SDR 컴포넌트의 동적 배치를 위한 SCA 기반 컴포넌트 프레임워크의 설계

(Designing SCA-Based Component Framework for Dynamic Deployment of SDR Components)

김 세 화 † 홍 성 수 †† 장 래 혁 ††
(Saehwa Kim) (Seongsoo Hong) (Naehyuck Chang)

요 약 SDR(Software Defined Radio, 소프트웨어 기반 무선 통신) 포럼에서 표준으로 인정된 SCA(Software Communication Architecture)는 내장형 시스템 소프트웨어의 설계 패턴을 잘 활용한 프레임워크를 제공하고 있다. 그러나 SCA는 (1) 컴포넌트 인터페이스를 표현하고 이를 구현하는 방법에 대하여 정의하는 컴포넌트 모델과 (2) 배치 단위에 무엇을 어떻게 패키지할 지에 대하여 정의하는 패키지 모델, 그리고 (3) 배치 환경과 절차를 정의하는 배치 모델에 대한 명시적인 표준을 제시하지 않고 있어 컴포넌트 프레임워크로서 부족한 문제점이 있다. 본 논문에서는 SCA를 기반으로 하여 SDR을 위한 컴포넌트 프레임워크를 제시한다. 구체적으로 (1) 객체 관리 기능을 지원하는 특성화된 CORBA 객체로서의 컴포넌트를 정의하는 컴포넌트 모델, (2) SCA의 XML 디스크립터를 활용하는 패키지 모델, (3) SCA 기반의 배치 환경, 배치 상태를 복구하는 시동 절차, 느린 응용 인스턴스화와 동적 컴포넌트 교체를 지원하는 배치 절차를 정의하는 배치 모델을 제시한다.

키워드 : 컴포넌트 기반 설계, 소프트웨어 프레임워크, 내장형 시스템, 소프트웨어 기반 이동 통신(SDR), 분산 객체 컴퓨팅

Abstract SCA (Software Communication Architecture), which has been adopted as a SDR (Software Defined Radio) Forum standard, provides a framework that successfully exploits common design patterns of embedded systems software. However, the SCA is inadequate as a component framework since it does not explicitly specify (1) a component model that defines how to express a component interface and how to implement it, (2) a package model that defines what and how to package in deployment units, and (3) a deployment model that defines the deployment environment and deployment process. In this paper, we propose a SCA-based component framework for SDR. Specifically, we present (1) a component model that defines a component as a specialized CORBA object that implements object management functionality, (2) a package model exploiting the existing XML descriptors of the SCA, and (3) a deployment model that defines a SCA-based deployment environment, a boot-up process that restores the deployment state, and a deployment process that supports lazy application instantiation and dynamic component replacement.

Key words : Component-Based Development, Software Framework, Embedded Systems, Software Defined Radio(SDR), Distributed Object Computing

· 본 연구는 과학기술부 국가지정연구실사업(2000-N-NL-01-C-136)과 한국과학재단 목적기초연구(R01-1999-00206) 지원으로 수행되었음.
† 비 회 원 : 서울대학교 전기컴퓨터공학부
ksaehwa@redwood.snu.ac.kr
†† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
sshong@redwood.snu.ac.kr
naehyuck@snu.ac.kr
논문접수 : 2002년 10월 17일
심사완료 : 2003년 3월 10일

1. 서 론

컴포넌트 소프트웨어 기술은 밑바닥부터 소프트웨어를 개발하는 대신 배치 수준의 소프트웨어를 조합하여 새로운 소프트웨어를 생성하는 것을 목적으로 하고 있다. 즉 소프트웨어의 재사용성을 극대화시켜 내장형 소프트웨어의 복잡성 문제를 해결하고 time-to-market을 줄이는데 크게 도움이 된다. 컴포넌트의 의미는 기존의

많은 문서와 논문 등에서 여러 가지의 의미로 사용되어 혼동을 일으켰는데, 본 연구에서는 OMG(Object Management Group)[1]에서 정한 컴포넌트 정의를 따른다: 컴포넌트는 모듈화되고 배치와 교체가 가능한 시스템의 일부로서 구현을 숨기고 인터페이스의 집합을 드러낸다[2]. 컴포넌트는 보통 공유 라이브러리의 형태이지만 배치 환경에 따라 바이너리, 바이트 코드, 또는 스크립트 언어로 작성된 소스 파일일 수도 있다.

SDR(Software Defined Radio) 포럼[3]에서는 JTRS(Joint Tactical Radio System)[4]의 SCA(Software Communication Architecture)[5]를 SDR 내장형 시스템을 위한 소프트웨어 구조의 표준으로 인정하였다. SCA는 통상적으로 디바이스 제어 프로그램과 응용 프로그램으로 이루어진 내장형 시스템 소프트웨어의 디자인 패턴을 잘 활용한 프레임워크를 제공하고 있다. 또한 분산 객체 모델의 실질적인 산업 표준인 CORBA(Common Object Request Broker Architecture)[6]를 기반 미들웨어로 채택하여 이기종의 하드웨어와 다양한 언어로 작성된 소프트웨어에 대하여 유연한 통합 환경을 제공한다. 그러나 현재의 SCA는 컴포넌트 소프트웨어를 위한 프레임워크로서는 부족한 문제점이 있다. 구체적으로 (1) 컴포넌트 인터페이스를 표현하고 이를 어떻게 구현하는 지에 대하여 정의하는 컴포넌트 모델과 (2) 배치 단위에 무엇을 어떻게 패키지할 지에 대하여 정의하는 패키지 모델, 그리고 (3) 배치 환경과 절차를 정의하는 배치 모델에 대한 명시적인 표준을 제시하지 않고 있다.

본 논문에서는 이러한 SCA의 문제점을 해결하여 SDR을 위한 SCA 기반 컴포넌트 프레임워크를 제시한다. 구체적으로 (1) CORBA 객체들을 포함하고, 포함된 객체들을 관리하는 기능을 제공하는 특성화된 CORBA 객체로서의 컴포넌트를 정의하는 컴포넌트 모델, (2) SCA 도메인 프로파일의 XML 디스크립터를 활용하여 배치 단위인 패키지를 정의하는 패키지 모델, (3) SCA 코어 프레임워크 인터페이스들을 활용하여 배치 환경을 정의하며, SCA 도메인 프로파일을 활용하여 배치 상태를 복구하는 시동 절차를 제시하며, 또한 느린 응용 인스턴스화와 동적 응용 컴포넌트 교체를 지원하는 배치 절차를 제시하는 도메인 모델을 제시한다. 본 연구의 느린 응용 인스턴스화를 통하여 응용 프로그램은 설치 뒤에 선택적으로 인스턴스화 될 수 있다. 또한 동적 컴포넌트 교체 메커니즘을 통하여 응용 프로그램은 동적으로 업그레이드될 수 있다.

1.1 관련 연구

분산 객체 컴퓨팅 환경에서의 컴포넌트 지원 프레임워크에 대한 연구 중 대표적인 것으로 CCM (CORBA Component Model)[7], EJB(Enterprise Java Beans)[8], DCOM(Distributed Component Object Model)[9]을 들 수 있다. 이 중에서 CCM이 가장 최근의 표준이며 나머지 둘을 포괄하는 진보된 기술을 지원한다. CORBA는 CCM의 도움 없이 그 자체로 DCOM 컴포넌트와의 연동을 지원한다. 또한 CCM의 기본 컴포넌트는 EJB 컴포넌트에 해당하며 CCM은 EJB 컴포넌트와의 연동 표준을 지원한다. 이와 같이 CCM이 EJB와 DCOM과 연동이 가능하며 또한 SCA가 CORBA에 기반을 두고 있다. 따라서 본 논문에서 CCM과 비교하여 본 연구의 차별성을 설명한다.

CCM은 1997년 6월에 처음 RFP[10]가 공표되고, 이어 2001년 11월 최종 초안이 나온 후 2002년 6월에 발표된 CORBA3[6]에 핵심적인 구성요소로 포함되었다. CCM은 표 1과 같은 구성 요소로 이루어져 완전한 컴포넌트 프레임워크를 제시하고 있다. 그러나 CCM은 범용 시스템에 설치되는 서버측 응용에만 초점을 맞추었기 때문에 SDR과 같은 내장형 시스템에 그대로 적용될 수 없다[11]. SDR 시스템은 대기하는 서버 응용 프로그램과 대조되는 독립적인 무선 응용들로 구성된다. 컨테이너, home executer, reflection 기능 등의 CCM의 많은 특성들의 사용을 강제하는 것은 SDR 이동 단말 시스템의 경량성을 헤치게 되기 때문에 바람직하지 못하다. 한편 소프트웨어의 배치와 관련되어 필요한 하드웨어의 구성 정보의 배치에 대하여서도 고려하여야 한다. 또한 비휘발성 저장소가 부족하여 런타임 메모리 상태를 백업하거나 완전한 체크 포인팅을 지원할 수 없는 SDR 내장형 시스템이 있을 수 있기 때문에, SDR 시스템이 시동할 때 배치 상태를 복구하는 방법에 대하여서도 고려하여야 한다. 표 1에 이와 같이 CCM을 SDR 시스템에 그대로 적용하였을 때 불합리한 문제점들을 정리하였다. CCM에 SDR 시스템에 적용하였을 때 불합리한 많은 특성들이 존재하는 이유는 CCM이 프로그래밍을 최대한 쉽게 하고 최소한 적게 하게 하는 것을 목적으로 하여 경량성을 헤칠 수 밖에 없는 빌트인 라이브러리들을 제공하기 때문이다.

CCM을 내장형 시스템에 특화시키고자 하는 연구들로서 특정 도메인에 적용시킬 수 있도록 표준을 확장하거나[13], CCM을 구현함에 있어서 성능을 향상시키기 위한 노력들이 몇몇 있었다[14, 15]. 또한 CCM이나 EJB, DCOM 등의 표준들과 상관없이 동적 소프트웨어의 재구성을 위한 자체적인 컴포넌트 프레임워크의 설계에 대한 연구들도 몇몇 있어 왔다[16, 17]. 그러나 이러한

표 1 CCM의 구성 요소와 SDR 시스템에 적용할 때 불합리한 점

	설 명	SDR 시스템에 적용할 때 불합리한 점
컴포넌트모델	• 컴포넌트 타입: CORBA 객체 인터페이스를 확장한 것 • IDL3: 컴포넌트 인터페이스를 기술하기 위하여 기존의 IDL을 확장하여 정의한 것 • CIDL(Component Implementation Definition Language): 컴포넌트가 설치되어 구동되게 되는 환경을 기술한 것	• 포트의 개념이 SCA의 포트 개념과 일치하지 않음 • 컨테이너가 구현 타입별로 각 노드에 빌트인 라이브러리로서 존재해야 함 • 컴포넌트마다 자체 reflection(navigation) 기능을 지원해야 함 • 이벤트 COS를 통해서 이벤트 기능을 사용할 수 있음에도 불구하고 중복적으로 컴포넌트 자체에서도 이벤트 서비스 기능을 지원해야 함 • 컴포넌트의 동적 생성을 위하여 각 컴포넌트마다 home executer가 존재해야 함
패키지모델	컴포넌트의 배치 정보를 기술하는 XML Descriptor DTD를 제시하고, 이를 기반으로 하여 ZIP 파일인 배치 단위의 구성 요소를 제시한 것	• 하드웨어 디바이스 구성 정보를 기술하는 표준 결여 • SCA의 인터페이스들의 패키지 매핑 모델이 존재하지 않음
배치모델	배치 소프트웨어가 제공해야 할 인터페이스와 컴포넌트의 설치 절차를 제시한 것	• 부팅 절차에 대한 표준 결여 • 하드웨어 특성을 고려한 배치 모델 표준 결여 • SCA 인터페이스 별로 특화된 배치 방법 결여

연구들은 본 연구에서와 같이 내장형 시스템의 특성을 고려하거나 SCA에 기반을 두고 있지 않고 있기 때문에 SDR 시스템에 사용될 수 없다는 문제점이 있다.

이하 논문의 구성은 다음과 같다. 먼저 2절에서 SCA를 개관한다. 3절에서 SCA를 컴포넌트 프레임워크로 보완하기 위하여 본 논문에서 제시하는 컴포넌트 모델, 패키징 모델, 배치 모델을 기술한다. 4절에서는 본 논문에서 제시한 컴포넌트 프레임워크의 장점을 기술하고 마지막으로 5절에서 결론을 맺는다.

2. SCA 개관

SCA의 소프트웨어의 구조를 간략하게 나타내면 그림 1과 같다[5]. 즉, 크게 응용 프로그램과 운영 환경(Operating Environment: OE)의 계층구조로 나누어지며, 운영 환경 OE는 RTOS, CORBA, 코어 프레임워크(Core Framework: CF) 인터페이스의 계층구조로 다시 나누어진다. 여기에서 CORBA와 RTOS는 COTS (Commercial Off-The-Shelf: 상용 출시 제품)를 사용하는 것을 가정한다. CORBA 미들웨어의 사용을 가정하기 때문에 응용 프로그램은 기본적으로 SCA코어 프레임워크를 준수하는 CORBA 객체들로 이루어진다. 코어 프레임워크 명세는 인터페이스들과 도메인 프로파일(domain profile)의 명세로 구성된다. 이하 각 절에서 이 두 가지를 설명한다.

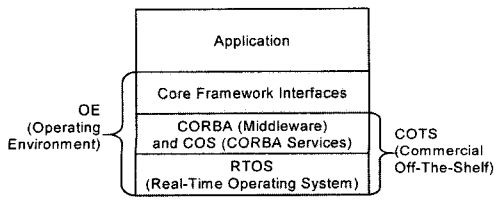


그림 1 SCA의 소프트웨어 구조[5]

2.1 프레임워크 인터페이스들

그림 2는 코어 프레임워크의 인터페이스들과 그들의 관계를 보여준다. 단순성을 위하여 상위 클래스로만 사용되는 몇몇 인터페이스들은 생략하였다. 그림에 나타냈듯이, 코어 프레임워크의 인터페이스들은 기본 응용 프로그램, 프레임워크 제어, 그리고 프레임워크 서비스의 세가지 그룹으로 분류된다.

(1) 기본 응용 프로그램 인터페이스들

A. Resource: 응용 프로그램을 구성하는 기본 컴포넌트[1] 인터페이스

B. Port: Resource 객체가 다른 Resource 객체로 연결하고 연결을 끊을 수 있게 하는 인터페이스를 제공

C. PortSupplier: Resource 객체가 상속하여 자신의 Port 객체에 대한 참조를 얻을 수 있도록 하게 함

D. ResourceFactory: Resource 객체에 대한 Factory 디자인 패턴을 제공

(2) 프레임워크 제어 인터페이스들

A. Application: 인스턴스화된 응용 프로그램을 관리하는 인터페이스를 제공

B. ApplicationFactory: Application 객체에 대한 Factory 디자인 패턴을 제공

1) SCA 용어에서 컴포넌트는 기능의 완전한 집합을 구현하는 소프트웨어 모듈이나 요소로서 정의되어 있다.

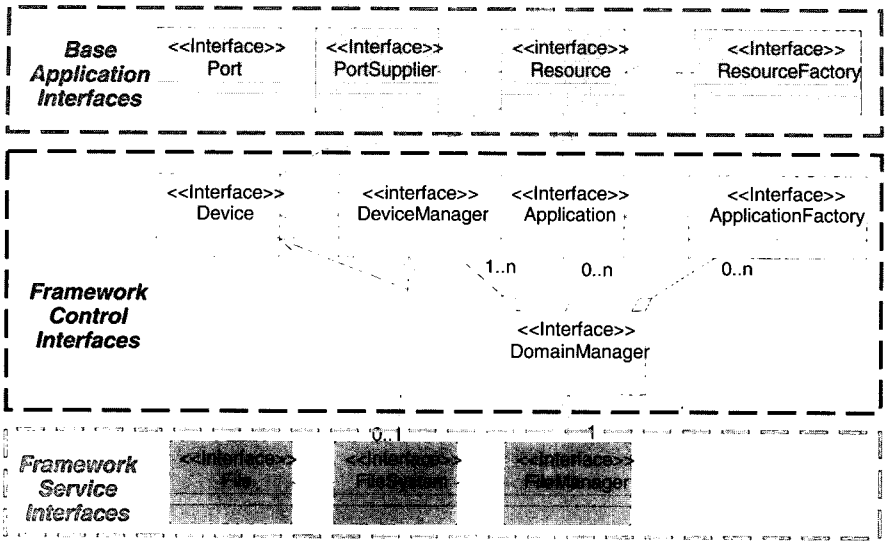


그림 2 SCA 코어 프레임워크의 인터페이스들[5] (몇몇 인터페이스는 생략함)

C. Device: 하나 또는 여러 개의 하드웨어 디바이스를 관리하는 논리 디바이스 객체의 인터페이스. 디바이스의 용량을 할당/비할당하고 파일을 로드/언로드/수행하는 인터페이스를 제공

D. DeviceManager: Device 객체와 서비스 객체의 컨테이너로서 Device 객체를 등록/비등록/검색하기 위한 인터페이스를 제공

E. DomainManager: Application, Device, 그리고 DeviceManager 객체들을 조율하고, 나열하고, 설치/제거하는 인터페이스를 제공

(3) 프레임워크 서비스 인터페이스들

A. File: 파일을 읽고/쓰고/닫는 인터페이스를 제공

B. FileSystem: 파일을 생성하고/삭제하고/복사하는 인터페이스를 제공

C. FileManager: 파일 시스템을 마운트/언마운트 하는 인터페이스를 제공. DomainManager 객체가 여러 개의 분산 파일 시스템을 관리할 수 있게 해줌

2.2 도메인 프로파일

도메인 프로파일은 시스템의 하드웨어와 소프트웨어의 구성 정보를 기술하기 위한 XML 디스크립터 파일들로 그림 3과 같은 DTD element로 구성된다. 최상위 수준에서 하나의 도메인을 기술하는 도메인 프로파일은 그림 3과 같이 (1) 하나의 DomainManager 구성(Configuration) 디스크립터, (2) Device 구성 디스크립터들, 그리고 (3) 소프트웨어 어셈블리 디스크립터들로 구성된다. 이들 디스크립터들은 또한 그림 3과 같이 다른 디스크립터들을 참조하는 연관 관계를 가지고 있다. DomainManager 구성 디스크립터는 DomainManager 인터페이스를 지원하는 컴포넌트의 배치와 인터페이스 정보를 기술한다. Device 구성 디스크립터는 하드웨어 디바이스들의 구성 정보와 그 하드웨어 디바이스들을 관리하는 논리 Device 객체들, 그리고 DeviceManager 객체의 배치와 인터페이스 정보를 기술한다. 마지막으로 소프트웨어 어셈블리 디스크립터는 하나의 응용 프로그램을 이루는 소프트웨어 컴포넌트들의 배치와 연결 정보를 기술한다.

이들 디스크립터와 연관되어 사용되는 나머지 도메인 프로파일 디스크립터들을 살펴보면 다음과 같다. 소프트웨어 팩키지 디스크립터는 특정 컴포넌트 타입의 구현과 배치 정보를 기술한다. 또한 해당 컴포넌트의 인터페이스 정보를 기술하는 소프트웨어 컴포넌트 디스크립터에 대한 참조를 가지고 있다. 디바이스 팩키지 디스크립터는 하드웨어 디바이스의 종류, 시리얼 넘버, 메모리 크기 등의 속성을 기술한다. 속성(Properties) 디스크립터는 이름, 값의 쌍들의 집합으로 임의의 필요한 정보를 기술한다. 마지막으로 프로파일 디스크립터는 디스크립터 파일의 참조를 기술한다.

3. 제안하는 컴포넌트 프레임워크

이하 각 절에서 SCA를 완전한 컴포넌트 프레임워크로서 보완하기 위하여 본 연구에서 제안하는 (1) 컴포넌트 모델, (2) 패키지 모델, (3) 배치 모델에 대하여 차례

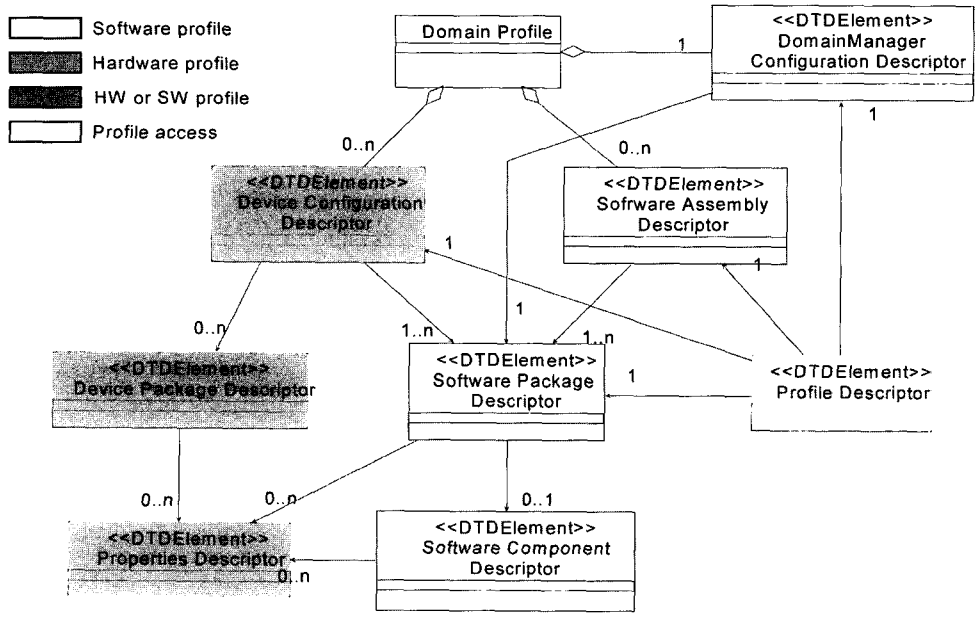


그림 3 도메인 프로파일[5]

로 설명한다.

3.1 컴포넌트 모델

이 절에서는 컴포넌트의 인터페이스를 어떻게 설계할지에 대하여 기술하고 이에 대한 근거를 제시한다. 이를 위하여, 본 연구에서는 먼저 객체 관리 기능을 격리시키는 본 연구의 접근 방법을 설명한다. 그리고 컴포넌트의 인터페이스를 기술하기 위하여 사용되는 포트의 개념에 대하여 설명한다. 이어 본 연구에서 제시하는 공통 컴포넌트 인터페이스를 제시하고, 본 연구의 컴포넌트 모델에서 컴포넌트들을 어떻게 연결하는지에 대하여 설명한다.

컴포넌트는 구현은 숨기고, 인터페이스의 집합을 드러내면서 소프트웨어를 구성하는 배치 가능하고 교체 가능한 파일이다. 분산 객체 컴퓨팅 시스템인 코바 환경에서, 본 연구의 프레임워크에서의 컴포넌트는 자신의 인터페이스를 가지고 런타임 코바 객체로서 표현될 수 있다. 그러면, 배치 도구가 컴포넌트를 조합함으로써 소프트웨어를 구성할 수 있도록 하기 위하여 표준화된 공통 컴포넌트 인터페이스를 제공할 필요성이 있다.

본 연구의 컴포넌트 모델에서, 런타임에서의 컴포넌트는 객체들의 집합으로 볼 수 있다. 컴포넌트에 포함된 객체들의 기능은 하나 또는 다수의 소프트웨어 응용의 일부를 이룬다. 본 연구의 컴포넌트는 또한 응용 도메인에 속하지 않는 기능인 객체 관리 기능을 제공한다. 객체 관리 기능은 객체의 라이프 사이클을 관리하고 객체들을 연결하는 일들을 포함한다. 객체 관리 기능이 응용 도메인 자체에 속하지는 않지만, 융통성 있고(메모리 및 CPU) 자원을 절약하는 서버 객체 시스템을 가능하게 하기 위하여 필수적이다. 사실상 코바 프로그래밍은 본질적으로 응용 객체의 주위에 스케일 가능한 적절한 객체 관리 기능을 감싸는 것이라고 할 수 있다.

3.1.1 접근 방법: 객체 관리 기능의 격리

본 연구의 컴포넌트 모델에서, 컴포넌트 인터페이스의 인스턴스화된 객체인 컴포넌트 객체는 객체 관리 기능만을 격리시켜 제공하고, 응용 도메인의 기능은 전혀 담당하지 않는다. 컴포넌트 객체는 (1) 내부적으로 포함된 객체들의 라이프 사이클을 관리하며, (2) 외부적으로는 포함된 객체들과 다른 컴포넌트의 객체들을 연결하는 기능만을 담당한다. 이러한 접근 방법에 따라 본 연구의 컴포넌트 모델에서의 컴포넌트 인터페이스는 코바 객체 인터페이스들을 조합할 수는 있으나, 다른 응용 도메인의 코바 객체 인터페이스를 상속할 수는 없도록 강제한다[2]. 이러한 접근 방법은 나눌 수 있는 기능에 따라 동

2) CCM에서의 컴포넌트의 인터페이스는 다른 일반 객체 인터페이스를 상속하는 것을 허용하고 있다. 그러나 컴포넌트 인터페이스를 상속하는 것에 대하여서는 단일 상속만을 강제하고 있다.

작을 감싸는 객체 지향의 기본적인 원리를 따른 것이다.

본 연구의 접근 방법은 또한 내장형 시스템 소프트웨어의 특성에도 근거를 두고 있다. 범용 시스템에서의 서버 응용을 이루는 객체들은 많은 익명의 클라이언트 객체들로부터 요청을 받을 수 있다. 즉, 서버 시스템으로의 객체들의 연결은 런타임에 동적으로 자주 발생하게 된다. 이와는 반대로 내장형 시스템에서의 객체들은 배치 시점에 정적으로 연결되고 이후 연결을 유지하게 된다. 따라서 중앙 집중화된 객체에 객체간 연결 기능을 포함한 객체 관리 기능을 격리시키는 것이 바람직하다. 본 연구에서의 컴포넌트 객체가 바로 이러한 중앙 집중화된 객체에 해당한다. 본 연구의 컴포넌트 IDL 인터페이스를 구현하는 컴포넌트 객체는 컴포넌트 간의 연결이 발생하는 배치 시점에서만 쓰레드나 프로세스와 같은 실행 가능한 프로그램 개체(서번트)와 연결이 된다. 시스템의 정상(steady) 상태에서 컴포넌트 객체는 공표된 IOR(Interoperable Object Reference)를 가지고 CORBA 객체로서 존재하지만 요청을 수행할 수 있는 서번트를 가지고 있지 않게 된다. 대신 응용 도메인에 적용되는 기능들을 담당하는 컴포넌트 내에 포함된 객체들만이 서번트를 가지고 있게된다. 컴포넌트 객체는 자신을 포함하여 연관된 컴포넌트가 교체된다든지 하여 연결을 재설정할 필요가 있을 때에만 다시 서번트를 가지게된다.

관리 기능을 응용 관련 기능과 분리하는 본 연구의 접근방법은 또한 CCM의 접근방법과도 상통하는 측면이 있다. 서버 측면 코바 프로그래밍은 그 객체 관리에 대한 프로그래밍의 난이도가 높은 것으로 악명이 높았다. CCM에서 이런 문제를 해결하기 위하여, 객체 관리 기능 부분을 일부는 코드를 생성함으로써, 일부는 빌트인 라이브러리를 제공함으로써, 프로그래머가 이를 자동적인 방법으로 구현할 수 있게 해준다. 즉, CCM도 응용 기능과 객체 관리 기능의 코드를 분리한다. 그러나 CCM의 전략은 경량성 있는 소프트웨어의 요구사항 때문에 내장형 시스템에서는 바로 적용되어 사용될 수 없다. 자동적인 메커니즘은 매우 한정적인 객체 관리 정책을 제공하며 양이 많은 코드를 생성하고, 무거운 라이브러리에 의존하는 경향이 있다. 따라서 프로그래머가 종종 자신의 최적화된 경량화된 코드를 직접 작성해야 할 필요가 있다.

3.1.2 포트

포트(ports)는 컴포넌트가 상호 동작할 수 있도록 하는 이름이 있는 연결점(named connection point)을 의미하는 용어로, CCM과 SCA 도메인 프로파일에서 컴포넌트의 연결 정보를 나타내기 위하여 사용되고 있다. 이 절에서는 컴포넌트의 인터페이스를 선언하고 컴포넌트들 간의 연결 정보를 기술하기 위하여 기본적으로 필요한 개념인 포트에 대하여 설명한다.

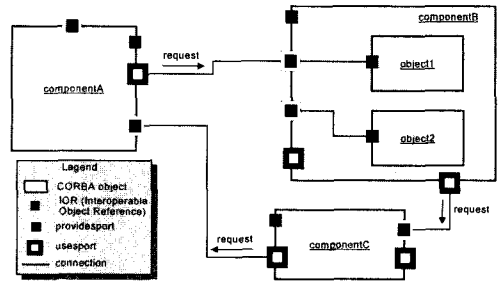


그림 4 본 연구에서의 컴포넌트 모델에서의 컴포넌트간의 연결

CCM에서의 컴포넌트 간의 연결(connection)은 컴포넌트 포트간의 단방향 이원 통신 연결로, 동기 통신을 위한 포트들뿐만 아니라 이벤트 채널을 통한 비동기 통신을 위한 포트들도 지원한다. 본 연구의 컴포넌트 모델에서는 이벤트 채널을 통한 비동기 통신을 위한 포트는 지원하지 않는다. 왜냐하면 이벤트 채널을 통한 연결은 이벤트 서비스의 이용을 통하여 일어나며, 객체들간 직접적인 연결을 필요로 하지 않기 때문이다. 즉, 컴포넌트의 합성을 위하여 사용해야 할 정보가 아니므로 고려하지 않는다. CCM에서 객체들간의 동기 통신을 위하여 지원하는 포트들은, 요청을 보내는 쪽을 *uses* 포트라고 하며, 요청을 받아서 처리하는 쪽의 포트를 연결되는 객체의 종류에 따라 *supports* 포트와 *provides* 포트라고 한다. *Supports* 포트는 CCM에서 컴포넌트가 상속하여 제공하는 객체로의 연결을, *provides* 포트는 컴포넌트에 속한 객체로의 연결을 의미한다.

앞에서 기술하였듯이, 본 연구의 컴포넌트 모델에서의 컴포넌트 인터페이스는 합성만이 가능하며 상속은 지원하지 않는다. 따라서 본 연구의 컴포넌트 모델에서는 *supports* 포트를 사용하지 않으며, *uses* 포트와 *provides* 포트만 사용한다. 따라서 본 연구의 컴포넌트 모델에서의 컴포넌트 간의 연결은 단방향 이원통신 연결이다. 여기에서, 연결 양 끝단의 각 포트에 대하여 서버 쪽을 *provides* 포트, 클라이언트 쪽을 *uses* 포트이다. *Provides* 포트는 컴포넌트에 속한 서버 객체의 객체 참조이며, *uses* 포트는 서버 객체와 연결된 대리 객체에 대한 객체 참조이다. 그림 4는 본 연구의 컴포넌트 모델에서의 포트를 통한 컴포넌트들의 연결을 개념적으로 보여준다.

3.1.3 컴포넌트 공통 인터페이스

3.1.1절에서 기술하였듯이, 중앙 집중화된 객체(컴포넌트 객체)가 객체 관리 기능을 담당하며 이 객체가 서버 객체로서 컴포넌트 인터페이스를 통해 자신의 인터페이스를 드러낸다. 그러면 여기에서 배치 도구가 표준화된 방법으로 컴포넌트를 배치할 수 있도록, CORBA IDL로서 표현되는 공통 컴포넌트 인터페이스를 정의할 필요가 있다. 이 절에서는 이에 대한 해결책을 제시한다.

본 연구의 컴포넌트 모델에서 컴포넌트 인터페이스는 IDL(IDL2)과 XML 디스크립터(SCD)만을 사용하여 선언된다. 구체적으로, IDL에서는 컴포넌트의 이름만을 기술하며, SCD에서 컴포넌트의 포트 정보를 기술한다[3]. IDL에서 컴포넌트 인터페이스를 정의하는데 사용될 컴포넌트의 공통 인터페이스로서 본 연구에서는 그림 5와 같은 SCAComponentObject 인터페이스를 제안한다. IDL에서 컴포넌트 인터페이스의 선언은 다음과 같이 컴포넌트의 이름 정보만을 기술한다.

interface component_name:SCAComponentObject {};

즉, SCAComponentObject만을 상속하여 몸체(body) 없이 기술한 인터페이스가 컴포넌트 인터페이스이다. SCAComponentObject는 전술하였듯이 컴포넌트에 속한 객체들의 연결을 위한 메소드들을 제공하는 인터페이스이다. SCAComponentObject는 그림 5와 같이 UsesPort와 ProvidesPort 인터페이스를 상속한다. ProvidesPort는 *provides* 포트를 제공하기 위한 메소드를 가지고 있으며, UsesPort는 *uses* 포트의 값을 세팅하고 얻기 위한 메소드를 가지고 있다. SCAComponentObject의 attribute인 *identifier*는 컴포넌트 인스턴스의 고유 식별자로서 사용된다. 프로그래머는 컴포넌트 마다 해당하는 포트 목록에 따라 SCAComponentObject 인터페이스가 제공하는 다음의 메소드들을 구현해야 한다.[4]

- Object getUsesPort(in string usesPortName) raises (UnknownPort); usesPortName으로 주어진 *uses* 포트 객체 참조를 리턴한다. 인자의 usesPortName이 SCD에 기술되지 않은 *uses* 포트일 경우 Unknown Port 예외를 발생시킨다.
- void connectPort(in string usesPortName, in Object providesPort, in string connectionId) raises (UnknownPort, InvalidPort, OccupiedPort);usesPort Name의 *uses* 포트를 providesPort 객체 참조와 연결시킨다. connecti onId는 연결을 끊을 때 사용하기 위한 포트간 연결의 식별자이다. 인자의 usesPortName이 SCD에 기술되지 않은 *uses* 포트일 경우 UnknownPort 예외를 발생시킨다. 인자의 providesPort 객체 참조가 usesPortName의 포트가 SCD에 기술한 인터페이스를 지원하지 않을 경우 InvalidPort의 예외를 발생시킨다. uses PortName에 connectionId로 이미 연결된 연결이 있거나, 연결 횟수가 정해놓은 횟수를 초과하여 연결을 할 수 없을 경우 OccupiedPort 예외를 발생시킨다.
- void disconnectPort(in string usesPortName, in string connectionId) raises(InvalidConnection);connectionId의 아이디로 usesPortName과 연결된 연결을 끊는다. 인자들로 식별되는 연결이 없을 경우 Invalid Connection 예외를 발생시킨다.
- Object *getProvidesPort(in string providesPort* Name) raises (UnknownPort); providesPortName으로 주어진 *provides* 포트 객체 참조를 리턴한다. 인자의 *providesPortName*이 SCD에 기술되지 않은 *provides* 포트일 경우 UnknownPort 예외를 발생시킨다.

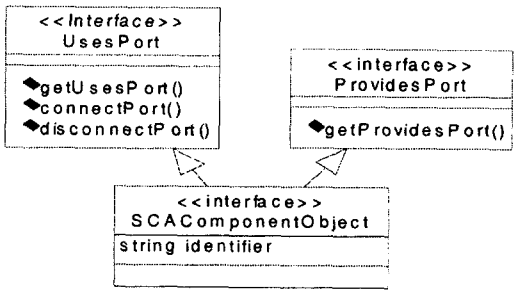


그림 5 SCAComponentObject 인터페이스

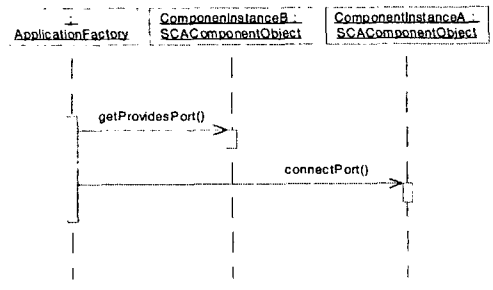
3.1.4 컴포넌트의 연결

컴포넌트들 간의 연결은 응용 프로그램이 활성화되면서 응용 프로그램에 속한 컴포넌트에 속한 객체들이 인스턴스화된 후에 일어난다. 따라서 컴포넌트들을 인스턴스화할 때 컴포넌트들을 연결하는 일은 응용 프로그램을 생성하는 ApplicationFactory 객체에서 담당해야 한

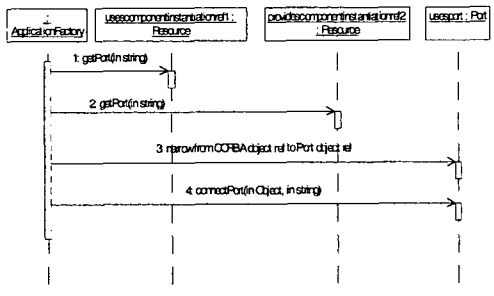
3) CCM에서는 프로그래머가 IDL3에서 포트 정보를 기술하며, IDL3 컴파일러를 통해 컴포넌트 executer의 skeleton을 얻어낸다. 이 skeleton 코드는 각 포트별로 연결 메소드를 가지고 있으며, 프로그래머는 이들의 body를 구현해야 한다. 본 논문에서는 IDL3와 IDL3 컴파일러를 사용하지 않는 대신 컴포넌트 공통 인터페이스를 위와 같이 제공하였다.

4) CCM에서는 프로그래머가 IDL3에서 포트 정보를 기술하며, IDL3 컴파일러를 통해 컴포넌트 executer의 skeleton을 얻어낸다. 이 skeleton 코드는 각 포트별로 연결 메소드를 가지고 있으며, 프로그래머는 이들의 body를 구현해야 한다. 본 논문에서는 IDL3 및 IDL3 컴파일러를 사용하지 않는 대신 컴포넌트 공통 인터페이스를 위와 같이 제공하였다.

다. 본 논문에서 제안한 컴포넌트 공통 인터페이스를 이용한 컴포넌트들의 각 연결은 그림 6(a)와 같이 이루어지게 된다. ApplicationFactory 객체는 연결될 *provides* 포트를 포함한 컴포넌트 인스턴스(ComponentInstanceB)에게 getProvidesPort() 함수를 호출하여 연결할 서버 객체 참조를 얻어낸다. 그리고 연결될 *uses* 포트를 가진 컴포넌트(ComponentInstanceA)에게 connectPort()를 호출하여 ComponentInstanceB의 서버 객체 참조(*provides* 포트)와 ComponentInstanceA의 proxy 객체 참조(*uses* 포트)를 연결한다. 이와 같은 컴포넌트 간의 연결이 끝난 후 *uses*포트를 가진 각 컴포넌트는 getUsesPort(usesPortName) 함수의 호출을 통하여 연결된 uses 포트 참조를 얻어내어 사용한다.



(a) 본 연구의 컴포넌트 모델



(b) 현 SCA 표준

그림 6 객체간의 연결 과정

3.2 패키지 모델

컴포넌트 파일은 소프트웨어를 구성하는 기본 단위이지만 그 자체가 설치의 단위가 될 수는 없다. 컴포넌트 소프트웨어 구성을 위한 배치 단위에는 컴포넌트 파일뿐만 아니라 컴포넌트의 배치 정보를 기술하는 XML 디스크립터와 같은 파일이 포함되어야 한다. 본 연구에서는 여러 파일을 묶은 ZIP 파일 포맷으로서 배치 단위를 나타내는 CCM의 용어인 *패키지*를 채용한다. 본 연구에서는 패키지를 (1) 하나의 컴포넌트 타입을 포함하는 *컴포넌트 패키지*와 (2) 여러 개의 컴포넌트 타입을 포함하는 *컴포넌트 어셈블리 패키지*로 분류한다.

SCA 도메인 프로파일이 디스크립터들과 이들간의 연관 관계에 대해 잘 정의하고 있기 때문에 각 패키지에 어떤 디스크립터들이 패키지되어야 할지를 결정하는 것은 쉬운 일이다. 컴포넌트는 SPD에 의하여 기술되므로, 컴포넌트 패키지는 하나의 최상위 수준 SPD를 포함하고, 이와 함께 SPD가 필요로 하는 다른 디스크립터를 포함하면 된다. 컴포넌트 어셈블리 패키지는 하나의 SAD나 DCD를 포함하고, 이들이 요구하는 다른 디스크립터를 포함하면 된다. 그림 7, 8은 각각 컴포넌트 패키기와 컴포넌트 어셈블리 패키지를 구성하는 요소들을 나타낸다.

한편 SCA 코어 프레임워크의 인터페이스 객체를 설치하는 데 있어서 객체 간에 서로 설치되는 순서에 있어서 제약이 있는 선행 조건과 특정 객체들이 같이 설치되어야 하는 병치(collocation) 조건이 존재한다. 따라서 패키지는 SCA 코어 프레임워크의 인터페이스들을 임의적으로 조합하여 포함할 수 없다. 따라서 어떤 인터페이스들이 같이 패키지될 수 있는지, 없는지를 결정하고 패키지의 종류를 분류할 필요가 있다. 이는 배치 모델과도 밀접하게 연관되므로, 다음 절에서 설명한다.

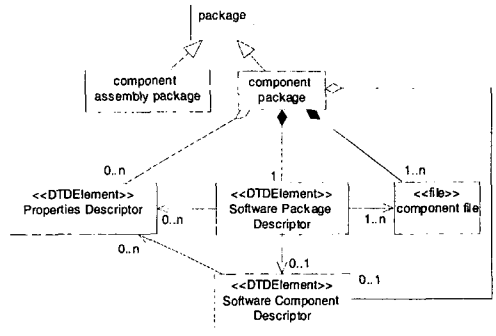


그림 7 컴포넌트 패키지의 구성

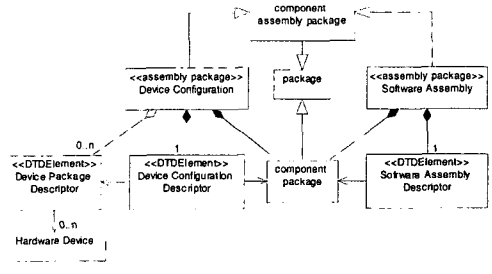


그림 8 컴포넌트 어셈블리 패키지의 구성

3.3 배치 모델

이 절에서는 배치 과정에 참여하는 구성 요소들과 그들의 상호 작용에 대하여 기술한다. 본 연구의 배치 모델에서는 (1) 완전한 배치 환경, (2) 배치상태를 복원하기 위한 시스템 시동 절차, (3) 느린 응용 인스턴스화와 응용 컴포넌트의 동적 교체를 지원하는 배치 절차를 제시한다.

3.3.1 배치 환경

그림 9는 본 연구의 배치 모델에서의 기본적인 배치 환경을 보여준다. 배치 환경을 이루는 설치된 클래스들과 활성화된 객체들은 기본적으로 SCA 코어 프레임워크의 인터페이스들이지만 추가적으로 XML parser와 같은 객체를 포함한다. 이러한 배치 환경을 갖추기 위해서는, DomainManager를 지원하는 컴포넌트 패키지와 DCD로 기술되는 컴포넌트 어셈블리 패키지들이 먼저 설치되어 있어야 한다. 이러한 배치 환경에서 어떤 인터페이스들이 같은 패키지 안에 병치되어야 하거나 말아야 하는 지가 암묵적으로 기술되고 패키지의 타입도 분류된다.

한편 본 연구의 배치 환경에서 프로세서는 범용 마이크로 프로세서나 DSP같은 것을 의미하며, CORBA를 올리는 것이 불가능하여 분산 객체 컴퓨팅이 힘든 FPGA와 같은 디바이스는 포함하지 않는다. 이들은 분산 객체 컴퓨팅이 가능한 다른 프로세서에 존재하는 논리 디바이스 객체에 의하여 접근되고 제어된다. 본 연구의 배치 환경에서 DomainManager 객체는 그림 9에서와 같이 singleton으로서 하나의 프로세서에 존재한다. DomainManager 객체와 함께 Application과 Application-Factory 인터페이스는 병치되어야 한다. 이는 전자가 후자를 직접적으로 사용하기 때문이다. 또한 File Manager 객체와 XML 파서 객체가 singleton으로서 존재해야 한다. 이러한 객체들은 DomainManager 객체와 반드시 병치될 필요는 없다. FileManager 객체는

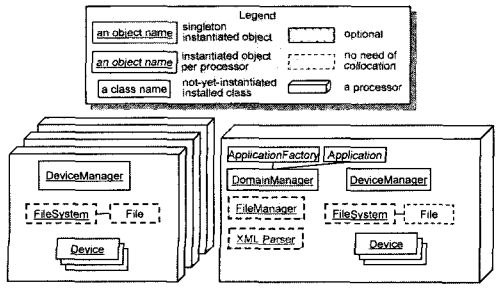


그림 9 배치 환경

DomainManager의 파일 시스템으로써 사용되며 XML parser는 도메인 프로파일로부터 정보를 뽑아내는데 사용된다.

각 프로세서는 또한 논리 디바이스 객체들을 포함한다. 이는 프로세서 자체를 나타내는 논리 디바이스 객체와 그 프로세서가 관리하는 하드웨어 디바이스들을 위한 것들로 구성된다. 이러한 대리 논리 디바이스들은 접근 가능한 이미지 파일들을 관리하고, 대상 디바이스에게 이미지를 로딩하고 로딩된 이미지가 대상 디바이스에서 수행하게 함으로써 크로스 로더로서 동작한다. 그럼으로써 논리 디바이스 객체의 역할은 [12]의 하드웨어 관리자의 역할과 유사하다. 각 프로세서에는 또한 인스턴스화된 논리 디바이스 객체들과 서비스 객체들을 관리하는 DeviceManager 객체가 하나씩 존재한다. 각 Device Manager 객체는 FileSystem 객체를 필요로 한다.

3.3.2 시동 절차

배치 상태를 복원하는 시동절차는 크게 (1) 배치 환경을 구축하고, (2) 시스템이 셧다운 될 당시 수행하던 응용 프로그램들을 활성화시키는 과정으로 이루어진다. 본 연구의 시동 과정은 도메인 프로파일의 구성 요소들이 비휘발성 저장소에 존재하는 조건을 필요로 한다. 구체적으로, DCD는 각 노드에 먼저 설치되어야 하며, 하나의 노드는 DomainManager 객체가 존재할 노드에 미리 설치된 DMD를 필요로 한다. 또한 적당한 노드들에 SAD도 먼저 설치되어 있어야 한다. 이에 더하여, 응용 프로그램이 셧다운 시점에 활성화되어 있었을 경우, 시스템이 시동되었을 때 활성화를 다시 시켜야 할 지, 말아야 할 지에 대하여 응용 프로그램이 그 성격에 따라 이 정보를 기술할 수 있는 방법이 필요하다. 이를 위해, 본 연구에서는 SAD에 *instantiated*와 *restore*의 두 가지 XML element를 추가하였다. 여기에서 *instantiated*는 응용 프로그램이 인스턴스화 되면 참으로 설정되고, 응용 프로그램이 셧다운되면 거짓으로 설정된다. 한편 *restore*는 응용 프로그램의 인스턴스화가 복구되어야 하는지 말아야 하는지를 나타내는 정적인 값이다. 이러한 조건 아래, 각 노드는 다음과 같은 시동 절차를 수행한다.

(1) DomainManager 구성:

만약 DMD가 지역 노드에서 발견이 되면, 시동 프로시저는 DomainManager 컴포넌트를 로드하고 수행시킨다. DomainManager 객체는 초기화하면서 FileManager 객체를 생성하고 이를 등록시킨다. DomainManager는 또한 이전에 설치된 응용 프로그램을 위한 ApplicationFactory 객체를

복구시키고, 복구된 ApplicationFactory 객체를 설치된 응용 프로그램 리스트 속성에 추가한다.

(2) DeviceManager 구성:
모든 노드에 존재하는 DCD를 사용하여, 시동 프로시저는 DeviceManager 컴포넌트를 로드하고 수행시킨다. DeviceManager 객체는 초기화하면서 FileSystem 객체를 인스턴스화하고 등록한다. 또한 DCD에 의하여 기술되는 논리 디바이스 객체들을 생성한다. 그리고 나서, DeviceManager 객체는 자신을 DomainManager 객체에게 등록시켜 등록된 논리 디바이스들이 DomainManager 객체에게 등록되고 DCD에 의하여 기술된대로 DomainManager에 의하여 연결되도록 한다.

(3) Application 구성:
만약 SAD가 지역 노드에서 발견되면, 시동 프로시저는 먼저 instantiated와 restore XML element의 두 값이 모두 참인지를 검사하여 응용 프로그램이 인스턴스화되어야 할지, 말아야 할지를 결정한다. 인스턴스화되어야 할 각 응용 프로그램에 대하여, 시동 프로시저는 해당 ApplicationFacgtory 객체에게 응용 프로그램을 생성하도록 요청한다.

이와 같이, 각 노드는 시동 프로시저에 의하여 도메인 프로파일 정보를 활용하여 배치 상태가 복원된다.

3.3.3 배치 절차

본 연구의 배치 모델은 배치되는 컴포넌트 타입에 따라 다른 배치 절차를 제공한다. 구체적으로, 본 연구의 배치 절차는 응용 프로그램이 설치된 후 바로 활성화되지 않고 후에 선택적으로 활성화할 수 있도록 해주는 느린 응용 활성화를 지원한다. 그러나 항상 활성화가 되어 있는 Device, DeviceManager, DomainManager, 그리고 FileSystem과 같은 SCA 코어 프레임워크의 제어나 서비스 인터페이스들을 구현하는 컴포넌트들은 설치 후 바로 인스턴스화 된다. 서버 시스템을 목적으로 하는 CCM의 배치 절차에서는 배치되는 컴포넌트들이 설치 후 바로 활성화되는 것을 주목하자. 그러나 본 연구의 SDR 내장형 시스템을 위한 프레임워크에서는 여러 독립적인 응용 프로그램들이 함께 설치되지만 선택적으로 인스턴스화된다.

설치와 활성화 과정은 사용자에 의하여 수동적으로 시작될 수도 있고, 소프트웨어 에이전트에 의하여 자동적으로 시작될 수도 있다. 설치 과정에서 대상 소프트웨어는 지역 시스템의 비휘발성 저장소로 다운로드될 수도 있고 또는 원격 시스템에 남아 있으면서 실제로 인스턴스화가 될 때 메모리로 다운로드 될 수도 있다. 이러한 소프트웨어 다운로드 절차는 본 논문의 범위 밖으로 다루지 않는다.

응용 프로그램의 설치는 DomainManager의 installApplication() 메소드로의 호출을 통해서 이루어진다. 이를 통해 설치되는 응용 프로그램을 위한 ApplicationFactory 객체 하나가 인스턴스화되고, 이는 DomainManager 객체에 등록된다. 응용 프로그램이 설치된 후, 이는 추후에 해당 ApplicationFactory 객체의 create() 메소드 콜을 통해 선택적으로 인스턴스화 될 수 있다. 그러면 인스턴스화된 응용 프로그램을 나타내는 Application 객체가 생성되고, 해당 SAD에서 기술된 컴포넌트들이 적절한 디바이스들에 로드되고 수행된다. 응용 프로그램을 구성하는 컴포넌트들이 모두 인스턴스화가 된 후, ApplicationFactory 객체는 SAD에 기술된 대로 컴포넌트들을 연결한다.

다른 논리 디바이스, 디바이스 관리자, 서비스 객체를 위한 컴포넌트들의 배치는 DomainManager 객체의 등록 메소드들을 통하여 이루어진다. 이들이 배치될 때, 즉 등록될 때, 해당 노드의 DCD 파일이 업데이트된다. 이 부류의 컴포넌트들을 업그레이드하기 위해서는 먼저 대상 컴포넌트의 비등록 작업을 거치고 교체가 되어야 한다.

응용 컴포넌트의 업그레이드는 본 연구에서 DomainManager에 추가한 replace() 메소드를 통하여 이루어진다. 인스턴스화된 응용의 업그레이드는 런타임과 느린 업그레이드의 두 가지 방식 중의 하나로 이루어진다. 느린 업그레이드에서는 먼저 대상 컴포넌트를 언인스톨하고, 새로운 패키지를 인스톨하고, 해당 SAD 디스크립터를 업데이트한다. 여기에서 업그레이드의 효과는 응용 프로그램이 다시 인스턴스화된 이후에만 나타난다. 런타임 업그레이드는 느린 업그레이드의 프로시저를 따르지만 SAD를 업데이트한 후에 느린 업그레이드와는 다르게 (1) 대상 응용 프로그램의 모든 객체의 수행을 정지시키고, (2) 대상 컴포넌트의 객체들과 다른 객체들의 연결을 끊고, (3) 교체된 컴포넌트를 인스턴스화하고, (4) 새로이 교체된 컴포넌트와 연관된 연결을 복원한다. 이러한 업그레이드 정책의 선택은 컴포넌트나 응용 프로그램의 특성에 달려있다. 이러한 이유에서 본 연구에서는 SAD와 SCD에 *upgradetype*의 XML element를 추가하였다. 이 값은 *runtime*이나 *lazy*가 될 수 있다. SAD와 SCD의 디스크립터 모두에서 이 값이 *runtime*일 때만 런타임 업그레이드가 이루어진다.

4. 제안한 프레임워크의 장점

4.1 일관적인 포트 개념

현재의 SCA 표준에서 포트의 개념은 전체 프레임워크에서 일관적이지 못하게 사용되고 있다. 구체적으로 SCA 코어 프레임워크 인터페이스에서와 도메인 프로파일에서의 포트의 개념이 일치하지 않는다. SCA 도메인 프로파일의 관점이 본 연구의 컴포넌트 모델에서의 관점과 일치한다. 본 연구에서 포트는 단순히 컴포넌트 간의 이름이 있는 연결점을 나타낸다. 그러나 SCA 코어 프레임워크 인터페이스에서 포트는 연결 기능을 제공하는 객체를 의미한다. 본 연구의 컴포넌트 모델에서는 연결기능을 제공하기 위하여 별도의 포트 객체를 사용하기 보다는, 공통 컴포넌트 인터페이스(SCAComponentObject)가 연결 기능을 제공한다. 본 연구의 컴포넌트 모델을 통해 전체 프레임워크를 통하여 포트의 개념이 일관적이게 된다. SCAComponentObject는 객체를 연결하기 위한 현 SCA의 Port와 PortSupplier 인터페이스를 불필요하게 한다.

그림 6(b)는 현 SCA에서 객체들이 어떻게 연결되는지를 보여준다. 즉 컴포넌트에 포함된 어떤 객체가 컴포넌트에 포함된 *uses* 포트를 연결하기 위하여 Port 인터페이스를 구현해야 한다. 이에 더하여 *provides* 포트로 드러나게 되는 객체들은 Resource 인터페이스에 의하여 상속되는 PortSupplier 인터페이스의 연결 메소드들을 구현하여 연결 기능을 제공해야 한다. 본 연구의 컴포넌트 모델에서는 컴포넌트 객체만이 컴포넌트의 연결을 관리하여, 포함된 객체는 응용 도메인에서 필요로 하는 기능만 구현하면 된다.

4.2 SCA 표준의 보완

본 논문에서 제안한 패키지 및 배치 모델은 SCA코어 프레임워크와 도메인 프로파일을 활용하고 있다. SCA 도메인 프로파일을 채택함으로써 하드웨어 디바이스의 구성 정보를 시스템과 연관시킬 수 있다. 본 논문에서는 표준에서 명확하지 않았던 배치 환경과 배치 절차에 대하여 SCA 코어 프레임워크의 인터페이스들을 활용하여 제시함으로써 동적 SDR 컴포넌트의 배치를 가능하게 하는 프레임워크로서 보완하였다.

제안된 컴포넌트 프레임워크는 SCA 표준을 바탕으로 하면서 SDR 컴포넌트의 동적 배치에 대한 지원을 추가하였다. 구체적으로 본 연구의 패키지 모델과 배치 모델의 시동 프로세스에서는 SCA 도메인 프로파일이 활용되었고, 제안된 배치 모델에서 배치 환경의 구성과 시동 프로세스, 배치 절차 모두에서 SCA 코어 프레임워크의 인터페이스들이 활용되었다. 또한 제안된 배치 절차에서는 SCA 도메인 프로파일과 SCA 코어 프레임워크 인터페이스에 몇몇 기능을 추가하면서 이를 다소 확장하였다. 이와 같이, 제안된 프레임워크는 완전히 새로운 것이 아니라, 현 SDR 소프트웨어 프레임워크를 최대한 유지하면서 이를 보완하였다.

4.3 내장형 시스템에 대한 특화

제안된 프레임워크는 내장형 시스템에 특화되어 설계되었다. 3.1.1절과 3.3절에서 기술하였듯이, 본 연구의 컴포넌트 모델과 배치 모델은 범용 또는 서버 시스템과 대조되는 SDR 내장형 시스템 고유의 특성을 고려하여 설계되었다. 구체적으로, 컴포넌트 모델에서 객체 관리 기능을 중앙 집중화시키는 것은 이러한 기능을 가진 객체가 설치와 컴포넌트 교체 시점에서만 incarnate되는 것을 가능하게 해준다. 이는 소프트웨어 객체가 배치 시점에 연결된 후 독립적인 응용을 구성하면서 정적인 연결을 유지하는 내장형 시스템 소프트웨어의 특성에 따른 것이다.

제안된 배치 환경에서, 각 하드웨어 디바이스를 위한 논리 디바이스는 먼저 설치되고 활성화되어야 한다. 이 또한 다양한 디바이스들이 유지되고 제어되어야 하는 내장형 시스템에 특화된 결과이다. 예를 들어, 셀프 로딩 기능이 없는 디바이스들을 위하여서, 다른 프로세서에서 인스턴스화된 논리 디바이스를 통하여 적절한 이미지가 크로스 로드되고 수행되어야 한다.

이에 더하여, 본 연구에서는 완전한 체크 포인팅이나 런타임 메모리 상태의 백업을 지원할 수 없는 제안된 저장소를 가진 SDR 내장형 시스템을 위하여 셧다운 시점의 배치 상태를 복원하기 위한 시동 절차를 제시하였다. 또한 본 연구의 배치 절차에서는 응용 컴포넌트들의 설치 절차와 인스턴스화 절차가 명시적으로 분리되어 있다. 이는 여러 파형 응용 프로그램이 함께 설치되고 경우에 따라 이들이 선택적으로 인스턴스화되는 SDR 내장형 시스템의 특성을 고려한 결과이다.

5. 결 론

SCA는 SDR 내장형 시스템의 디자인 패턴을 잘 활용한 프레임워크를 제공하고 있으나, 배치 수준에서의 소프트웨어 재사용을 위한 컴포넌트에 대한 표준을 제시하고 있지 않다는 문제점이 있다. 최근 표준으로 인정된 CCM(CORBA Component Model)은 잘 정리된 컴포넌트 프레임워크를 제시하고 있으며, SCA에서 기반으로 하고 있는 CORBA를 기반으로 하고 있지만 내장형 시스템을 위한 프레임워크가 아니기 때문에 그대로 사용될 수 없는 문제점이 있다.

본 논문에서는 SDR 컴포넌트의 동적 재배치를 지원

하는 컴포넌트 모델, 패키지 모델, 배치 모델로 구성되는SCA 기반 소프트웨어 프레임워크를 제시하였다. 본 연구의 컴포넌트 모델에서는 컴포넌트에 포함된 객체들의 연결을 위한 공통 컴포넌트 인터페이스가 제공된다. 컴포넌트는 특성화된 코바 객체로서 응용 도메인에 적용되는 기능들을 격리시키면서 객체 관리 기능들을 지원한다. 본 연구의 패키지 모델은 SCA 도메인 프로파일을 활용하면서 배치 단위의 패키징에 대하여 기술한다. 마지막으로 본 연구의 배치 모델은 (1) SCA 코어 프레임워크 인터페이스들에 기반한 배치 환경, (2) 배치 상태를 복구하기 위한 시동 프로세스, (3) 느린 응용 인스턴스화와 동적 컴포넌트 교체를 지원하는 배치 과정을 제시한다.

본 논문의 주된 기여 사항은 세 가지로 요약될 수 있다. 첫째, 본 연구는 일관적인 포트의 개념으로서 내장형 시스템에 특화된 컴포넌트 모델을 제안하였다. 둘째, 본 연구는 현 SDR 소프트웨어 표준에 근거하여 이를 보완한 배치 모델을 제안하였다. 셋째, 본 연구는 컴포넌트간의 정적 연결 관리, 배치 상태를 복구하기 위한 시동 절차의 필요성, 느린 응용 인스턴스화 정책과 방법 등의 내장형 시스템의 특성을 제기하면서 이러한 내장형 시스템에 특화된 컴포넌트 프레임워크를 제시하였다.

현재 본 연구진는 제안한 프레임워크의 유용성을 검증하기 위하여 제안된 프레임워크를 지원하는 배치 도구를 구현하고 있다. 또한 본 연구를 확장시켜 CORBA 미들웨어 위의 응용 계층의 컴포넌트 뿐만 아니라 커널 모듈이나 디바이스 드라이버와 같은 시스템 컴포넌트를 지원할 계획이다. 한편 현재 CCM과 본 논문의 모델 모두 컴포넌트들이 조합된 결과가 컴포넌트 어셈블리로서, 더 이상 컴포넌트가 아니라는 문제점이 있는데, 이를 해결하는 연구도 필요하다. 또한 이러한 컴포넌트 모델을 지원하면서 동시에 디자인 패턴을 적극적으로 활용하여, 내장형 시스템의 컴포넌트들의 런타임 환경을 제공하는 경량화된 컨테이너를 설계하는 것도 하나의 이슈이다. 아울러 컴포넌트의 비기능적 측면인 수행 시간, 고장 감내, 보안 같은 QoS에 대한 명세와 이를 검증하는 도구에 대한 연구도 고려할 만하다.

참 고 문 헌

[1] Object Management Group (OMG), http://www.omg.org.

[2] Unified Modeling Language Specification Version 1.4 Appendix B - Glossary, Object Management Group, September 2001.

[3] Software Defined Radio (SDR) Forum, http://www.sdrform.org.

[4] Joint Tactical Radio System (JTRS), http://www.jtrs.saalt.army.mil/.

[5] Software Communications Architecture (SCA) Specification MSRC-5000SCA V2.2, Joint Tactical Radio Systems, November 17, 2001, Available at http://www.jtrs.saalt.army.mil/SCA/SCA.html.

[6] The Common Object Request Broker: Architecture and Specification, Version 3.0, Object Management Group, June 2002.

[7] CORBA Component Model Version 3.0, Object Management Group, June 2002.

[8] Enterprise JavaBeans Technology, Sun Microsystems, Inc, http://java.sun.com/products/ejb/.

[9] The Distributed Component Object Model (DCOM), Microsoft Corporation, http://www.microsoft.com/com/tech/DCOM.asp.

[10] CORBA Component Model Request For Proposal (RFP), Object Management Group, June 1997.

[11] W. Emmerich and N. Kaveh, Component technologies: Java Beans, COM, CORBA, RMI, EJB and the CORBA Component Model, In Proceedings of International Conference on Software Engineering, pp. 691-692, 2002.

[12] Benjamin H. Wang, Pangan Ting, S. Charles Tsao, Hung-Lin Chou, and Nanson Huang. Integration of system software and SDR hardware platforms, SDRF-01-I-0052-V0.00, Software Defined Radio Forum Contribution, August 2001.

[13] P. J. Clemente, J. Hernandez, J. M. Murillo, M. A. Perez, and F. Sanchez. AspectCCM: an aspect-oriented extension of the corba component model, In Proceedings of Euromicro Conference, 2002.

[14] N. Wang, K. Parameswaran, M. Kircher, and D. C. Schmidt. Applying reflective middleware techniques to optimize a QoS-enabled CORBA component model implementation, In Proceedings of Computer Software and Applications Conference (COMPSAC), 2000.

[15] Nanbor Wang, Douglas C. Schmidt, and David Levine, Optimizing the CORBA Component Model for high-performance and real-time applications, In Work-in-Progress session at the Middleware 2000 Conference, ACM/IFIP, 2000.

[16] J.-L. Bakker and H. J. Batteram. Design and evaluation of the distributed software component framework for distributed communication architectures, In Proceedings of Enterprise Distributed Object Computing Workshop (EDOC), 1998.

[17] J. M. Fischer and M. D. Ercegovac. A component framework for communication in distributed applications, In Proceedings of Parallel and Distributed Processing Symposium (IPDPS), 2000.

김 세 화

1997년 2월 서울대학교 전기공학부 학사 1997년 1월~1997년 12월 서울대학교 자동화시스템공동연구소(연구원). 2000년 2월 서울대학교 전기컴퓨터공학부 석사 2000년 3월~현재 서울대학교 전기컴퓨터공학부 박사과정. 관심분야는 설계 방법론, 내장형 시스템, 실시간 시스템

홍 성 수

1986년 서울대학교 컴퓨터공학과 학사 1988년 서울대학교 컴퓨터공학과 석사 1988년~1989년 한국전자통신연구소(연구원). 1994년 University of Maryland, Department of Computer Science (Ph.D.). 1994년~1995년 University of Maryland, Department of Computer Science(Faculty Research Associate). 1995년~1995년 Silicon Graphics Inc. (Member of Technical Staff). 1995년~1997년 서울대학교 전기공학부 전임강사. 1997년~2001년 서울대학교 전기공학부 조교수. 2001년~현재 서울대학교 전기컴퓨터공학부 부교수. 관심분야는 실시간 시스템, 실시간 미들웨어, 운영체제

장 래 혁

1996년 서울대학교 제어계측공학과 박사 1997년 미국 미시간 주립대학교(앤아버소재) 연구원. 1997년~현재 서울대학교 컴퓨터공학부 조교수. 관심분야는 저전력 시스템, 내장형 시스템, 컴퓨터하드웨어