankSystem2의 DAOSMain에서 시스템을 구성하는 객체들을 선언하고, with절 내에서 인터페이스 변수를 이용하여 모니터링하고자 하는 객체를 연결한다 그리고 분산 스케줄러가 위치할 IP Address를 선언한다. 이러한 인터페이스 변수를 통한 객체간의 연결이 DAOS 방식에서 지원하는 구성적 인터페이스를 통한 객체간의 조립이다

```
// 분산 프로세스1 메인
class DAOSMain TankSystem1 {
  DAOSbind("128.134.11.226")
                // 분산 스케줄러 IP Address
  sValve valve1 with
    { inTank = null, outTank = tank1; }
  Tank   tank1,
  DValve valve2 with
    { inTank = tank1, outDtank = tank2; }
}
```

```
// 분산 프로세스2 메인
class DAOSMain TankSystem2 {
  DAOSbind("128.134.11.225",
                // 분산 스케줄러 IP Address
  DTank tank2,
  Valve  valve3 with
    { inDtank = tank2, outTank = tank3; }
  Tank   tank3,
  Valve  valve4 with
    { inTank = tank3, outTank = null; }
}
```

그림 7 분산 프로세스 1, 2 시스템 구성

(5) 구현한 모든 Java 파일을 SAOS 컴파일러로 컴파일한다

3. DAOS 구현

본 절에서는 객체들의 능동 행위를 지원하는 DAOS의 클래스 계층도와 DAOS 방식으로 작성한 분산 응용 프로그램의 동작 원리를 설명한다.

3.1 DAOS의 클래스 계층도

그림 8은 DAOS를 구성하는 클래스의 계층도이다. DAOSystem은 SAO와 DAO의 능동적인 행위를 지원하며, DAOS와 DAOSScheduler로 구성된다.

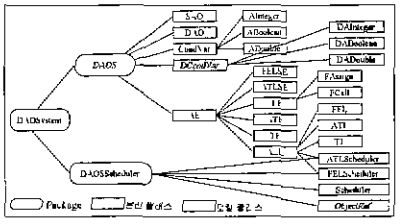

그림 8. DAOS의 클래스 계층도

DAOS는 SAO, DAO, CondVar, DCondVar, AE(Action Element) 클래스들로 구성된다. SAO와 DAO 클래스는 객체들의 능동적인 특성을 모델링하고, CondVar 클래스와 DCondVar 클래스는 SAO와 DAO의 상태를 모델링한다. AE 클래스는 객체들이 능동적으로 수행할 행위를 모델링하며, 이들은 프로그램 실행 중에 TE(Trigger Element), ATE(Activated Trigger Element), FEE(Future Event Element), ATLSE(Activated Trigger List Scheduler Element), FELSE(Future Element List Scheduler Element)로 변환된다. AEL(Action Element List) 클래스는 AE를 관리하는 시스템 리스트로, TL(Trigger List), ATL(Activated Trigger List), FEL(Future Event List)로 구성된다. TL에는 객체들의 능동 행위를 명시하는 전이 문장이 변환된 TE가 삽입되고, ATL에는 TE 중 전이 규칙이나 동등 매정문이 변환된 TE가 ATE로 변환되어 삽입되며, FEL에는 지연 호출과 지연 비정이 변환된 TE가 FEE로 변환되어 삽입된다. DAOSScheduler는 분산 프로세스를 스케쥴링하고 DAO를 관리하는 리스트인 ATLScheduler, FELScheduler 클래스들을 가진다.

3.2 DAOS 동작 원리

DAOS 방식에서 SAO와 DAO의 능동 행위는 DAOSDL의 전이 문장으로 표현한다. DAOS 방식으로 작성한 분산 응용 프로그램은 SAO와 DAO의 전이 문장으로 기술된 행위가 능동적으로 수행된다. 상태 변수(Condition Variable)의 변화에 의해 전이는 능동 객체의 상태를 변화시키고, 다른 사건들을 발생시킨다. 새로운 상태와 사건은 여러 개의 전이를 발생시킬 수 있으며, 다른 능동 객체의 전이 문장을 트리거시킬 수도 있다.

DAOS 방식으로 구현된 그림 1의 분산 Tank 제어 시스템의 실행 화면은 그림 9와 같고, 동작 원리는 그림 10과 같다. 그림 1과 같이 tank1의 level은 safetyLevel보다 높고, tank2는 낮다고 가정한다. 먼저 valve2는 모니터링하는 tank1과 tank2의 level을 체크한다. tank1의 level은 safetyLevel보다 높고 tank2의 level은 낮으므로, valve2는 자신의 state를 open으로 변화시킨다. 이때, valve2는 DAO이므로 전이 문장(opening)이 변환된 TE는 트리거 되어 ATL에는 ATE 형태로, ATLScheduler에는 ATLSE 형태로 변환되어 삽입된다. ATE, ATLSE가 TransitionATL Scheduler와 Transition Scheduler에 의해 수행될 때, valve2는 tank1의 level을 변화시킨다. tank1은 SAO이므로 자신의 level이 변화됨을 감지하고, 전이 문장(setLevel)이 변환된 TE를 트리거 하여 ATL에 삽입한다. 이와 병행하여 valve2가 tank2의 level을 변화시킨다. tank2의 동작은 tank1과 같다. 전체 응용 프로그램이 실행될 때, SAO는 로컬상의 ATL에서 즉시 실행되고, DAO는 실행될 차례가 되면 대기하다가 분산 스케줄러의 실행 명령을 받은 후 실행된다. ATLScheduler, FELScheduler는 ATLSE, FELSE를 순서대로 실행하여 각 프로세스에 존재하는 ATL과 FEL을 스케쥴링함으로써 서로 연관되게 순차적으로 수행시킨다.

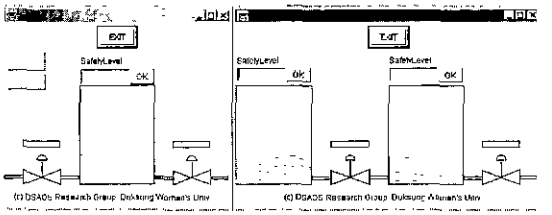

그림 9 분산 Tank 시스템의 실행화면

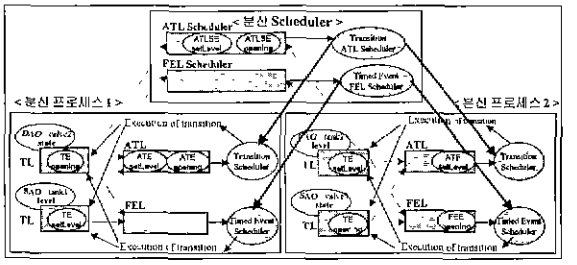

그림 10 DAOS의 동작 원리

4. 결론

본 논문에서는 분산 시스템의 생산성 향상을 위해, 분산 객체에 능동성을 부여함으로써, 시스템을 구성적으로 쉽고 빠르게 작성할 수 있는 분산 능동 객체 시스템(Distributed Active Object System: DAOS) 방식과 그 구현 원리를 소개한다. DAOS 방식을 기반으로 구축된 분산 시스템은 능동 객체들로 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. 또한 능동 객체는 상태와 행위 정보만 캡슐화한 기존 객체지향 기술의 객체에 비해 자신의 제어까지 캡슐화하여 응집력이 강하고, 연관성이 적어 소프트웨어 구성요소의 이식성을 높이고, 컴포넌트화를 촉진시킨다.

참고문헌

[1] 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.

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

[3] 음두헌외 3인, 분산 능동 객체 시스템(DAOS)의 설계, 한국정보과학회 춘계학술발표논문집, 제 26권 1호, pp 551-558, 1999.

[4] Budd, T., Understanding object-oriented programming with JAVA, Addison-Wesley, 1998.

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

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