CORBA 환경에서 분산 JavaBeans 컴포넌트 통합을 위한 연결자 설계 및 구현

정성옥* · 김재석**

Design and Implementation of Connector for Distributed JavaBeans Component Integration in the CORBA Environment

Sung-Ok Jung* · Jae-Seog Kim**

요 약

현재의 소프트웨어 아키텍처에 관한 연구는 컴포넌트 집합과 같은 소프트웨어 시스템을 구성하는 객체 또는 컴포넌트의 상호 동작 및 관련성을 보다 효과적으로 연결할 수 있는 다양한 기법이 제시되고 있다. 본 논문에서는 JavaBeans에 기반을 둔 분산 시스템 환경에서 객체와 객체간에 관련성을 모델링하기 위해 컴포넌트, 연결자 및 컴포넌트 스키마로 구성된 구조화된 모델을 제시하고 구현한다. 특히 JavaBeans에서 객체간의 관련성을 모델링하기 위한 연결자의 구성에 중점을 둔다. 본 연구에서 제시된 연결자 모델은 JavaBeans기반 분산 시스템 환경에서 다양한 객체간의 의존성을 명확하게 표현하는데 효과적이며 분산되어 있는 컴포넌트를 정형화된 방법으로 통합할 수 있는 효과를 가진다.

ABSTRACT

Current research for software architecture views and models a software system as a set of components and connectors. Components are abstractions of system level computational entities. Connectors are abstractions of components interrelationships. In this paper, we focus on connectors for the JavaBeans-based systems that are built using object integration technologies like CORBA. We present connector model in JavaBeans-based system for object-oriented component integration. We start with a discussion of related work of software architecture research and object-oriented modeling that focuses on the description of component collaborations. We propose connectors as transferable abstractions of system level component interconnection and inter-operation.

키워드

컴포넌트(Components), 연결자(Connectors), 자바빈즈 (JavaBeans)

* 광주여자대학교 인터넷정보학과

**조선대학교 컴퓨터공학과

접수일자 : 2002. 10. 17

Ⅰ. 서 론

객체지향 소프트웨어 시스템은 상호 연관되어 있는 수많은 객체의 집합으로 구성되며 객체지향 분석 및 설계 단계에서 객체에 대한 정보는 일반적으로 클래스 및 클래스간의 관계를 정형화된 기법을 적용하여 명확하게 기술한다[1].

OMG(Object Management Group)의 CORBA (Common Object Request Broker Architecture)에 기반을 둔 분산 객체 시스템에서도 서로 연관되고 상호 동작을 수행하는 수많은 객체의 집합으로 구성되어 있다. 하지만 CORBA에 기반을 둔 시스템에서는 객체간에 모델을 설정하고 설계를 하는데 있어 객체지향 기술에서 사용하였던 클래스에 기반을 둔 모델링 기술을 적용하는데 제약을 가진다. 왜냐하면, CORBA 객체를 기술하는데 있어 클래스 추상화를 이용하면 엄격한 인터페이스 부분과 구현 부분을 분리하기가 어렵기 때문에 충분하게 CORBA 객체의 특성을 기술할 수 없다. 따라서 추상화와 CORBA 객체간의 관련성을 일반적인 클래스 관련성을 이용하여 표현하는 데는 새로운 기법이 적용되어야 한다.

본 논문에서는 JavaBeans에 기반을 둔 분산 시스템 환경에서 객체와 객체간에 관련성을 모델링하기 위해 컴포넌트, 연결자(connector) 및 컴포넌트 스키마(scheme)로 구성된 구조화된 모델을 제시하고 구현한다. 특히 JavaBeans 환경에서 객체간의 관련성을 모델링하기 위한 연결자의 구성에 중점을 둔다.

Ⅱ. 소프트웨어 컴포넌트 아키텍처

1. 분산 컴포넌트 기술

최근까지 객체지향 프로그래밍 기술이 인기를 누려왔지만 최근에는 재사용 가능한 소프트웨어 컴포넌트를 조합하여 새로운 어플리케이션을 만드는 방법이 사용자의 요구에 따른 어플리케이션을 개발하는 데 생산성이 높은 것으로 인식되고 있다. 컴포넌트는 버튼과 같이 아주 작은 것에서부터 복잡한 컴포넌트를 비롯하여 워드프로세서나 스프레드시트와 같은 어플리케이션에 이르기까지 다양하다. 소프트웨어 컴포넌트는 사용자 눈에 보일 수도 있고 보이지 않을 수도 있다. 마이크로소프트의 Visual Basic은 윈도우 기반에서 어플리케이션을 개발하는 데 주로 사용되어 왔다. 또한 Borland의 Delphi는 보다 강력한 데이터 액세스 컴포넌트와 확장 기능을 추가함으로써 여러 가지 방법을 보완하였다. SUN의 JavaBeans는 컴포넌트 기반의 소프트웨어 패러다임을 플랫폼 독립적이고 네트워크를 인식할 수 있는 컴퓨팅으로 확장시키고 있다. JavaBeans는 Hot-Java, Netscape-Navigator 등의 다양한 컨테이너와 통합 도구 및 컴포넌트에서 재사용 할 수 있다. 컴포넌트 기반 소프트웨어(Component-based software)는 특정 아키텍처와 관련된 API(Application Program Interface)의 집합으로 구성되는 일종의 소프트웨어 모델을 기술하는 데 사용된다. 이 모델에서 소프트웨어 컴포넌트를 정의해 동적으로 결합해서 새로운 응용 어플리케이션을 만들어 낼 수 있다. 이러한 모델은 컴포넌트, 아키텍처, 컨테이너 및 스크립팅의 요소로 구성된다.

2. COM/DCOM

마이크로소프트사에서 제시한 COM/DCOM은 하나의 규약으로서 컴포넌트 소프트웨어를 구축하기 위한 이진 표준이다. 자동화, 영속성, 모니커와 통일 데이터 전송으로 구성되어 있고 마이크로소프트의 OLE 및 ActiveX의 기반 기술이다. 이러한 COM/DCOM은 마이크로소프트사의 윈도우 운영체제가 널리 사용되기 때문에 CORBA에 경쟁하는 의미로 COM/DCOM의 위치는 중요하다고 할 수 있다. COM은 마이크로소프트가 만든 컴포넌트 모델로서 컴포넌트의 모듈 형태와 컴포넌트를 사용하기 위한 방법을 표준화한 것이다. COM은 어떤 프로그램이나 시스템을 이루는 컴포넌트들이 상호 통신할 수 있도록 하는 메커니즘이라고 할 수 있다.

DCOM은 COM 어플리케이션이 원격 컴포넌트를 프록시나 스터브를 이용하여 마치 같은 기계 내에 있는 것처럼 사용하게 하는 것으로서 분산 응용 프로그램을 구성할 때 많은 장점을 갖는다. 따라서 DCOM은 복잡한 네트웍 프로그래밍에 대한 부담을 줄여주며 개발 시간을 단축시켜 준다. 또한 기본적으로 COM의 특성을 가지고 있으므로 재사용의 특성을 이용해 개발 기간 및 비용을 절감할 수 있다.

3. CORBA

CORBA는 OMG에서 정의한 분산 객체 컴퓨팅 환경의 표준으로서 이질적인 분산 환경에서 어플리케이션을 구현하는데 필요한 프레임워크, 객체 서비스, 객체 클래스 라이브러리 등이 정의되어 있다. OMG는 CORBA 외에 객체의 기본적인 조작과 복합 문서, 시스템 관리 등 일반적인 어플리케이션의 공통 기능인 인터페이스, 비즈니스 객체, 특정 분야의 고유 객체 등을 가지고 있으며 또 다른 어플리케이션을 포함한 OMA(Object Management Architecture)로서 이루어져 있다. OMA는 서로 이질적인 특징을 갖는 이 기종의 분산 시스템 환경에서 객체지향 기술을 기반으로 하여 어플리케이션들을 서로 통합할 수 있는 표준 프레임워크로서 코어 객체 모델과 참조 모델로 구분된다.

ORB 기능은 클라이언트로부터 요구를 전달받은 후 클라이언트의 요구를 실질적으로 구현해 줄 객체를 찾아서 매개변수를 전달하고 해당되는 메소드를 호출하며 다시 결과를 반환해 주는 역할을 수행한다. 이 때 클라이언트는 ORB를 이용하여 구현 객체의 메소드를 투명하고 명확하게 호출할 수 있다. CORBA는 ORB 구성 요소에 대한 인터페이스를 제공하고 인터페이스에 대한 구현은 실제로 ORB를 제작하는 개발자에게 맡긴다[2][3].

CORBA ORB는 그림 1과 같은 구조를 가지며 보통 IIOP(Internet Inter-ORB Protocol)를 사용하여 통신을 수행하는데 IIOP는 ORB간의 상호 통신을 위한 프로토콜인 GIOP(General Inter-ORB Protocol)를 인터넷 통신 프로토콜인 TCP/IP상에서 동작하도록 구현한 것이다. CORBA는 네트워크 프로토콜 독립적이어서 다른 네트워크 프로토콜을 이용하여 통신할 수도 있다[3].

그림 1의 ORB 구조에서 클라이언트는 연산을 수행하고자 하는 요소이고 객체의 구현은 실제로 객체를 구현하는 코드와 데이터로 구성된다.

클라이언트 IDL 스터브는 서버 객체에 대해 정적인 인터페이스를 제공하며 클라이언트가 서버상의 서비스를 호출하는 방법을 정의한다. CORBA 서비스는 분산 객체 시스템을 구축하는데 있어서 필요한 기본적인 서비스들의 집합이다. 객체 서비스는 이미 채택된 서비스와 현재 준비중인 서비스를 포함하여

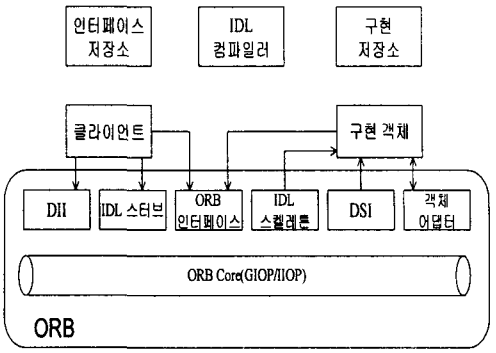

그림 1. CORBA ORB 구조
Fig. 1. Structure of the CORBA ORB

총 15개의 서비스로 구성되어 있으며 현재에도 계속 개발중이다. 서비스들은 분산 객체 기반 시스템을 위한 기본적인 설계 단위들로서 분산 컴포넌트를 생성하기 위해 이용되며 다른 어플리케이션 객체와 마찬가지로 ORB와 접속하여 클라이언트에게 서비스를 제공한다[2][6].

4. JavaBeans

JavaBeans는 소프트웨어 컴포넌트 개발을 위한 표준 API이다. 소프트웨어 컴포넌트란 마이크로소프트사의 Visual Basic 언어의 VBX나 볼렌드사의 델파이 컴포넌트와 같이 자동차의 부속품 같이 여러 응용 프로그램을 만드는데 사용될 수 있는 작은 단위의 소프트웨어를 말한다. 이러한 소프트웨어 컴포넌트는 재사용성을 높이려는 객체지향 프로그래밍에 잘 부합한다고 할 수 있다. JavaBeans 구조의 가장 큰 목적은 플랫폼 중립적인 컴포넌트 구조를 제공하는데 있다.

JavaBeans가 다른 JavaBeans내에 중첩되어 사용되더라도 모든 플랫폼에서 모든 기능을 지원한다. JavaBeans는 이러한 자바 환경을 바탕으로 더 좋은 개발 환경으로 확장하는 것이다. 현재의 자바 애플릿은 단순한 정적 컴포넌트 모델만을 제공하고 있다. 컴포넌트들은 서로 다른 시간에 서로 다른 개발자에 의해서 만들어 질 수도 있다. 즉, 컴포넌트들이 하나의 응용 프로그램 안에 합쳐질 필요가 없다는 것을 의미하며 그들간에는 동적인 통신이 이루어진다.

JavaBeans 컴포넌트 모델은 Java에서 지원하는 다양한 클래스에 기반을 두고 있다. 따라서 이러한

JavaBeans 컴포넌트 모델을 사용하기 위해서는 클래스가 도구화 되어야하고 재사용 등의 특성을 활용하기 위해서는 몇 가지 규칙을 따라야 한다. 또한 JavaBeans는 블랙박스 컴포넌트로서 소스 코드가 제공되지 않는 컴포넌트에 대해서는 사용자가 이러한 컴포넌트를 확장할 수 없는 특징을 가지고 있다. 따라서 사용자 입장에서 보면 JavaBeans는 그림 2와 같이 외부에 자신의 메소드, 속성(property) 그리고 생성하는 이벤트만을 보여준다. 또한 BeanInfo 클래스를 통해서 자기 자신에 대한 정보를 제공한다.

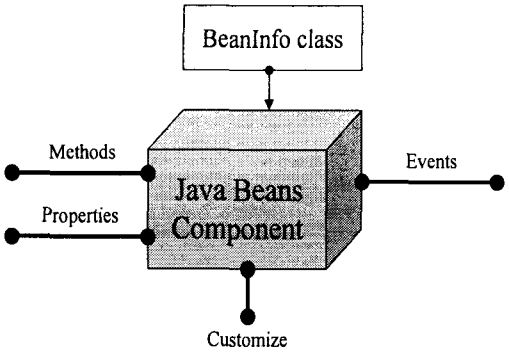

그림 2. JavaBeans 컴포넌트
Fig. 2. JavaBeans Component

JavaBeans 기술시에 이용되는 IDL은 지원되지 않고 있으며 JavaBeans의 인터페이스를 외부에 표현하기 위해서는 디자인 패턴 또는 명명법이라고 부르는 것을 이용하여 자신의 메소드, 이벤트, 속성 등을 그림 3과 같이 표현한다.

```
속성 디자인 패턴
public <PropertyType> getXXX() ;
public void setXXX( <PropertyType> value) ;
이벤트 디자인 패턴
public void addLottWinnderListener
(LottWinnderListener a);
public void removeLottWinnderListener
(LottWinnderListener a);
```

그림 3. JavaBeans 디자인 패턴
Fig. 3. JavaBeans Design Pattern

JavaBeans 클래스는 어플리케이션이나 또 다른 Java Beans을 생성할 경우에 이용되는 클래스로서 생성자인 bean()와 7개의 메소드를 기본적으로 제공한다.

또한 이러한 JavaBeans 클래스는 기본적으로 Java에서 지원하는 "java.lang"의 "Object Class"에서 생성되며 Instantiate(), getInstanceOf(), isInstanceOf(), isDesigntime(), isGuiAvailable(), 소드를 이용하여 그림 4와 같이 실제적으로 JasetDesignTime(), setGuiAvailable()와 같은 메소드를 지원한다.

```
public class bean extends Object
{
//생성자
public bean() ;
//클래스 메소드
public static Object Instantiate(ClassLoader cls,
     String beanName) throws IOException,
     ClassNotFoundException ;
public static Object getInstanceOf(Object bean,
                              Class targetType);
public static boolean isInstanceOf(Class bean,
                              Class targetType) ;
public static boolean isDesignTime() ;
public static boolean isGuiAvailable() ;
public static boolean SetDesignTime(boolean
     isDesignTime) throws SecurityException ;
public static boolean setGuiAvailable(boolean
     isDesignTime) throws SecurityException ;
}
```

그림 4. JavaBeans 클래스 생성 예
Fig. 4. Example of JavaBeans Class Generation

III. 컴포넌트 통합을 위한 연결자 설계

본 논문에서는 분산 시스템 환경을 기반으로 하여 객체 및 컴포넌트 사이의 관련성을 기술하고 상호 연관 동작을 갖는 컴포넌트를 연결하기 위해 그림 5와 같은 역할을 하는 연결자를 설계하고 구현한다.

또한 연결자를 이용하여 분산 시스템 환경에서 원활하게 컴포넌트간 연결 및 상호 작용을 지원하기 위하여 그림 6과 같이 컴포넌트, 컴포넌트 스키마, 연결자를 갖는 컴포넌트 통합 연결자 모델을 설계하였다.

컴포넌트는 클라이언트에게 정보를 제공하는 인터페이스 부분, 인터페이스에 대한 내부 구현 부분으로 연결해주는 표현-지도 부분, 실질적으로 컴포넌트의 행위를 기술한 표현 부분으로 구성하였다.

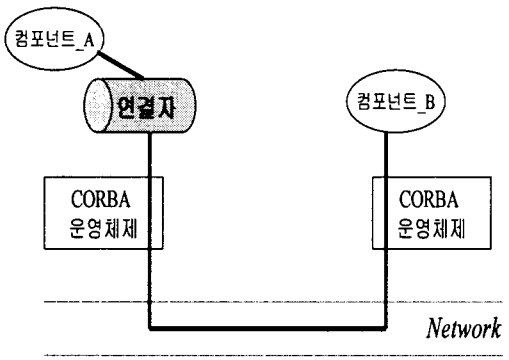

그림 5. 연결자의 역할

Fig. 5. Role of Connector

본 논문에서 JavaBeans 환경에서 컴포넌트 통합을 위한 연결자 모델링을 설계하고 구현하기 위해 소프트웨어 아키텍처, 아키텍처 기술 언어(architectural description language), 객체지향 모델링 기술, 분산 객체의 처리를 위한 참조 모델 등에 기반을 둔다.

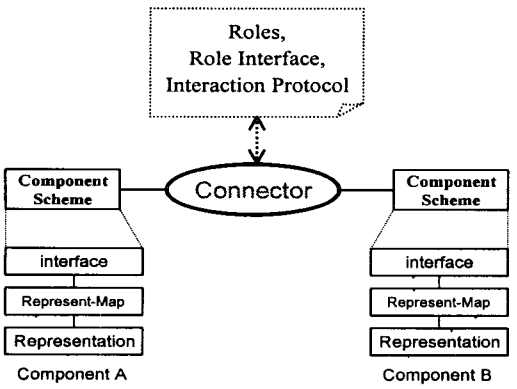

그림 6. 컴포넌트 통합 연결자 모델링

Fig. 6. Modeling of the Component Integration Connector

이를 위해 JavaBeans에 기반을 둔 소프트웨어 아키텍처를 모델링하며 그림 6과 같이 구성된 컴포넌트, 연결자 및 컴포넌트 스키마로 구성된 프레임워크를 설계하고 구현한다. 컴포넌트는 JavaBeans 환경에 존재하는 분산 객체의 집약적인 기술로서 인터페이스 집합인 인터페이스 명세 부분, 내부적인 객체지향 스키마인 표현(representation) 부분, 외부적인 객체지향 스키마인 표현-지도(represent-map)를 포함하고 있다.

연결자는 컴포넌트의 논리적인 관련성을 갖는 통합과 객체의 동적인 관계를 나타내는 컴포넌트의 상호작용을 추상화하며 연결자는 기능(roles), 기능 인터페이스(role interface), 상호 작용 프로토콜(interaction protocol)의 기술을 포함하고 있다. 또한 연결자는 컴포넌트 사이의 상호 동작을 위해 컴포넌트로부터 요구되는 행위의 형태를 기술한다. 컴포넌트 스키마는 그림 7과 같이 특정 컴포넌트를 위한 컴포넌트의 추상화를 번역한다.

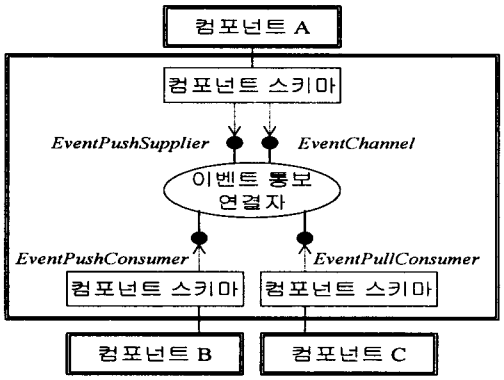

그림 7. 컴포넌트 스키마의 역할

Fig. 7. Role of the Component Scheme

IV. 컴포넌트 통합을 위한 연결자 구현

1. 이벤트통보 연결자 구현

객체의 동작이 동기화 된 수행의 결과로 메소드 호출이 발생되는 표준화된 CORBA 객체 통신 모델링 방법에 반대로, 이벤트는 다양한 객체사이에서 비동기화 된 방법으로 객체간에 통신을 한다. OMG의 이벤트 서비스는 표준화된 인터페이스와 publish/

subscribe에 기반을 둔 이벤트-참여를 위한 객체 상호 작용 모델을 지정한다.

하나 이상의 객체들은 이벤트 데이터를 발생시키고 이벤트 제공자(supplier)로서 동작한다. 또한 수많은 다른 객체들은 이벤트 데이터를 받고 이벤트 소비자(consumer)로서 동작한다. 본 논문에서 일반적인 이벤트 통신 방법과 반대가 되는 모든 이벤트 공급자는 동적으로 이벤트를 전송하는 push 형태이고, 모든 클라이언트는 이벤트의 전송을 기다리는 push 형태 또는 동적으로 이벤트 발생에 대한 질의를 하는 pull 형태를 갖는 이벤트 채널(channel)로서, 포괄 이벤트(generic event) 통신 방법으로 모델링된다.

연결자 EventNotification의 명세서에는 이벤트 통보를 집약적으로 수행하기 위해 객체사이의 상호 의존성을 기술하며 객체의 기능(roles), 기능 인터페이스, 상호 작용 프로토콜에 의해 구조화된 형태를 갖게된다. 기능(roles) 부분은 컴포넌트를 구성하기 위해 협력하는 참여자를 말한다. 이벤트 통보 연결자의 경우 "EventPushSupplier", "EventPushConsumer", "EventPullConsumer", "EventChannel"과 같은 4개의 객체 기능이 존재한다. 이러한 객체들은 확장된 수준에서 특정 컴포넌트에 의해 동작되어진다. 또한 각각의 컴포넌트는 서로 같거나 다른 연결자에 의해 하나 이상의 기능이 수행될 수 있으며 각각의 기능에 대해 하나 또는 그 이상의 기능 인터페이스가 기술될 수 있다.

2. 기능 인터페이스

기능 인터페이스 부분에서는 서로 협력관계를 유지하는 컴포넌트가 지원해야할 서비스 명세를 기술한다. 기능 인터페이스 부분은 그림 8과 같이 OMG에서 지원하는 일반적인 이벤트 명세서에 정의된 IDL 인터페이스 기준을 따르며 다음과 같은 문법 규칙을 가지고 있다.

```
  문법
 <Role>.<Module>::<Interface>
 {:<Module>::<Interface> }
Connector EventNotification
Role :
 EventPushSupplier,
 EventPushConsumer,
 EventPullConsumer,
 EventChannel,
Role Interface :
EventPushSupplier.CosEventComm::PushSupplier;
EventPushConsumer.CosEventComm::PushConsumer ;
EventPullConsumer.CosEventComm::PullConsumer;
EventChannel.CosEventChannelAdmin
::ProxyPushSupplier:CosEventComm::PushSupplier;
EventChannel.CosEventChannelAdmin
::ProxyPushConsumer:CosEventComm::
PushConsumer ;
 EventChannel.CosEventChannelAdmin
 ::ProxyPullSupplier:CosEventComm::PullSupplier;
 EventChannel.CosEventChannelAdmin::
 ConsumerAdmin;
EventChannel.CosEventChannelAdmin::SupplierAdmin;
EventChannel.CosEventChannelAdmin::EventChannel;
```

그림 8. 이벤트통보 연결자의 기능과 기능 인터페이스

Fig. 8. Role of the Event Notification Connector and Role Interface

3. 상호작용 프로토콜

협력관계를 유지하는데 있어서 연결자 내부의 실제적인 동작은 상호작용 프로토콜을 통해 지정되어진다. 인터페이스 프로토콜은 선행조건(precondition) 및 후행조건(postcondition)을 고려하여 협력 관계에 대한 동작과 동작의 순서를 기술한다. 그림 9에서와 같이 이벤트 데이터 교환(exchanging event data)을 위해서는 4개의 동작으로 표현된다. 인터페이스 프로토콜을 기술할 때 오름차순 순서를 갖는 숫자는 수행되는 동작의 순서를 의미하며, 알파벳 문자의 순서는 동시에 수행되는 병행 동작을 의미한다.

```
Connector EventNotification
Protocol :
- Interaction Exchanging Event Data :
Precondition :
```

each role is played by a component using the interaction Creating EventChannel, Connecting PushConsumer to Channel, . . .
Postcondition :
event data is transmitted from an EventPushSupplier
to all registered EventPushConsumer and buffered for EventPullConsumers
Actions :
1 EventPushSupplier sends event to EventChannel
2a EventChannel sends event pushes data to EventPushConsumer
2b EventChannel buffers event for EventPullConsumer
2b/3 EventPullConsumer quries EventChannel, EventChannel delivers buffered event
-Interaction Creating EventChannel :
. . .
-Interaction Connecting PushSupplier to Channel:
. . .
-Interaction Connecting PullConsumer to Channel:
. . .
-Interaction Connecting PushConsumer to Channel:
Precondition :
an EventChannel and a Consumer component exist
Postcondition :
Consumer is connected to EventChannel according to the push-model
Actions :
1 PushConsumer obtains ConsumerAdmin factory via EventChannel interface
2 PushConsumer obtains ProxyPushSupplier reference via the factory ConsumerAdmin
3 PushConsumer connects to channel via ProxyPushSupplier interface

그림 9. 이벤트통보 연결자의 상호작용 프로토콜
Fig. 9. Interaction Protocol of the Event Notification Connector

V. 결 론

본 연구는 기존에 객체지향 환경 및 분산 환경 시스템에서 제시된 객체간 및 컴포넌트간의 연결 및 통합을 위한 기법을 기반으로 하여 현재 분산 시스템 환경에서 적용할 수 있도록 활용 범위를 확대하였다.

따라서 본 논문의 특징은 기존의 객체 및 컴포넌트를 위한 연결 기법을 기반으로 하여 연결자와 컴포넌트를 원활하게 연결하기 위한 컴포넌트 스키마를 도입하였으며, 이를 CORBA 환경에서 분산 JavaBeans 컴포넌트간에 연결자를 사용하여 기존의 동기화 방식에 추가적으로 비동기화 방식으로도 컴포넌트 연결을 위한 정보를 주고받을 수 있도록 하였다.

향후에 연구 과제로는 보다 보완 된 상용화된 컴포넌트의 통합 및 연결이 이루어질 수 있도록 하기 위하여, 다양한 응용 분야에 존재하는 객체 및 컴포넌트에 대한 연결 및 통합이 진행되어야 한다.

참고문헌

[1] Joao Pedro Sousa and David Garlan, "Formal Modeling of the Enterprise JavaBeansTM Component Integration Framework", FM '99 : World Congress on Formal Methods, Sep, 1999.
[2] Robert Orfall, Dan Harkey, Client/Server Programming with JAVA and CORBA, Guide, John Wiley & Sons Inc., 1997.
[3] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Logensen, Object-oriented Modeling and Design, Prentice-Hall International Editions, 1991.
[4] Philippe Kruchten, Modeling Component Systems with the Unified Modeling Language, relation Software Corp, 1997.
[5] Robert Orfall, Dan Harkey, Jeri Edwards, The Essential Distributed Objects Survival Guide, John Wiley & Sons Inc., 1996.
[6] Mary Campione, Kathy Walrath, The Java

Tutorial: Object-oriented Programming, Addison Wesly, 1996.
[7] Ken Arnoldm James Gosling, The Java Programming Language, Addison Wesly, 1999.
[8] "Component-Based Software Engineering", IEEE Software, pp.34-36, 1998.

저자소개

정성옥(Sung-Ok Jung)
1987년 조선대학교 전자계산학과(이학사)
1989년 조선대학교 전자계산학과(이학석사)
2000년 조선대학교 전자계산학과(이학박사)
1992년~1996년 광주여자전문대학 전산정보처리과 조교수
1997년~현재 광주여자대학교 인터넷정보학과 조교수
※관심분야: 소프트웨어공학, 객체지향프로그래밍, 분산처리시스템, 멀티미디어시스템, 컴포넌트 소프트웨어

김재석(Jae-Seog Kim)
1994년 광주대학교 전자계산학과(공학사)
1996년 조선대학교 전자계산(공학석사)
2002년 조선대학교 컴퓨터공학과(공학박사)
※관심분야: 멀티미디어시스템, 영상처리, Morphology, 분산객체 시스템