특집

SDR(Software Defined Radio)의 소프트웨어 구조

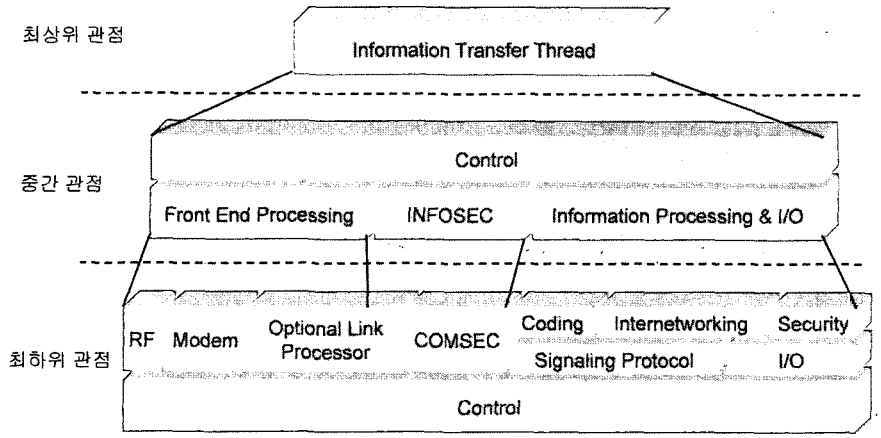
김세화, 유종훈, 홍성수(서울대학교 실시간 운영체제 연구실)

Ⅰ.서 론

전통적인 무선통신은 특정 주파수 범위의 waveform을 대상으로 설계된 하드웨어 컴포넌트로 구현되었다. 이와 대비되는 SDR(Software Defined Radio, 소프트웨어 기반 무선통신)은 다양한 waveform을 유연하게 지원할 수 있도록 소프트웨어에 의해 구현된 무선통신이다. SDR의 가장 큰 목적은 새로운 또는 여러 무선통신 기술을 하드웨어 교체 없이 사용할 수 있게 하는 것이다.

SDR은 1980년대에 그 개념이 등장하여 기지국(base station) 시스템으로 제품화되기 시작하였다. 아직까지는 하드웨어 부품의 가격과 제약 때문에 단말기에의 상업적 적용은 활발히 이루어지지 않고 있다. 그러나, 최근 군용 시스템으로 급속도로 제품화가 이루어지고 산업계 제품으로도 시제품이 나오기 시작하고 있다. FCC (Federal Communications Commission, 미국 연방통신위원회)는 2004년 11월에 처음으로 SDR을 승인하였다. FCC는 이어 2005년 9월에는 산업계에 SDR의 사용을 인가하고 동시에 Cisco의 Wi Fi (802.11a) SDR을 최초의 산업계 SDR 제품으로 승인하였다. 이 Wi Fi SDR 기술은 무선 랜 채널을 확장하여 대역폭과 용량을 업그레이드 시킬 수 있게 해준다. 이와 같이 SDR은 단순히 하나의 단말기로 여러 무선통신을 동시에 사용하는 것뿐만 아니라, 가용한 주파수를 찾아 대역폭을 확장할 수 있는 길도 열어준다. 이 외에도 향후 SDR을 활용한 새로운 혁신적인 무선통신 제품의 개발이 기대된다. FCC의 SDR 채용으로 인해 미국에서의 SDR의 상업화가 가속화될 것은 자명하며, 이러한 추세로 볼 때 SDR이 군용 시스템은 물론 산업용 시스템 및 개인용 단말기로 확산될 날도 그리 멀지 않은 것으로 보인다.

SDRF(SDR 포럼)[1]는 전세계에서 실질적인 SDR 표준화 기관으로 인정받고 있고 있는 기관이다. SDR 포럼은 1996년에 결성되었으며, 1998년 이전에는 MMITS(Modular Multifunction Information Transfer System) 포럼으로 불리었다. SDRF은 먼저 SDR 시스템의 기술 개발 촉진을 위해 SDR 시스템을 기능적 관점에서 정의하는 작업을 추진하였다. 그 결과 그림 1과 같은 SDR 소프트웨어 참조 모델[2]을 제시하였다.

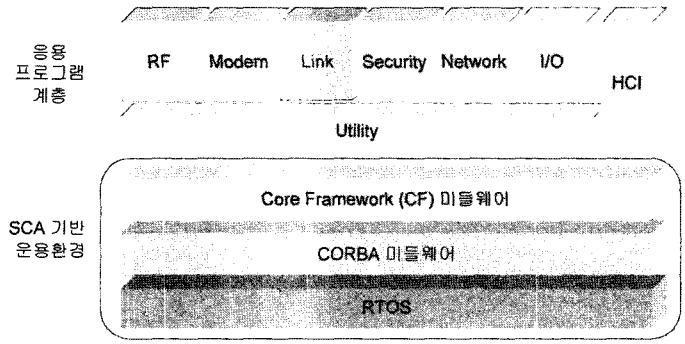


〈그림 1〉 SDR 소프트웨어 참조 모델 (상위 수준 기능 모델)

SDR 소프트웨어 참조 모델은 SDR 소프트웨어가 수행할 수 있는 다양한 기능적 역할들을 나타내며 추상화 정도에 따라 그림 1에서와 같이 세 가지 관점으로 정의된다. 가장 추상적인 관점을 갖는 최상위 관점에서 SDR 소프트웨어는 정보 전송 쓰레드의 역할을 하는 것으로 기술된다. 여기에서 왼쪽 측면의 입력 인터페이스는 전파(air) 인터페이스이며, 오른쪽 측면의 출력 인터페이스는 유선과 사용자 인터페이스이다. 다음 수준의 중간 관점은 네 가지 SDR의 핵심 기능들의 순서가 있는 흐름을 규명한다. 그림에서와 같이 이는 (1) 전단 (front end) 처리, (2) 정보 보안 (information security), (3) 정보 처리, (4) 제어로 이루어진다. 여기에서의 처리는 모두 양방향(받기와 보내기)으로 이루어질 수 있다. 이들은 다시 최하위 관점의 기능 모델로 그림에서와 같이 세부 기능으로 나누어진다. 예를 들면, 정보 처리는 경로 선택, 멀티플랙싱, 소스 코딩, 시그날링 프로토콜, I/O 기능 등으로 매핑된다. 여기에서 처리하는 정보는 데이터일 수도 있고 제어 정보일 수도 있다.

SCA(Software Communication Architecture)[4]는 이러한 SDR 소프트웨어 참조 모델을 기반으로 하여 설계된 소프트웨어 프레임워크이다. SCA는 미국방성 JTRS(Joint Tactical Radio System)[3]에서 개발된 것으로 전세계 산업계로부터 SDR의 실질적인 소프트웨어 표준으로 인정되고 받고 있다. SDRF는 SCA를 SDR 소프트웨어 표준으로서 그대로 채택하였다. SCA는 waveform의 이식성을 극대화하기 위하여 응용 프로그램들을 위한 공통 운용환경(common operating environment)을 제시할 뿐만 아니라, 응용 프로그램을 구성하는 컴포넌트들의 인터페이스를 위한 표준도 포함하고 있다.

본 고에서는 이러한 SCA 표준에 기반한 SDR 시스템의 소프트웨어 구조에 대하여 논한다. 먼저 SCA에 따른 SDR 시스템의 소프트웨어 구조에 대하여 논한다. 이어 SDR 소프트웨어를 이루는 두 계층인 (1) SCA 기반 운용환경과 (2) 응용 프로그램 계층 각각에 대하여 논하고 결론을 맺는다.



〈그림 2〉 SCA(Software Communication Architecture)에서 제시한 SDR 소프트웨어 구조

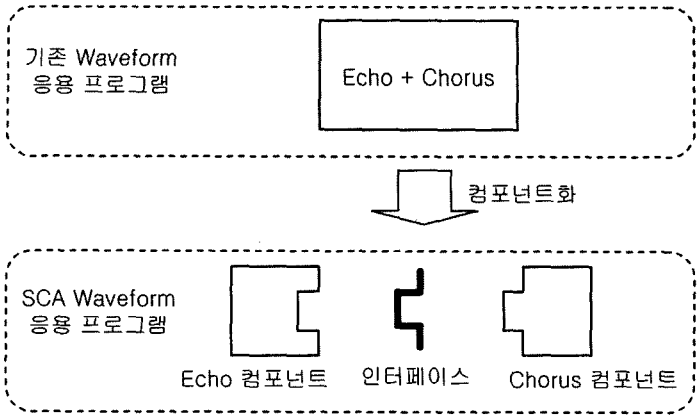
II. SCA 기반 소프트웨어의 구조

본 장에서는 SDR 소프트웨어 표준인 SCA에 기반한 SDR 소프트웨어의 구조에 대하여 개관한다. SDR 시스템은 내장형 무선 응용 시스템이고 자원 제약성을 가지고 있으며 실시간 작업처리를 필요로 한다. 과거에는, 그리고 가끔은 지금도, 소프트웨어의 용량을 줄이기 위해 비교적 간단한 내장형 시스템에서는 종종 저급 언어를 사용하여 하드웨어 구동을 비롯한 다양한 기능을 하나의 일체형(monolithic) 소프트웨어로 구현한다. 그러나 이러한 방식은 소프트웨어의 복잡성 문제를 해결할 수 없으며, 결과적으로 time to market을 만족시킬 수 없게 되어 이른바 소프트웨어의 위기 문제를 가져오게 된다.

이러한 문제를 해결하기 위해서는 SDR 소프트웨어를 계층적으로 구성할 필요가 있다. JTRS SCA에서는 이에 따라 그림 2와 같이 여러 계층으로 구성된 소프트웨어 구조를 채택하고 있다. 먼저 SDR 소프트웨어를 크게 두 계층으로 구분하면 (1) SDR 무선 응용 기능을 담당하는 응용 프로그램 계층과 (2) 이를 지원하는 운용환경 계층으로 나눌 수 있다.

먼저 응용 프로그램 계층은 SDR 소프트웨어 참조 모델에서 기술하는 다양한 기능들을 구현하기 위해서 SDR 시스템은 여러 하드웨어 노드들로 구성된 분산 시스템으로 설계되어야 한다. 이러한 분산 시스템 상에서 SDR 소프트웨어 시스템을 구현하려면 재구성성과 유연성이 필요하며 따라서 이를 충족시켜줄 수 있는 컴포넌트 기반 컴퓨팅 기술이 필요하게 된다. 즉, SDR 소프트웨어의 구조는 기본적으로 분산 컴포넌트 기반 소프트웨어 구조가 되는 것이다.

다음으로 운용환경 계층은 코어 프레임워크(Core Framework: CF) 미들웨어, CORBA(Common Object Request Broker Architecture)[5] 미들웨어, RTOS(Real Time Operating Systems, 실시간 운영체제)의 계층 구조로 다시 나누어진다. 여기에서 SDR 응용 프로그램의 컴포넌트 기반 컴퓨팅을 가능하게 해주는 것이 코어 프레임워크 미들웨어이다. 코어 프레임워크 미들웨어는 도메인 프로파일이라는 XML 디스크립터 정보를 활용



〈그림 3〉 컴포넌트 소프트웨어의 기본 개념

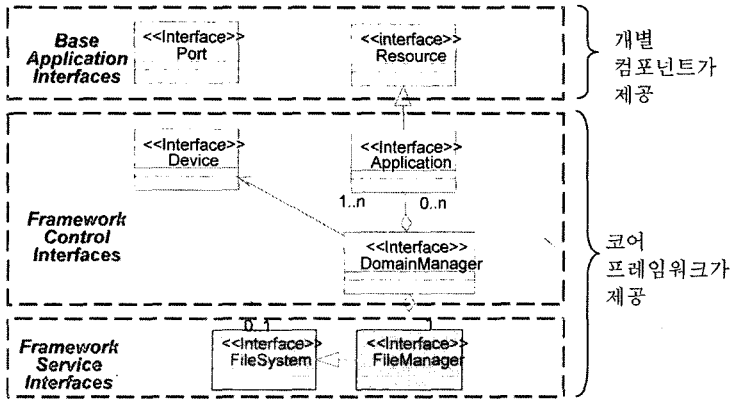
하여 컴포넌트들을 배치하고 연결하여 응용 프로그램이 동적으로 재구성될 수 있는 환경을 제공한다. 코어 프레임워크 미들웨어가 관리하는 컴포넌트들은 크게 (1) 하드웨어를 추상화하는 디바이스 컴포넌트들과 (2) waveform 소프트웨어를 구성하는 소프트웨어 컴포넌트들로 나뉘어진다. 코어 프레임워크 미들웨어의 하부 계층을 구성하는 CORBA 미들웨어와 RTOS는 여러 노드들로 이루어진 분산 환경과 다양한 하드웨어 자원을 관리하고 추상화시켜주는 기능을 제공한다. 이들 운용환경 계층은 SDR 무선 응용의 시간 제약 조건이 만족될 수 있도록 실시간성을 지원하여야 하며, SDR 내장형 시스템에 사용될 수 있도록 경량성을 갖추어야 한다. 다음 장에서 이러한 SCA 기반 운용환경의 구조에 대하여 논하고, 이어서 응용 프로그램 계층의 구조에 대하여 논한다.

III. 운용환경의 구조

본 장에서는 SDR 시스템을 구성하는 운용환경의 구조를 논한다. 이하 하부 절에서 SCA 기반 운용환경의 세 가지 구성 요소인 (1) 배치 미들웨어인 코어 프레임워크(CF, Core Framework), (2) 분산 미들웨어인 CORBA 미들웨어, 그리고 (3) RTOS 각각에 대하여 살펴본다.

1. 코어 프레임워크 미들웨어

컴포넌트 소프트웨어 기술은 밑바닥부터 소프트웨어를 개발하는 대신 배치 수준의 소프트웨어를 조합하여 새로운 소프트웨어를 생성하는 것을 목적으로 하고 있다. 컴포넌트 소프트웨어 기술은 소프트웨어의 재사용성을 극대화시켜 내장형 소프트웨어의 복잡성 문제를 해결하고 time to market을 줄이는데 크게 도움이 된다. SCA에서는 컴포넌트 소프트웨어 기술을 지원하는 미들웨어로서 코어 프레임워크를 정의하고 있다. 코어 프레임워크 미들웨어는 도메인 프로파일이라는 XML 디스크립터 정보를 활용하여 컴포넌트들을 배치하고 연결하여 응용 프로그램이 동적으로 재구성될 수 있는 통합 환경을



〈그림 4〉 코어 프레임워크의 인터페이스

제공한다.

컴포넌트의 의미는 기존의 많은 문서와 논문 등에서 여러 가지의 의미로 사용되어 혼동을 일으켰는데, 본 고에서는 OMG(Object Management Group)[8]에서 정한 컴포넌트의 정의에 따른다. 컴포넌트는 모듈화되고 배치와 교체가 가능한 시스템의 일부로서 구현을 숨기고 인터페이스의 집합을 드러낸다. 컴포넌트는 보통 공유 라이브러리의 형태이지만 배치 환경에 따라 바이너리, 바이트 코드, 또는 스크립트 언어로 작성된 소스 파일일 수도 있다.

그림 3은 컴포넌트 소프트웨어의 기본적인 개념을 보여 주고 있다. Echo와 Chorus라는 두 가지 기능을 갖는 기존의 waveform 응용 프로그램이 컴포넌트화되어 독립적인 기능을 갖는 두 개의 컴포넌트로 분할되었다. 컴포넌트 내부가 어떻게 구현되었는지는 외부에 감추어 지고 오직 정해진 인터페이스만이 드러나고 있다. 코어 프레임워크는 이와 같은 인터페이스의 정의들로 구성되며 이를 통해 여러 개의 응용 프로그램 컴포넌트를 분산 시스템의 각 프로세싱 노드에 배치하고 설치와 제거, 시작과 정지 등의 관리 작업을 수행한다. 인터페이스는 구체적으로 CORBA 인터페이스로 구현이 되며 CORBA IDL(Interface Definition Language)에 의해 정의된다. 한편 응용 컴포넌트 정보와 컴포넌트간 연결 정보는 도메인 프로파일이라고 하는 XML로 씌어진 디스크립터에 기술되며 이를 통해 컴포넌트를 분산 노드에 동적으로 배치 및 연결하는 것이 가능해진다.

본 장의 나머지 절에서는 컴포넌트 프레임워크의 핵심을 이루는 코어 프레임워크 인터페이스와 도메인 프로파일에 대하여 차례로 설명한다.

가) 프레임워크 인터페이스

그림 4는 코어 프레임워크의 핵심적인 인터페이스 구조를 보여 준다. 코어 프레임워크의 인터페이스는 기본 응용 프로그램(Base Application), 프레임워크 제어(Framework Control), 그리고 프레임워크 서비스(Framework Service)의 세 가지 그룹으로 분류된다.

개별 응용 컴포넌트 소프트웨어는 기본 응용 프로그램 인터페이스를 구현하여 제공해야 한다. Resource 인터페이스는 각 응용 컴포넌트가

하나씩 구현해야 하며 코어 프레임워크는 이를 이용해 응용 프로그램의 설치, 제거, 실행, 정지 등의 관리 작업을 수행한다. Port 인터페이스는 응용 컴포넌트간 통신을 위해 구현하며 코어 프레임워크는 도메인 프로파일의 정보를 이용하여 서로 통신하고자 하는 컴포넌트들의 Port를 연결해 준다.

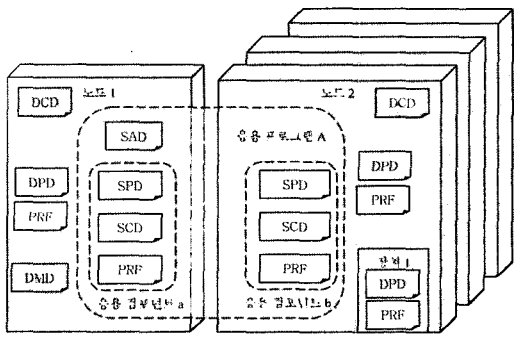
코어 프레임워크는 프레임워크 제어 및 프레임워크 서비스 인터페이스를 응용 컴포넌트에게 제공한다. 먼저 코어 프레임워크의 핵심 인터페이스인 DomainManager는 전체 시스템에서 단 한 개가 존재하며 응용 프로그램을 설치, 제거하고 장치를 등록, 해제하기 위한 인터페이스를 제공한다. Application은 사용자가 새로운 응용 프로그램을 설치했을 때 코어 프레임워크 내에 자동으로 생성되며 응용 프로그램 및 응용을 구성하는 각 컴포넌트(Resource)의 관리를 수행한다. Device는 하드웨어 장치를 관리하기 위한 인터페이스로서 단순한 디바이스 드라이버일 수도 있고 GPP, DSP, FPGA와 같은 프로세싱 노드에 대한 논리적 장치에 대한 인터페이스를 제공할 수도 있다.

마지막으로 프레임워크 서비스 인터페이스의 파일 관리자(FileManager)는 파일 시스템에 대한 인터페이스를 제공하며 여러 개의 분산 파일 시스템을 동일한 주소 공간에 있는 것처럼 관리할 수 있게 해 준다.

나) 도메인 프로파일

시스템의 하드웨어와 소프트웨어의 구성 정보는 도메인 프로파일이라는 별도의 텍스트 파일로 저장된다. 코어 프레임워크는 이 정보를 통해 시스템 동작 중에 동적으로 컴포넌트의 배치 및 자원 관리를 수행한다. 또한 응용 프로그램 개발자 및 관리자는 이를 통해 구성 정보를 쉽게 판독 및 수정할 수 있다. 도메인 프로파일은 XML로 작성되며 그림 5와 같은 구성 요소로 이루어져 있다.

SAD(Software Assembly Descriptor)는 하나의 응용 프로그램을 기술하는 핵심적인 도메인 프로파일이다. SAD는 응용 프로그램 내에 어떤 응용 컴포넌트들이 있으며 이들이 서로 어떻게 연결되는지를 기술한다. 응용 프로그램 내의 개별 응용 컴포넌트들은 SPD, SCD, PRF에 의해 기술된다. 먼저 SPD(Software Package Descriptor)에는 컴포넌트의 실제 파일 위치, 플랫폼, 런타임 환경, 메모리와 CPU 요구량 등의 각종 정보가 기술되어 있다. CORBA 컴포넌트의 형태로 구현된 응용 프로그램은 추가로 SCD(Software Component Descriptor)를 필요로 한다. 여기에는 어떤 타입의 CORBA 인터페이스를 사용하고 혹은 제공하는지에 대해 기술되어 있다. PRF(Properties Descriptor)는 컴포넌트의 설정 가능한 옵션을 기술하기 위한 도메인 프로파일이다. 이를 통해 SCA 코어 프레임워크 미들웨어



DMD: DomainManager Configuration Descriptor
SAD: Software Assembly Descriptor
SPD: Software Package Descriptor
PRF: Properties Descriptor
DCD: Device Configuration Descriptor
DPD: Device Package Descriptor

〈그림 5〉 도메인 프로파일의 구성요소

에게 특정 컴포넌트가 제공하는 옵션의 이름, 타입, 기본값, 그리고 설명 등을 알릴 수 있다.

각 노드의 구성 정보는 DCD(Device Configuration Descriptor)에 의해 기술되며 각 노드들은 DCD를 이용하여 자신의 Device들을 로딩함으로써 SCA 코어 프레임워크를 부팅시킨다. 노드 및 노드에 속해 있는 하드웨어 장치들은 DPD(Device Package Descriptor)에 의해 기술되며 여기에는 실제 하드웨어 장치의 종류, 시리얼 넘버, 메모리 크기와 같은 정보가 들어간다. 각 Device의 옵션도 소프트웨어 컴포넌트(Resource)와 마찬가지로 PRF에 의해 기술될 수 있다.

마지막으로 DMD(DomainManager Configuration Descriptor)는 DomainManager에 대한 정보를 기술하기 위해 필요하다. DomainManager가 실행되어야 할 위치 정보 및 DomainManger가 사용하는 코어 프레임워크의 서비스 등의 정보가 DMD에 기술된다.

2. CORBA 미들웨어

SDR 시스템은 GPP, DSP, FPGA 등 여러 개의 노드로 구성되는 분산 시스템의 특징을 가진다. 분산 시스템에서는 단일 노드 시스템에서는 존재하지 않았던 많은 문제점들을 해결하여야 하며 이를 위해 CORBA가 코어 프레임워크의 기반 미들웨어로 채택되었다. CORBA (Common Object Request Broker Architecture)는 OMG(Object Management Group)[8]에 의해서 정의된 분산 미들웨어의 표준이며 현재 분산 실시간 환경을 지원하기 위한 RT CORBA 기능이 포함되어 있는 버전 3까지 정의되어 있다.

분산 환경에서 발생하는 문제점 중 가장 중요한 것은 분산 시스템을 구성하는 노드들의 이기종성이다. 이기종성은 하드웨어, 운영체제, 네트워크 프로토콜, 프로그래밍 언어 등의 다양성을 의미한다. 이런 문제점은 CORBA와 같은 분산 미들웨어를 통해 효과적으로 해결될 수 있다. 즉, CORBA에서 공통적으로 제공하는 인터페이스를 사용하면 한 번만 프로그램을 작성해도 그 프로그램이 어떤 종류의 프로세싱 노드에서 수행되던지 같은 방식으로 동작하도록 만들 수 있다.

분산 환경의 이기종성 이외에도 해결해야 할 많은 문제들이 있는데 그 중 대표적인 것은 통신하고자 하는 개체가 어떤 노드에 있는지 찾을 수 있어야 한다는 것이다. 이는 분산 시스템에서는 어떤 개체(프로그램 혹은 객체)가 특정 노드에서 수행되는 것이 정적으로 결정되는 것이 아니라, 시스템의 상황에 따라 동적으로 변화하기 때문이다. 또한, 상대 노드가 작동 불능일 경우에 대비하여야 한다. 분산 시스템에서는 네트워크 문제나, 노드의 하드웨어 문제 등과 같은 다양한 이유들로 인하여, 상대 노드와의 통신이 불가능할 경우가 종종 발생한다. 이런 경우 분산 미들웨어는 유연하게 대처할 수 있어야 한다. CORBA에서는 이러한 문제들을 해결해 주기 위하여 Naming Service, Event Service, Timing Service 등 다양한 서비스를 제공해 준다.

3. RTOS

SDR 시스템은 실시간 시스템의 속성을 가지고 있다. CDMA와 같은 무선 통신 프로토콜은 메시지들이 미리 정해진 시간 간격에 맞추어 보내어지고 받아져야 한다는 조건을 가지고 있으므로 경성 실시간 시스템이라고 할 수 있다. 한편 디코딩 코덱 처리와 같은 부분은 연성 실시간 시스템의 속성을 가진다. 코덱을 통해 부호화된

음성 데이터를 디코딩하는 과정은 시간 제약이 있기는 하지만, 그 제약을 어겼다고 해서 시스템을 못 쓰게 되는 것은 아니기 때문이다.

RTOS는 이러한 실시간 시스템을 위한 OS로 실시간 응용들이 제한된 시간 내에 올바른 결과를 생성할 수 있도록 지원하는 역할을 한다. 특히 앞서 설명한 코어 프레임워크 및 RT CORBA의 기능을 지원하기 위해서는 RTOS가 필수적이다. RTOS가 시스템에서 하는 주된 역할은 실시간 태스크의 지원과 하드웨어 특성의 추상화, 효율적인 자원 관리 등이다. 아울러 RTOS는 파일 시스템, 메모리 할당, 네트워크 프로토콜 제공 등의 다양한 역할을 수행한다.

SDR 시스템의 실시간 특성을 만족시키기 위해 SCA는 운영 체제로서 POSIX .13의 PSE52 Realtime Controller System profile[6], [7]의 요구사항을 만족하는 RTOS를 사용할 것을 규정하였다. 여기에서는 (1) 다중 스레딩, (2) 완전 선점성, (3) 우선순위 기반 스케줄링, (4) 주기 태스크 지원, (5) 한정된 인터럽트 지연 시간, (6) 한정된 스케줄링 지연 시간, (7) 실시간 자원 동기화 기법의 지원, (8) 자원 관리와 같은 기능을 지원한다.

Ⅳ. 응용 프로그램 계층의 구조

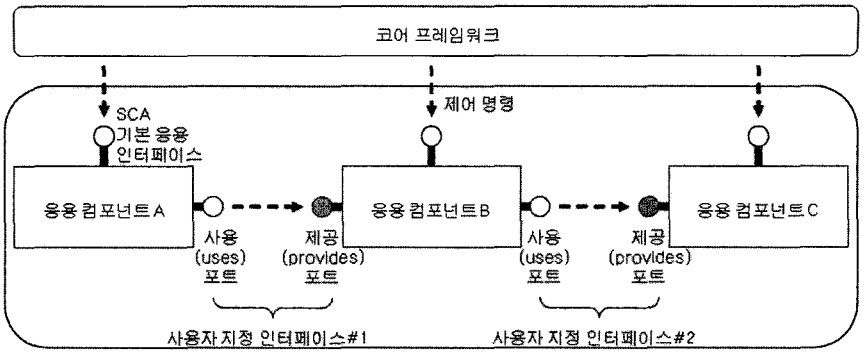
SDR 시스템의 응용 프로그램은 일반적으로 신호 처리, 패킷 입출력, 사용자 인터페이스, 보안, 라우팅 등의 다양한 종류의 컴포넌트들로 구성된다. 응용 컴포넌트는 SCA 명세에 의해 기본적으로 CORBA 컴포넌트로 구현되며 앞서 설명한 운용환경의 서비스를 이용한다. 또한 CORBA를 이용하지 않은 기존의 컴포넌트에 대해서도 별도의 어댑터를 사용하여 호환성을 유지하도록 하고 있다. 이 장의 나머지 절에서는 SDR 시스템의 응용 프로그램의 컴포넌트화 및 개별 컴포넌트의 구현 과정에 대하여 설명한다.

1. 응용 프로그램의 컴포넌트화

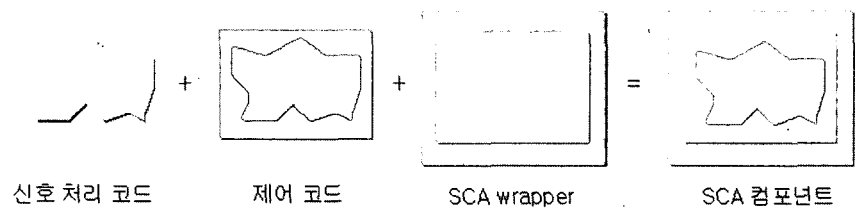
SDR 응용 프로그램의 작성을 위하여 먼저 하여야 하는 일은 응용 프로그램의 기능을 정의하고 각 기능을 분할하여 모듈을 추출하는 작업이다. 이 때 모듈은 결국 하나 하나의 SCA 응용 프로그램 컴포넌트가 되는데, 따라서 최대한 재사용성이 높으면서 독립적으로 배포 가능하도록 모듈화가 이루어져야 한다.

다음으로 하여야 할 일은 나누어진 컴포넌트 간의 사용자 지정 인터페이스를 정하는 작업이며 이는 기존의 함수나 메시지 호출과 같은 방법 대신 CORBA의 IDL 형태로 기술되어야 한다. 응용 컴포넌트는 Port 인터페이스를 통해 사용자 지정 인터페이스를 구현하는데 제공(provides) 포트와 사용(uses) 포트의 두 가지 종류가 있다. 이와 같은 이름은 포트를 제공하는지 혹은 사용하는지에 따라 결정되며 실제 데이터의 이동은 사용 포트에서 제공 포트로 흘러 가게 된다.

그림 6은 이와 같은 과정을 거쳐 추출된 응용 프로그램의 예를 나타낸 것이다. 그림에서 하나의 신호 처리 응용 프로그램이 세 개의 응용 컴포넌트로 분할되어 동작한다. 응용 컴포넌트는 다른 응용 컴포넌트와 데이터를 교환하기 위해 제공 및 사용 포트를 사용하며 왼편에서 오른편으로 데이터가 흘러 간다. 각 응용 컴포넌트는 SCA 기본 응용 인터페이스를 구현하고 있으며 이를 통해 코어 프레임워크에 의해 제어된다.



〈그림 6〉 컴포넌트화된 응용 프로그램의 예



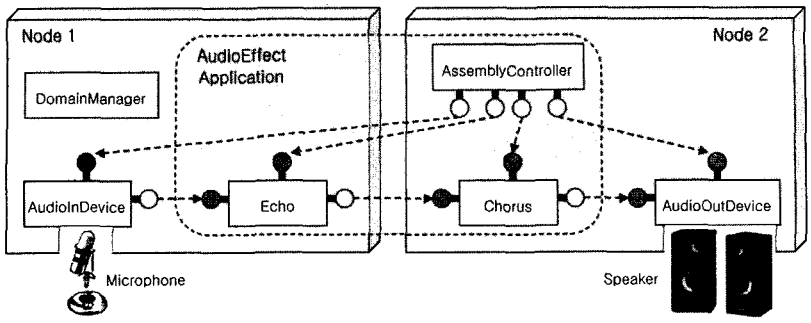
〈그림 7〉 개별 응용 컴포넌트의 구성

2. 응용 컴포넌트의 구현 및 패키징

응용 컴포넌트는 SCA 코어 프레임워크에서 정의한 기본 응용 인터페이스를 구현해야 하며 일반적으로 CORBA 컴포넌트의 형태로 구현되어야 한다. 이를 위해 응용 컴포넌트는 컴포넌트 본연의 임무를 수행하는 코드에 제어 코드 및 wrapper 코드가 합쳐져 이루어진다. 제어 및 wrapper 코드는 기본 응용 인터페이스에서 정의한 인터페이스를 구현하는 것을 목적으로 한다. 그림 7은 응용 컴포넌트 구현의 구성 요소를 나타낸 것이다.

기본 응용 인터페이스 및 사용자 지정 인터페이스는 IDL의 형태로 기술되며 이를 입력으로 하여 CORBA IDL 컴파일러에 의해 자동으로 생성된 CORBA stub 코드를 wrapper 코드라고 한다. 제어 코드는 wrapper 코드와 신호 처리 코드를 연결하여 주는 코드로서 기본 응용 인터페이스 및 사용자 지정 인터페이스의 실질적인 동작을 구현한다. 이는 신호 처리 코드와 마찬가지로 사용자가 직접 작성해야 하는 부분이며 일반적으로 응용 컴포넌트의 작성이라 함은 이 제어 코드의 작성을 의미한다. 마지막으로 개별 응용 컴포넌트를 기술하는 도메인 프로파일의 XML 디스크립터, 즉 SPD, SCD, PRF 파일을 작성하면 하나의 응용 컴포넌트가 완성된다.

응용 프로그램 개발자는 이렇게 작성된 응용 컴포넌트들 외에 조립하고 다른 컴포넌트들을 관리하는 AssemblyController라는 특수한 컴포넌트를 추가한다. 응용 프로그램의 시작과 정지와 같은



〈그림 8〉 AudioEffect 응용 프로그램 예제

제어 흐름에 따라 각 컴포넌트가 어떻게 동작하여야 할지를 도메인 프로파일로 기술하기는 매우 어렵다. 따라서 시작과 정지와 같은 명령은 코어 프레임워크로부터 AssemblyController 컴포넌트 하나에게만 전달되며 AssemblyController는 이를 적절히 각 컴포넌트에게 전달한다. 여기에 컴포넌트간 연결 관계를 기술하는 도메인 프로파일인 SAD를 추가하고 모든 파일을 묶어 하나의 배포 가능한 응용 프로그램으로 패키징한다.

3. SDR 응용 프로그램 예제

본 절에서는 앞서 설명한 응용 프로그램 작성의 이해를 돕기 위해 AudioEffect 응용 프로그램의 예를 들어 설명한다. AudioEffect는 마이크를 통해 사용자의 목소리를 입력으로 받아 Echo와 Chorus 처리를 한 후 다시 스피커로 소리를 출력하는 waveform 소프트웨어이다. 이 때 마이크로부터 스피커까지의 데이터 흐름은 실시간으로 이루어진다. 그림 8은 AudioEffect 응용 프로그램의 내부 구조를 보여준다.

AudioEffect 응용 프로그램은 총 세 개의 소프트웨어 컴포넌트인 AssemblyController, Echo, Chorus로 구성되며 그림 8에서는 두 개의 노드에 나뉘어 배치되어 있다. Echo와 Chorus는 샘플링된 음성 데이터를 받아 실시간으로 에코(echo)와 코러스(chorus) 처리를 하는 컴포넌트이다. 두 컴포넌트는 각각 음성 데이터를 입력으로 받기 위한 제공 포트와 처리한 데이터를 출력으로 내보내는 사용 포트, 그리고 AssemblyController의 제어를 받기 위한 사용 포트를 가지고 있다. AssemblyController는 개별 컴포넌트를 제어하는 컴포넌트로서 다른 컴포넌트들에 대해 사용 포트를 가지고 있다.

AudioEffect는 마이크에 대한 논리적 하드웨어 장치 컴포넌트인 AudioInDevice와 스피커에 대한 논리적 하드웨어 장치 컴포넌트인 AudioOutDevice를 필요로 한다. 이들은 각각 PC 환경의 마이크 및 사운드 카드 장치를 감싸고 있는 논리 장치로 볼 수 있다. AudioInDevice는 사용 포트로 마이크에서 받은 샘플링 데이터를 내보내고 AudioOutDevice는 제공 포트로 받은 데이터를 스피커로 출력하게 된다. 그밖에 데이터의 입출력을 제어하는 신호를 받기 위해 제공 포트를 한 개씩 갖고 있다.

AudioEffect 내의 컴포넌트 간 연결 구조는 크게 실시간 데이터 통신을 위한 연결과 비실시간 제어를 위한 연결로 나누어 볼 수 있다. 실시간 데이터

통신 연결은 AudioInDevice 로부터 Echo와 Chorus 컴포넌트를 거쳐 다시 AudioOutDevice 까지에 해당하는 연결로 실제 음성 데이터가 흘러간다. 비실시간 제어 연결은 AssemblyController 컴포넌트를 중심으로 각 소프트웨어 컴포넌트와 하드웨어 컴포넌트로 향하는 연결로 시작과 종료 등의 제어 신호 전달을 위해 사용된다.

V. 결 론

본 고에서는 SCA 표준에 기반하여 SDR 시스템의 소프트웨어 구조에 대하여 논하였다. SDR 시스템은 계층적인 소프트웨어의 구조를 따르는 동시에 분산 컴포넌트 기반 소프트웨어 구조를 따른다. 우선 계층적인 소프트웨어 구조는 자원 제약과 실시간성을 가진 SDR 무선 응용 소프트웨어의 복잡성 문제를 해결하기 위해 채택된 구조이다. SCA 표준에 기반하여 SDR 소프트웨어는 크게 응용 프로그램과 운용환경의 두 계층으로 구성된다.

분산 컴포넌트 기반 소프트웨어 구조는 SDR 시스템이 분산 시스템이면서 재구성성과 유연성을 필요로 하기 때문에 채택된 구조이다. SDR 소프트웨어 참조 모델에 기술된 바와 같이 SDR 시스템은 다양한 기능을 구현하기 위하여 통상 분산 시스템으로 구성된다. 또한 SDR 시스템은 소프트웨어에 의해 waveform이 동적으로 교체될 수 있어야 하기 때문에 소프트웨어의 재구성성과 유연성이 필수적이다. 이를 위해 필요한 기술이 컴포넌트 소프트웨어 기술이며, 이를 위해 SCA에서 운용 환경 계층의 최상층 계층으로 정의된 것이 코어 프레임워크 미들웨어이다. 코어 프레임워크 미들웨어는 응용 프로그램을 구성하는 컴포넌트들을 동적으로 배치하고 연결하여 상호 동작하게 함으로써 소프트웨어의 재사용성을 극대화할 뿐만 아니라 SDR의 핵심인 소프트웨어적인 재구성성을 가능하게 해준다. SCA 운용 환경은 코어 프레임워크 미들웨어에 더하여 CORBA 미들웨어와 RTOS로 구성된다. CORBA 미들웨어와 RTOS는 각각 분산 컴퓨팅과 자원 관리 기술을 지원한다.

SDR은 더 이상 실험실 안이나 군용 시스템을 위한 기술이 아니라, 상업적 기술로 변모하고 있다. SDR은 단순히 하나의 단말기로 여러 무선 통신을 동시에 사용하는 것뿐만 아니라, 가용 주파수 대역을 늘리는 등 다양한 혁신적인 응용이 기대되고 있다. 이에 세계 유수의 선진국들에서 국가적 차원의 프로젝트로서 SDR 기술 확보에 매진하고 있다. SDR의 소프트웨어 표준인 SCA는 이러한 SDR 소프트웨어의 구조를 결정하며 SDR 기술의 핵심을 이룬다.

참고문헌

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

[2] SDR Forum, Architecture and Elements of Software Defined Radio Systems as Related to Standards, SDRF Technical Report 2.1, November 1999.

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

[4] 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.

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

[6] ISO/IEC Std 9945 1, ANSI/IEEE Std 1003.1, POSIX (Portable Operating System Interface) Part 1:

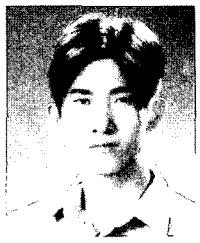
System Application Program Interface, July 1996.
[7] ISO/IEC ISP 15287 2, IEEE Std 1003.13, Information Technology Standardized Application Environment Profile POSIX Realtime Application Support (AEP), February 2000.
[8] OMG (Object Management Group), Minimum CORBA Specification Version 1.0, formal/02 08 01, August 2002.

저자소개

김 세 화

1997년 서울대학교 전기공학부 졸업(학사)
1997년-1998년 서울대학교 자동화시스템공동연구소(연구원)
2000년 서울대학교 전기컴퓨터공학부 졸업(공학석사)
2006년 서울대학교 전기컴퓨터공학부 졸업(공학박사)
2006년-현 재 서울대학교 자동화시스템공동연구소(박사후 연구원)
주관심분야 내장형 시스템 소프트웨어, 실시간 운영체제, 내장형 미들웨어, 객체지향 설계방법론

저자소개



유 종 훈

2001년 KAIST 전기및전자공학과 졸업(학사)
2005년-현 재 서울대학교 전기컴퓨터공학부 석사과정
주관심분야 내장형 시스템 소프트웨어, 실시간 운영체제, 내장형 미들웨어

홍 성 수

1986년 서울대학교 컴퓨터공학과(학사)
1988년 서울대학교 컴퓨터공학과 졸업(공학석사)
1994년 University of Maryland. Department of Computer Science (공학박사)
1988년-1989년 한국전자통신 연구소(연구원)
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-현 재 서울대학교 전기컴퓨터공학부 부교수
주관심분야 실시간 운영체제, 내장형 시스템 개발 방법론, 내장형 미들웨어