SDR을 위한 RTOS와 내장형 미들웨어의 설계

서울대학교 전기 · 컴퓨터공학부 교수 홍 성 수

차 례

1. 서론

고성능 DSP와 고집적 FPGA가 등장하게 되어 무선 통신 기능을 소프트웨어적으로 구현하는 SDR의 실현이 기술적으로 가능해지고 있다. 아직까지는 하드웨어 부품의 가격과 제약 때문에 단말기보다는 기지국(base station)에서만 SDR이 적용되고 있지만, 더욱더 다양한 무선 환경에서 내장형 시스템들이 복잡하고 유연한 기능을 가진 응용을 수행하여야 하기 때문에 이동 단말기에서도 머지 않아 SDR 기술이 적용될 것이라고 기대된다.

SDR 시스템은 내장형 무선 응용 시스템이고 자원 제약성을 가지고 있으며 실시간 작업 처리를 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. 먼저 SDR 소프트웨어를 크게 두 계층으로 구분하면 (1) SDR 무선 응용 기능을 담당하는 응용 소프트웨어와 (2) 이를 지원하는 시스템 소프트웨어 계층으로 나눌 수 있다. SDR은 소프트웨어의 재구성성이 핵심이므로, SDR을 지원하기 위해 시스템 소프트웨어가 제공해야 할 기능은 (1) SDR에서 사용되는 다양한 하드웨어를 추상화시켜 SDR 응용을 고급 언어로 개발할 수 있게 하는 것과 (2) SDR 응용이 재구성될 수 있도록 컴포넌트 기반 컴퓨팅을 지원할 수 있어야 하는 것이다. 전자의 기능은 시스템 소프트웨어의 최하부 계층으로서 RTOS(Real-Time Operating Systems, 실시간 운영체제)가 제공하고, 후자의 기능은 응용 소프트웨어와 RTOS 사이에 위치하는 중간 계층인 내장형 미들웨어가 제공한다. 이들 시스템 소프트웨어들은 실시간 시간 제약 조건이 만족될 수 있도록 SDR 무선 응용을 처리하여야 하며, SDR 내장형 시스템에 사용될 수 있도록 경량성을 갖추어야 한다.

본 고에서는 SDR을 위한 RTOS와 내장형 미들

웨어의 설계에 대하여 설명하고자 한다. 먼저 SDR 포럼[1]에서 제시한 소프트웨어 프레임워크를 통해 SDR 무선 응용의 특성에 대하여 살펴본다. 이를 토대로 SDR을 지원하기 위한 시스템 소프트웨어로서 RTOS와 내장형 미들웨어의 요건을 이끌어낸다. 이어서 이들의 개념과 역할, 요건에 대하여 설명하고, 다양한 SDR 시스템의 구성에 따른 RTOS의 설계, 소프트웨어의 재구성성을 위한 미들웨어의 설계, 그리고 이와 관련된 표준들에 대하여 설명한다.

2. SDR 응용의 특성

이 절에서는 SDR을 지원하기 위해서 시스템 소프트웨어가 갖추어야 할 요건을 살펴보기 위하여 SDR 무선 응용의 특성을 살펴본다. 구체적으로 SDR 포럼에서 제시한 SDR 참조 모델과 소프트웨어 구조의 표준을 살펴본다.

2.1. SDR 소프트웨어 참조 모델

SDR 포럼에서는 SDR 소프트웨어를 기능적인 관점에서 기술하기 위하여 그림 1과 같은 'SDR 소프트웨어 참조 모델'[2]을 제시하였다. SDR 소프트웨어 참조 모델에서는 추상화 정도에 따라 그림에서와 같이 세 가지 관점으로 SDR 소프트웨어의 기능이 기술한다.

첫 번째 관점에서 SDR 소프트웨어는 정보 전송 쓰레드로 기술된다. 정보 전송은 송신과 수신의 양방향으로 이루어질 수 있다. 그림에서 수신 방향은 왼쪽에서 오른쪽이다. 정보 수신 쓰레드의 입력 인터페이스는 전파(air) 인터페이스이며, 출력 인터페이스는 유선과 사용자 인터페이스이다. 여기에서 처리하는 정보는 데이터일 수도 있고 제어 정보일 수도 있다.

두 번째 관점에서는 보다 구체적으로 SDR의 핵심 기능을 네 가지로서 기술한다. 그림에서와 같이 이는 (1) 전단 (front end) 처리, (2) 보안 처리 (information security), (3) 정보 처리, (4) 제어로 이루어진다.

마지막으로 세 번째 관점에서는 두 번째 관점의 기능들을 그림에서와 같이 세부 기능으로 나누어 기술한다. 예를 들면, 정보 처리는 경로 선택, 멀티플랙싱, 소스 코딩, 시그날링 프로토콜, I/O 기능 등으로 매핑된다.

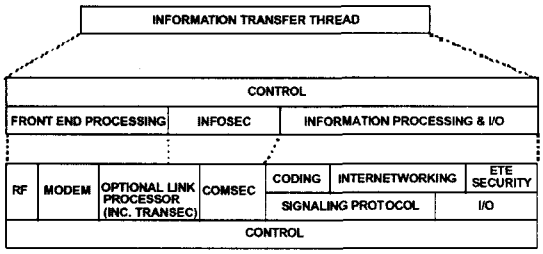


그림 1. SDR 소프트웨어 참조 모델 (상위 수준 기능 모델)

SDR 소프트웨어 참조 모델에 기술된 이러한 기능들은 제한된 시간 안에 결과를 내는 것이 결과의 정확성 못지 않게 중요하다. 따라서 SDR 응용을 수행하는 시스템 소프트웨어가 실시간 처리 능력을 제공하는 것은 필수적이 된다. 특히 RTOS 뿐만 아니라, 내장형 미들웨어의 실시간 처리 능력도 간과해서는 안 된다. 이러한 RTOS와 미들웨어의 요건에 대하여서는 3.2절과 4.2절에서 자세히 기술한다.

2.2. SDR 소프트웨어 구조

앞에서 기술한 SDR 소프트웨어 참조 모델은 SDR 소프트웨어의 기능만을 정의하였을 뿐, 실제로 SDR 시스템을 구현할 때 고려해야 할 소프트웨어의 구조의 문제에 대하여서는 다루지 않고 있다. SDR 소프트웨어 참조 모델에서 기술하는 다양한 기능들을 구현하기 위해서 SDR 시스템은 여러 하드웨어 노드들로 구성된 분산 시스템으로 설계되어야 한다. 이러

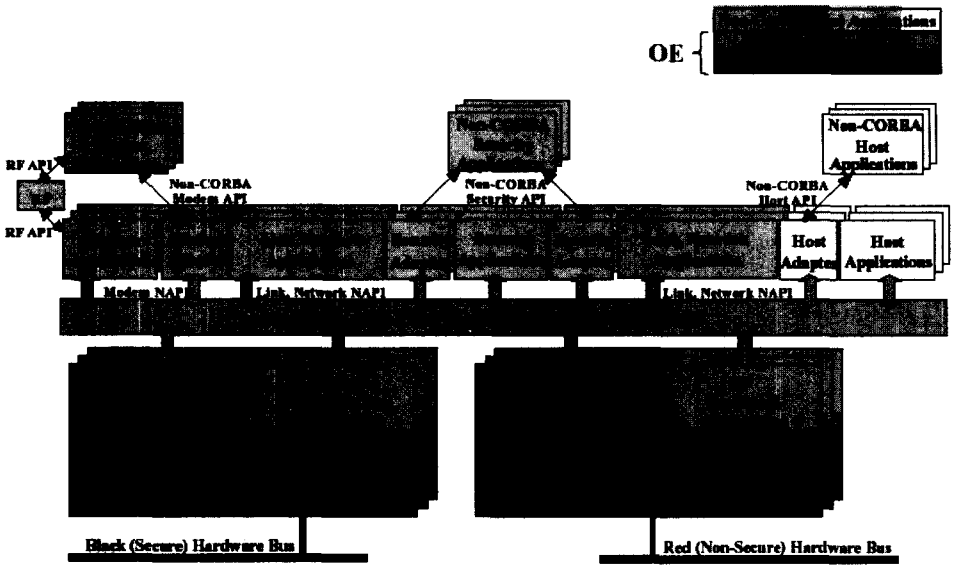


그림 2. SCA(Software Communication Architecture)에서 제시한 SDR 소프트웨어 구조

한 분산 시스템 상에서 SDR 소프트웨어 시스템을 구현하려면 재구성성과 유연성이 필요하며 따라서 이를 충족시켜줄 수 있는 객체 지향 컴퓨팅 기술이 필요하게 된다. 즉, SDR 소프트웨어의 구조는 기본적으로 분산 객체 지향 소프트웨어 구조가 되는 것이다.

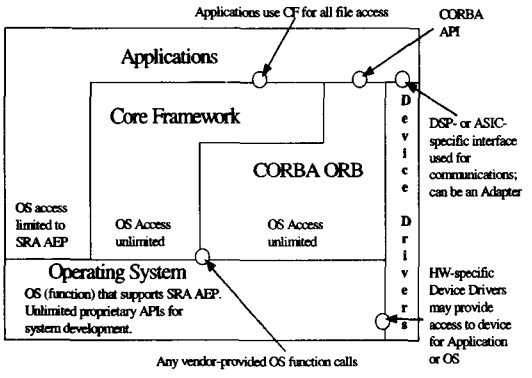


그림 3. SCA Operating Environment 개관

SDR 포럼에서는 JTRS(Joint Tactical Radio System)[3]의 SCA(Software Communication Architecture)[4]를 SDR 소프트웨어 구조의 표준으로 인정하였다. SCA에서 제시한 SDR 소프트웨어 구조는 그림 2와 같다. SCA는 크게 응용 프로그램과 운영 환경(Operating Environment: OE)의 계층구조로 나누어지며, 운영 환경 OE는 그림 3과 같이 RTOS, CORBA(Common Object Request Broker Architecture)[5], 코어 프레임워크(Core Framework: CF) 인터페이스의 계층구조로 다시 나누어진다. 여기에서 CORBA와 RTOS는 COTS(Commercial Off-The-Shelf: 상용 출시 제품)를 사용할 것을 권장하고 있다. CORBA와 코어 프레임워크 계층을 통틀어 미들웨어로 본다면, SCA에서도 앞에서 설명한 계층 구조를 그대로 따르게 된다. 한편 그림 2에서와 같이 SDR 참조 모델의 기능들이 분산 객체로서 구현되어 분산 객체 지향 소프트웨어 구조를 이루는 것을 알 수 있다.

이러한 분산 객체 구조를 가능하게 하는 핵심은 CORBA 미들웨어이다. CORBA는 '소프트웨어 버스'로서 분산성과 이종성을 숨기고 분산 객체들이 유

연하게 통신할 수 있게 해주는 핵심 역할을 담당한다. CORBA에 대하여서는 4.3절에서 보다 자세히 설명한다. 또한 SCA에서 CORBA와 함께 미들웨어 계층을 구성하는 코어 프레임워크에 대하여서는 4.4절에서 설명한다.

3. SDR을 위한 RTOS의 설계

SDR에서 가장 핵심적인 시스템 소프트웨어는 단연 RTOS라고 할 수 있다. 본 절에서는 RTOS의 개념과 역할, 내장형 RTOS의 요건, SDR 시스템을 위한 RTOS의 구성, SCA가 지정한 RTOS 표준인 POSIX.13 표준 등에 대해서 살펴보도록 한다.

3.1. RTOS의 개념과 역할

RTOS는 실시간 시스템을 위한 OS(Operating Systems, 운영체제)이다. 실시간 시스템은 '시간에 맞게' 올바른 결과를 전달해야 하는 시스템으로 정의된다. 전통적으로 RTOS는 자원 제약을 가지는 내장형 실시간 시스템을 위한 OS로 개발되어 사용되어 왔다. 이에 따라 RTOS는 내장형 시스템을 위한 OS라는 의미로서도 널리 사용되고 있다.

RTOS가 SDR 시스템 개발자에게 제공하는 주된 기능은 (1) SDR 하드웨어 특성의 추상화, (2) SDR 하드웨어 자원의 관리, (3) 멀티태스킹이라는 세 가지로 요약할 수 있다. 이를 구체적으로 설명하면 다음과 같다. 첫째로, RTOS는 복잡한 SDR 무선 응용을 개발할 때 효율적으로 소프트웨어를 개발할 수 있도록 하드웨어를 추상화하는 역할을 한다. 둘째로, RTOS는 이에 더하여 SDR 하드웨어 자원의 제약성을 고려하여 이들을 효율적으로 관리하는 역할을 한다. 셋째로, RTOS는 여러 SDR 응용이 동시에 수행되도록 함으로써 프로세서의 활용률을 높이기 위한 멀티태스킹 기능을 지원하여야 한다.

이러한 기능들은 범용 시스템을 위한 OS에서도 일반적으로 제공되는 기능들이기도 하다. RTOS는 이에 더하여 "실시간성"이라는 QoS 지원 기능을 제공하여야 한다. 2절에서 설명하였듯이 SDR 응용은 실시간 처리를 요구하기 때문에 실시간성의 지원은 SDR의 시스템 소프트웨어로서 RTOS가 제공해야 하는 필수적인 조건이다. 구체적으로 RTOS는 멀티태스킹을 지원함에 있어서 태스크간의 문맥 전환에 걸리는 시간을 일정 시간 이하로 제한시켜서 예측 가능하도록 해야 한다. 이러한 RTOS의 요건에 대하여서는 다음절에서 자세히 설명한다.

3.1.1. 하드웨어 특성의 추상화

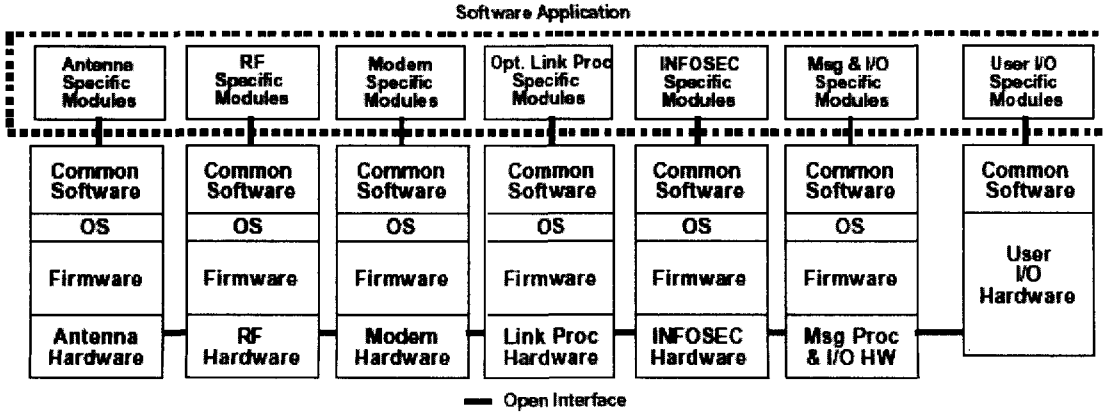


그림 4. SDR 시스템 구성의 예

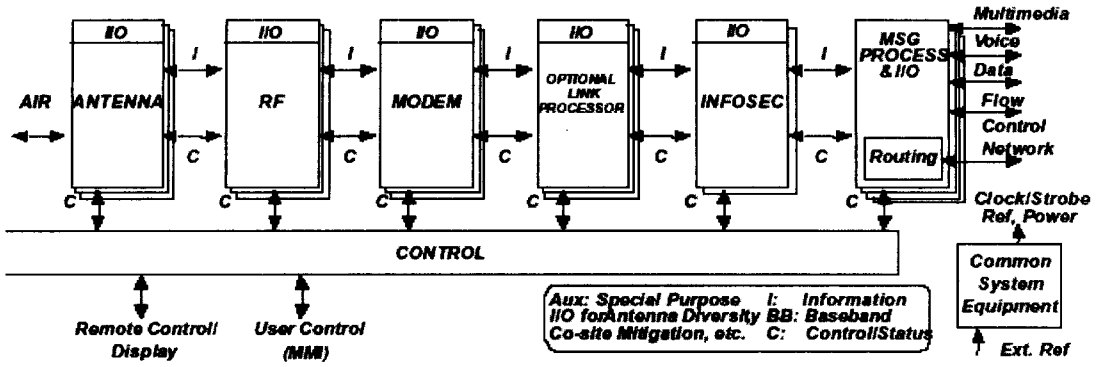


그림 5. 다중 모드를 지원하는 SDR 시스템의 예

그림 4는 SDR 포럼에서 제시한 SDR 참조 모델과 SDR 소프트웨어 구조에 따른 SDR 시스템 구성의 한 예를 보여준다. 그림에서와 같이 SDR 시스템을 구성하는 하드웨어는 안테나, RF, 모뎀, 암복호화 등의 기능에 따라 다양한 프로세서와 각기 다른 메모리, 저장장치로 이루어진다. 그 외 통신을 위한 다양한 주변 장치들을 포함할 수도 있다. RTOS를 사용하지 않고 응용 프로그램들이 이들을 조작하기 위한 모든 기능을 일일이 구현해야 한다면, 이는 불가능하지는 않다 하더라도 소프트웨어의 유연성, 탄력성, 유지 보수, 그리고 안정성의 모든 측면에서 매우 불합리한 일이다. RTOS는 각 디바이스를 조작할 수 있는 디바이스 드라이버를 제공하고, 이들 디바이스를 조작할 수 있는 공통 API를 제공함으로써 하드웨어의 특성을 추상화시켜 준다.

일례로, "Hello, World." 라는 간단한 문장을 디스플레이 화면에 보여주는 코딩에 대하여 생각해보자. RTOS를 사용하면 고급언어를 사용하여 몇 줄로 간단히 코딩할 수 있지만, RTOS 없이 프로그래밍할 경우에는 비디오 카드 초기화, 메모리 관리, 비디오 카드 동작의 모든 작업들에 대하여 프로그래머가 코딩을 해주어야 한다. SDR 시스템은 다양한 하드웨어를 기반으로 하여 복잡한 고기능의 작업을 수행해야 하기 때문에 RTOS의 도움이 없이 SDR 시스템의 소프트웨어를 개발하고 유지 관리하는 것은 불가능에 가깝다.

3.1.2. 하드웨어 자원의 관리

RTOS는 단순히 하드웨어가 제공하는 기능을 추상화하는 것을 넘어, 하드웨어 자원을 좀 더 효율적으로 관리하는 기능을 제공하여야 한다. 이는 하드웨어 자원의 제약이 강한 SDR 시스템에서 더욱 절실히 요구되는 기능이다. 예를 들면, 메모리를 보다 효율적으로 사용하기 위하여 가상 메모리 기능을 제공하는 것, 저장 장치를 효율적으로 사용하도록 파일 시스템을 제공하는 것, 배터리 용량의 제약 문제를 극복하기 위하여 시스템을 대기 모드로 들어가게 하는 등의 저전력 정책을 취하는 것 등을 들 수 있다.

한 시스템이 다수의 응용 프로그램으로 구성되는 경우 RTOS의 도움 없이 각 응용 프로그램에서 이런 기능들을 직접 구현하는 것은 근본적으로 불가능한 경우가 많다. 또한 가능한 경우에도 시스템의 안정성 측면에서도 매우 불합리하다.

3.1.3. 멀티태스킹

태스크는 응용 프로그램의 실행 가능한 기본 단위이며 통상 응용 프로그램의 독립적인 수행 컴포넌트를 나타낸다. 태스크는 수행을 중단했다가 다시 시작

할 수 있도록, 메모리에 런타임 문맥을 가진다. SDR 내장형 시스템의 소프트웨어는 종종 상호 작용하는 태스크들의 그룹으로 구조화된다. 멀티태스킹이란 하나 이상의 태스크들을 CPU 상에서 스케줄링하고 전환하는 과정을 말한다. 이런 멀티태스킹을 통하여 응용 프로그램의 동시성을 증가시킬 수 있으며, 이는 CPU의 이용률을 증가시키는 데에도 도움이 된다.

그림 5는 다중 모드를 지원하는 SDR 시스템의 예를 보여준다. 여기에서 SDR 참조 모델의 기능을 구현하는 각 모듈을 하나의 태스크로 본다면, SDR 시스템은 여러 개의 태스크로 구성된 멀티태스킹 시스템의 전형적인 예로 볼 수 있다.

하나의 응용 프로그램 내부에 멀티태스킹 기능을 구현하여 넣는 것은 RTOS의 도움을 받지 않고서도 사실 가능하다. 예를 들면, 가장 간단하게는 무한 루프를 사용하여 각 태스크에 해당하는 코드가 순차적으로 수행되도록 하는 방법이 있다. 이런 방식을 사용하게 되면 각 태스크의 최악 응답 시간이 다른 태스크들의 수행시간의 합이 되어 응답시간이 과도하게 길어지게 된다. 이를 해결하기 위하여서 각 태스크의 코드를 쪼개어 루프의 매 회마다 태스크의 일부분만이 순차적으로 수행되게 할 수도 있다. 그러나 이러한 방식은 코드를 알아보게 힘들게 하고 쪼개진 태스크의 일부분이 전체적으로 제대로 동작하는지를 검증하는 작업도 부가적으로 필요하게 된다. 결과적으로 복잡한 응용을 구현하기에는 유지 보수 측면에서 거의 불가능하게 된다.

이와 다른 방법으로서 하드웨어 인터럽트 핸들러로서 각 태스크를 구현하는 방법이 있다. 그러나 이것도 인터럽트의 처리가 과도해 질 경우 원래의 무한 루프를 써서 멀티태스킹을 구현할 때와 같은 문제점이 있다. 또한 보통 시스템에서 사용 가능한 하드웨어 인터럽트의 수가 일반적으로 심히 제한되어 있다는 것도 문제점이다.

따라서 RTOS에서 직접적으로 지원하는 멀티태스킹 기능을 이용하는 것이 실시간 처리를 요구하는 복잡한 SDR 응용 프로그램을 구현하기 위하여서는 필수적이다. 멀티태스킹은 수행 중인 태스크의 문맥을 저장하고 복구할 수 있는 기능이 지원되어야 가능하다. 태스크의 수행 중 문맥은 태스크 코드, 데이터 세그먼트, 스택, CPU 레지스터 등으로 구성된다. 태스크의 문맥 전환은 멀티태스킹 커널이 다른 태스크를 수행하겠다고 결정할 때 일어난다. 이는 커널이 현재 수행 중인 태스크의 문맥을 그 태스크의 스택에 저장하고, 새로 수행시켜야 할 태스크의 문맥을 그 태스크의 스택으로부터 복구함으로써 간단하게 이루어진다.

멀티태스킹에 있어서 태스크는 중요도에 따라 우

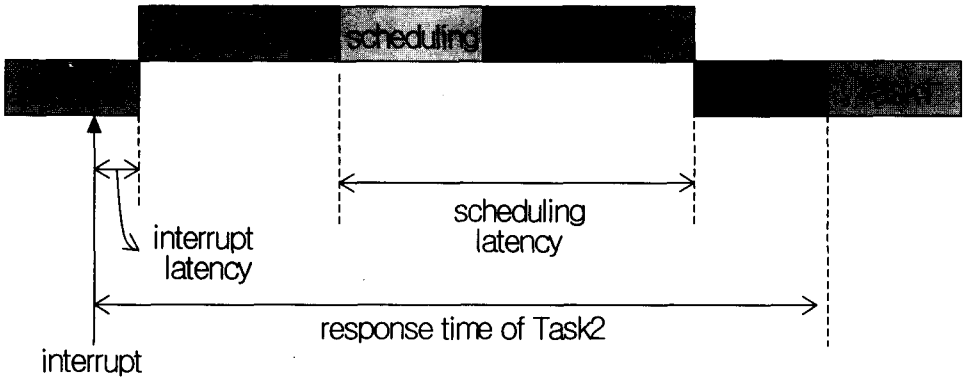


그림 6. 문맥 전환 시간의 분해

선 순위를 가질 수 있다. 가장 높은 우선 순위를 가진 태스크가 항상 대기중인 태스크들 중에서 선택되어 CPU의 제어권을 받아야 한다. 멀티태스킹 스케줄링 방식은 또한 선점형 스케줄링과 비선점형 스케줄링의 두 가지 종류로 분류할 수 있다. 완전 선점형 스케줄링에서는 만약 현재 수행중의 태스크의 우선 순위보다 더 높은 우선 순위의 태스크가 시스템에 도착하면, 현재 수행중인 태스크는 선점되고, 높은 우선 순위의 태스크가 즉각적으로 CPU의 제어권을 받게 된다. 비선점형 스케줄링에서는 현재 수행중인 태스크가 수행을 종료할 때까지 문맥 전환이 이루어지지 않는다. 선점형 스케줄링은 시스템 응답 시간이 중요할 때 사용되며, 대부분의 상용 실시간 운영체제에서 지원된다.

3.2. 내장형 RTOS의 요건

3.1절에서 설명하였듯이 RTOS는 SDR 실시간 응용 태스크들이 시간에 맞게 결과를 낼 수 있도록 멀티태스킹을 지원하여야 한다. 이를 위해서 RTOS는 태스크간의 문맥 전환에 걸리는 시간을 한정할 수 있어야 한다. 이는 결국 (1) 우선 순위 기반 선점 스케줄링을 제공하고, (2) 인터럽트 지연 시간과 스케줄링 지연 시간이 일정 범위 안에 있어야 함을 의미한다.

처음의 조건은 응답 시간이 중요한 태스크(실시간 태스크)가 높은 우선 순위를 가지고 불리었을 때, 제한된 지연 시간 안에 처리되어야 함을 의미한다. 만약에 실시간 운영체제가 비선점형 커널을 가지고 있다면, 새롭게 도착한 높은 우선 순위의 태스크의 수행이 현재 수행 중인 낮은 우선 순위 태스크가 CPU를 내놓을 때까지 지연될 수 있다.

두 번째 조건은 첫 번째 조건이 만족되었을 때 문맥 전환 시간을 한정하기 위하여 필수적인 조건이다. 그림 6은 문맥 전환 시간을 분해한 모습을 보여준다. 여기에서는 태스크 1이 수행중인 동안 시스템에 인터럽트가 발생하여 우선 순위가 더 높은 태스크 2가 호출되는 상황을 보여주고 있다. 그림에서와 같이, 태스크 2가 수행되기 전에 스케줄러의 루틴이 들어 있는 인터럽트 서비스 루틴이 수행된다. 여기에서 인터럽트 발생으로부터 인터럽트 서비스 루틴의 첫 번째 명령이 수행될 때까지의 시간 간격을 "인터럽트 지연 시간"이라고 한다. 또한 스케줄러 루틴의 실행 시간을 "스케줄링 지연 시간"이라고 한다. 태스크 2의 응

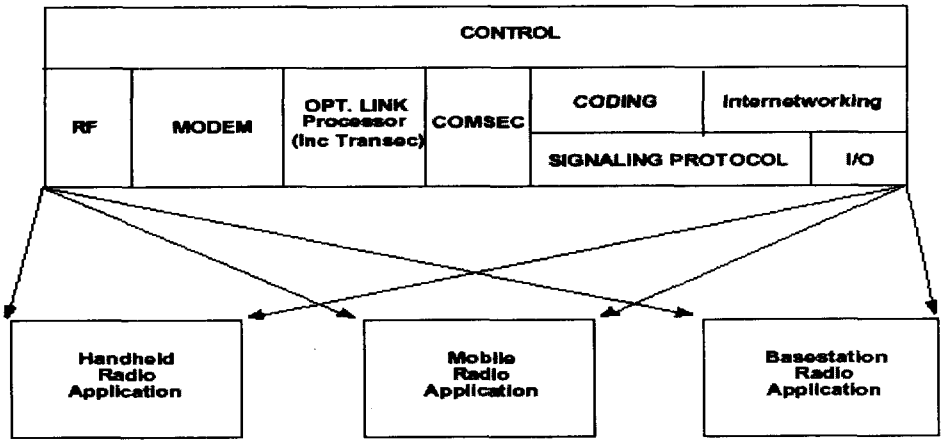


그림 7. SDR 참조 모델이 적용되는 응용 시스템의 예

답 시간의 범위를 제한하기 위하여 RTOS는 태스크를 선점하는 데 걸리는 문맥 전환 지연 시간, 즉 인터럽트 지연 시간과 스케줄링 지연 시간의 범위를 제한해야 한다. 여기에서 인터럽트 서비스 루틴들은 응용 태스크 코드처럼 진정한 운영체제 코드가 아니기 때문에 이들의 수행 시간은 고려하지 않았다. 인터럽트 지연 시간은 운영체제와 응용 프로그램들에서 인터럽트 불능 코드 영역에 의해 결정된다. 스케줄링 제한 시간은 RTOS의 우선 순위 중재 작용에 의해 결정된다.

이러한 조건 외에도, 내장형 RTOS는 적용될 SDR 시스템에서 요구하는 수준의 경량성을 만족시켜야 하며, 편리한 개발 도구도 뒷받침하여야 한다. 사실 RTOS들이 기능적 측면에서 큰 차이를 보이지 않기 때문에 이러한 부수적인 측면이 RTOS를 선택함에 있어서 종종 실질적으로 더 중요한 기준이 되기도 한다.

3.3. SDR 시스템의 구성에 따른 RTOS의 구성

그림 7은 SDR 포럼에서 제시한 SDR 참조 모델이 적용되는 SDR 시스템의 예를 보여준다. 이와 같이 현재 SDR의 대상 모델은 (1) 휴대 단말기(handheld), (2) 이동 정보 시스템, (3) 기지국으로 분류된다. 이동 정보 시스템은 자동차, 배, 비행기 등의 이동이 가능하면서 휴대 단말기보다는 상대적으로 자원 제약이 덜한 시스템을 의미한다. 여기에서 주목할 점은 휴대 단말기는 한 순간에 하나의 무선 표준만을 하나의 세션으로서 제공하지만, 이동 정보 시스템과 기지국은 여러 무선 표준을 여러 세션을 통해 제공할 수 있어야 한다고 SDR 포럼이 규정하고 있다는 점이다. SDR 시스템마다 이렇게 자원 제약 정도, 이동/정지의 동작 환경, 기능상의 차이가 존재하기 때문에 요구되는 기반 RTOS에 있어서도 차이가 발생하게 된다.

예를 들면, 메모리 관리 기능에 있어서 MMU (Memory Management Unit)가 없는 시스템과 있는 시스템에 대하여 RTOS의 메모리 관리자의 구현이 달라져야 한다. 파일 시스템에 있어서도 시스템에 따라 아예 제공할 필요가 없는 경우도 있고, 디렉토리 구조가 없는 단순한 파일 시스템을 제공하는 경우도 있을 수도 있다. 멀티태스킹의 구현에 있어서도 자원의 제약 정도에 따라 단일 프로세스에서 멀티쓰레딩으로만 구현될 수도 있고, 좀더 일반화된 형태로 멀티프로세싱 기능이 멀티쓰레딩과 함께 제공될 수도 있다.

3.3.1. 컴포넌트 기반 RTOS

이와 같이 SDR 시스템의 종류에 따라 상이하게 요구되는 RTOS를 특정 응용과 사용되는 디바이스의 종류에 맞추어 구성할 필요가 있다. 이에 실시간 내장형 시스템에 탑재될 특정 응용에 적합한 운영체제(application specific operating system)를 기존에 만들어 둔 기능 블록(컴포넌트)들로부터 가능한 한 빠르게 조율하고 재구성하는 컴포넌트 기반 RTOS가 주요한 기술로 대두된다.

대표적인 컴포넌트 기반 RTOS의 예로 eCos(embedded Configurable operating system)를 들 수 있다. eCos는 운영체제의 기능 각 부분을 모듈화, 컴포넌트화하여 원하는 시스템을 위한 운영체제를 빠르게 구성하는 것을 목적으로 한다. 이러한 운영체제들은 필요한 기능을 수행하는 미리 작성된 코드 컴포넌트들을 단순히 수집함으로써 구성되기 때문에 time-to-market을 줄일 수 있을 뿐만 아니라 필요 없는 기능을 쉽게 제거할 수 있다. 따라서 범용 운영체제에 비하여 훨씬 뛰어난 수행 능력을 보여준다.

컴포넌트 기반 RTOS를 설계하는데 있어서 가장 큰 이슈는 각 모듈의 크기를 결정하는 일이다. 컴포넌트가 필요한 기능들을 너무 많이 구현하게 되면 크기가 너무 커지게 되고, 반면 컴포넌트의 재사용성을

높이기 위해서 크기를 작게 해서 구현하게 되면 컴포넌트간의 의존도가 증가하여 관리가 어려워지게 될 것이다. 또 컴포넌트들을 모으고 재구성하기 위해서는 각 컴포넌트의 기능과 의존성, 그리고 인터페이스 등이 잘 기술될 수 있어야만 한다. 왜냐하면 이렇게 기술된 정보를 바탕으로 각 컴포넌트가 서로 연결될 수 있고 상호 적합성이 사전에 검증될 수 있어야 하기 때문이다.

최근에는 RTOS가 컴포넌트 기반화 되는 것 이외에 한발 더 나아가 수행 시에 컴포넌트의 추가, 교체, 제거를 지원하는 기법에 관한 연구가 진행 중에 있다. 이러한 운영체제의 동적 재구성 기능은 수행 시에 일부 컴포넌트만 독립적으로 업그레이드하거나 추가할 수 있고 네트워크 대역폭이나 CPU 사용량에 따라 운영체제 자체가 적응할 수 있는 기능 등을 의미한다. 특히 SDR 시스템에서 FPGA를 이용하여 하드웨어를 소프트웨어적으로 수행 시에 재구성하는 것이 가능해지고 사용자에게 QoS를 제공하는 것이 강조되면서 이는 더욱 중요한 기능이 되고 있다.

3.4. POSIX.13 실시간 시스템 프로파일

SCA에서는 RTOS가 POSIX.13 실시간 시스템 프로파일[7] 중의 하나를 준수하여 개발될 것을 규정하고 있다. POSIX.13은 RTOS 기능들의 조합에 대한 표준을 제공하여 SDR 시스템이 적절한 RTOS 사양을 선택하게 하는데 도움을 준다.

POSIX[6]는 'Portable Operating System Interface (into a UNIX like kernel)'의 의미이다. 이는 IEEE에서 제정하여 산업계 표준으로 받아들여지고 있는 UNIX 계열의 운영체제 API의 표준이다. 실시간 내장형 시스템의 운영체제는 POSIX의 API들을 전부 지원할 필요가 없기 때문에 POSIX의 부분집합만을 지원하게 된다. 따라서 이러한 가능한 POSIX API의 부분집합에 대하여 표준을 마련할 필요가 있게 된다. 이러한 필요성에 의하여 생겨난 표준이 POSIX.13 실시간 시스템 프로파일이다. POSIX.13은 실시간 시스템이 시스템의 특성에 따라 지원할 필요가 있는 POSIX API의 일부분들만 그룹지어 정의한 표준이다.

3.3절에서 설명하였듯이 실시간 시스템에 따라 멀티태스킹, 메모리 관리, 파일 시스템 등의 기능에 있어서 요구되는 정도가 다른데, POSIX.13은 이에 대하여 가능한 조합을 미리 정의해 놓은 것이라고 할 수 있다. POSIX.13 실시간 시스템 프로파일은 총 4개의 프로파일로 구성되며, 이는 구체적으로 다음과 같다.

- Minimal Realtime System Profile (PSE51)
  대상 시스템은 조작자의 개입 없이 하나 또는 여러 개의 특별한 IO 장치를 제어하는 시스템이다. 대상 하드웨어는 메모리가 있는 단일 프로세서로, MMU가 없는 시스템이다. 멀티태스킹 기능은 단일 프로세스에 멀티쓰레딩으로서 지원되며, 파일 시스템도 존재하지 않는다.
- Realtime Controller System Profile (PSE52)
  Minimal Realtime System Profile의 확장으로 대상 하드웨어도 PSE51과 같다. PSE51처럼 여전히 단일 프로세스 멀티쓰레딩 기능만 지원하지만, 파일 시스템 인터페이스와 비동기 I/O 인터페이스를 지원한다.
- Dedicated Realtime System Profile (PSE53)
  이것도 Minimal Realtime System Profile의 확장이지만, 멀티 프로세스를 지원한다. 디바이스 드라이버와 파일을 위한 공통된 인터페이스가 있지만, 파일시스템에 디렉토리 체계가 없다. 메모리 잠금이 가능하고, MMU가 있는 시스템도 지원 가능하고, 여러 프로세서를 지원하는 것도 가능하다.
- Multi-Purpose Realtime System Profile

(PSE54)
실시간 및 비실시간 작업의 혼용을 지원한다. POSIX의 모든 API를 지원하며, 조작자와의 상호 작용을 위하여 쉘 API까지도 지원한다. 대상 하드웨어는 고속 저장 장치와 디스플레이 장치, 네트워크를 지원하며 MMU가 있는 다중 프로세서 시스템이다.

4. SDR을 위한 내장형 미들웨어의 설계

미들웨어는 SDR을 구성하는 분산 노드들의 유연한 통신을 위해 필수적인 시스템 소프트웨어이다. 본 절에서는 내장형 미들웨어의 개념과 역할, 내장형 미들웨어의 요건, SCA가 지정한 미들웨어 표준인 CORBA와 SCA 코어 프레임워크 표준 등에 대해서 살펴보도록 한다.

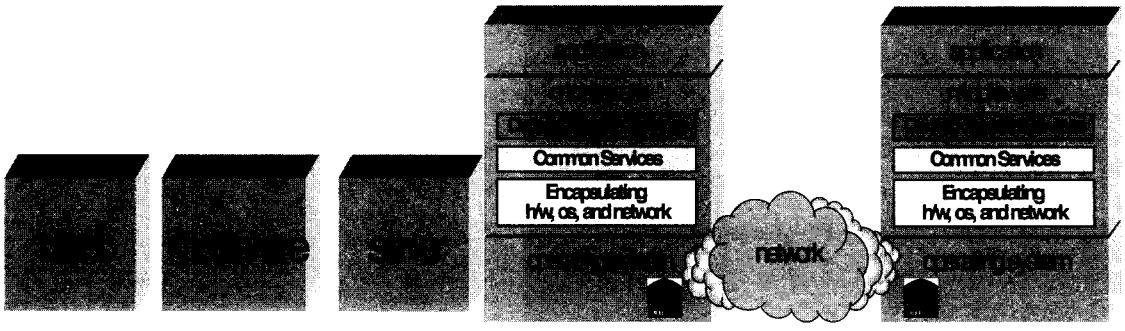
4.1. 미들웨어의 개념과 역할

미들웨어는 분산 노드들의 다양한 이종성을 숨기고 분산 응용의 부분들이 상호 작용할 수 있게 하는 역할을 담당하는 시스템 소프트웨어이다. 한마디로 요약하면, 미들웨어는 컴포넌트 기반 분산 시스템 컴퓨팅을 위한 소프트웨어이다. SDR 시스템은 통상적으로 이기종의 분산 컴포넌트들이 모여 상호 작용해야 하기 때문에 이러한 역할을 하는 미들웨어가 필수적이다.

그림 8은 미들웨어의 기본 개념을 보여준다. 그림 8 (a)에서처럼 미들웨어는 단순하게는 클라이언트와 서버로 상호 작용하는 부분들의 가운데에 존재하여 둘 사이의 통신을 매개해주는 역할을 담당하는 소프트웨어를 의미한다. 그러나 일반적으로는 이러한 수평 방향으로의 중간 매개 소프트웨어의 의미보다는 그림 8 (b)와 같이 수직 방향으로 운영체제와 응용 프로그램 사이에 위치하는 중간 매개 소프트웨어의 의미로 받아들여진다.

미들웨어가 SDR 시스템 개발자에게 제공하는 주된 기능은 (1) 다양한 이기종의 하드웨어, 운영체제, 네트워크의 존재를 추상화 시켜주고, (2) 특정 컴포넌트를 찾기 위한 디렉토리와 명명 서비스 등의 서비스를 제공하고, (3) 필요에 따라 특정 도메인에서 필요한 라이브러리 서비스를 제공하는 것으로 세분화된다. 이와 더불어 응용 프로그램의 이식성을 돕기 위한 수단으로서 다양한 언어로 작성된 응용 프로그램이 분산 통신을 할 수 있도록 도와주는 역할을 수행할 수도 있으며, 컴포넌트의 상호 계약을 나타내기 위한 계약 기술 언어의 명세를 제공할 수도 있다.

4.2. 내장형 미들웨어의 요건



(a) (b)

그림 8. 미들웨어의 기본 개념

SDR 시스템을 위한 내장형 미들웨어는 RTOS와 더불어 시스템에서 요구하는 경량성을 만족시키고, 프로세싱 시간이 한정되어야 하는 실시간성을 만족시켜야 한다. 미들웨어가 제공하는 클라이언트와 서버 간의 통신 방식은 크게 (1) 단방향 (one-way), (2) 비동기, (3) 동기 통신의 세 가지 방식으로 구분할 수 있다. 단방향 통신에서는 클라이언트가 서버에게 메시지를 보내기만 하고, 서버가 메시지를 받았는지, 제대로 전달되었는지, 오류가 발생하였는지에 대하여 전혀 신경을 쓰거나 별도의 처리가 없는 방식이다. 비동기 통신은 서버에게 메시지를 보내고 블록킹 없이 수행을 계속 진행하지만, 제대로 전달되지 않는 등의 오류가 발생하였을 경우 이를 처리하도록 하는 방식이다. 마지막으로 동기 통신은 클라이언트가 서버에게 메시지를 보내고 응답이 올 때까지 기다리는 방식이다.

동기 통신은 네트워크 부하를 포함하기 때문에 실시간성을 보장하기가 어려워지는 측면이 있으며, 나머지 두 방식은 클라이언트와 서버간의 데이터와 상태의 동기화가 힘든 면이 있어 각각의 장단점이 있다. 네트워크의 부하는 네트워크의 종류 및 주변 환경에 따라 예측하기가 힘들고, 또한 내장형 시스템에서 응용 프로그램간의 컴포넌트 분산 통신은 독점적인 네트워크를 통하여 이루어지기 때문에, 통상 내장형 미들웨어의 성능 지수는 네트워크의 부하가 없는 상태에서 측정하여 제시된다.

내장형 미들웨어의 성능 지수는 클라이언트와 서버의 배치에 따라 (1) 병치(collocated), (2) 국지(local), (3) 프로세서 대 프로세서의 세 가지 배치에 대하여 정해진 데이터의 양(통상 32bit 프로세서에서 64byte 스트링)을 위에서 설명한 각 세 가지 통신 방식에 대하여 응답 시간을 측정하여 제시한다. 병치 배치는 클라이언트와 서버가 같은 프로세스에 속한 경우를, 국지 배치는 클라이언트와 서버가 같은 프로세서에서 수행되나 다른 프로세스에 속한 경우를, 프로세서 대 프로세서는 클라이언트와 서버가 각

```
module AM_FM_Virtual_Link { // Namespace for FM/AM VHF/UHF
    interface Link_Command {
        attribute float frequency;
    };
    enum ModulationType { AM, FM };
    exception OutOfRange { string errormsg; };
    exception IllegalFrequency { string errormsg; };
    interface Xmit : Link_Command { // Transmit inherits from Link_Command
        void set_xmit_channel (in float frequency,
                               in ModulationType Modulation)
                          raises (OutOfRange, IllegalFrequency);
        void transmit (); // PTT is asserted
    };
    interface Rcv : Link_Command { // Receive inherits from Link_Command
        void set_rcv_channel (in float frequency,
                              in ModulationType Modulation)
                          raises (OutOfRange);
        void receive (); // Initial condition, PTT has been released
    };
};
```

그림 9. CORBA IDL을 이용한 SDR 시스템 조작자 인터페이스 기술의 예

각 다른 프로세서의 프로세스에 속한 경우를 말한다.

SDR 포럼의 문서에 따르면, 실시간 프로토콜을 처리함에 있어서 상용 CORBA를 사용할 경우, CORBA에서의 처리 시간이 전체 프로토콜 처리 시간의 20%를 넘지 않았다는 결과가 있다. 이는 불필요한 부하가 아니라 분산 통신을 처리하기 위하여 드는 시간임을 주목할 필요가 있다.

4.3. CORBA와 CORBA 서비스

이 절에서는 SCA에서 채택한 미들웨어의 표준인 CORBA에 대하여 좀 더 자세히 살펴본다. SCA에서는 경량성을 위하여 SDR 시스템의 미들웨어로서 CORBA의 내장형 시스템을 위한 표준인 Minimum CORBA[8]를 규정하고 있다. Minimum CORBA에서는 전체 CORBA 명세에서 주로 동적 인터페이스 호출 기능을 지원하지 않음으로써 경량성을 꾀한다. 한편 SCA에서는 시스템을 이루는 소프트웨어 컴포넌트들을 검색하기 위하여 CORBA의 명명(naming) 서비스를 사용할 것을 명세하고 있다.

CORBA는 4.1절에서 설명한 미들웨어의 역할을 다음과 같이 실현하여 지원한다. 우선 컴포넌트의 상호 계약을 나타내기 위한 계약 기술 언어의 명세로서 IDL(Interface Definition Language)을 제공한다. 그림 9는 IDL을 사용하여 SDR 시스템의 조작자에 대한 인터페이스를 기술한 예를 보여준다. 또한 CORBA는 구현 언어의 이종성을 숨기기 위하여 IDL stub과 skeleton, 그리고 이를 생성해주는 IDL 컴파일러를 제공하며, 다양한 네트워크의 존재로 인한 분산성과 이종성을 숨기고 하드웨어간에 달라질 수 있는 데이터 표현 방식(endian)의 이종성을 숨기기 위하여 IOP(Inter-ORB Protocol)을 제공한다. 그리고 디렉토리와 명명 서비스를 위하여 인터페이스 저장소(IR: Interface Repository)를 제공하고, CORBA 명명 서비스를 제공한다.

그림 10은 CORBA의 기본 구조를 보여준다. 최상위에서 추상적으로 보면, 클라이언트 객체는 서버 객체에 인자(args)와 함께 operation을 호출하고, 서버 객체는 이에 리턴 값과 가능한 경우 (out arg에 대하여) 인자의 값을 변경시킴으로써 응답한다. CORBA는 이러한 메시지 전달 과정이 유연하게 이루어지도록, 분산성과 이종성을 숨기는 역할을 한다. 클라이언트 쪽에서 IDL stub은 서버 객체의 프록시 역할을 수행한다. 클라이언트와 서버가 다른 언어로 작성되더라도 클라이언트의 언어로 생성된 stub 코드를 통하여 클라이언트는 서버 객체의 서비스와 통신할 수 있게 된다. 클라이언트의 CORBA ORB는 IOP에 따라 하드웨어 프로세서 종류에 따라 달라질 수 있는 데이터 표현 방식을 표준 스트링으로 변환시

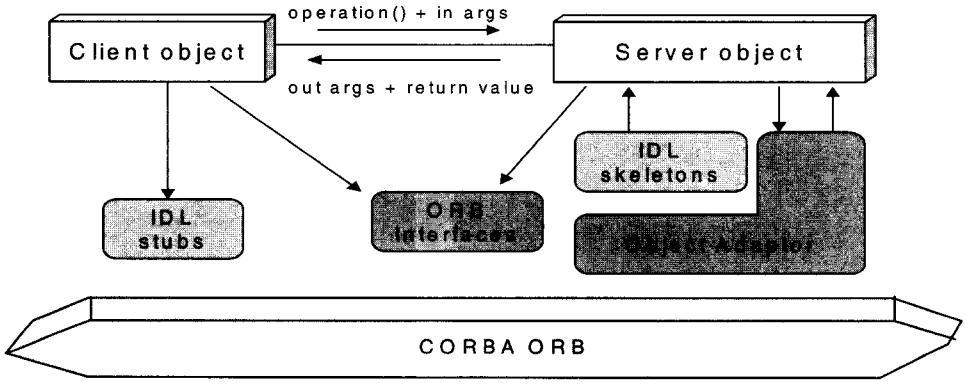


그림 10. CORBA의 구조

켜 서버 객체가 있는 CORBA ORB에게 전달한다. 서버측의 ORB는 다시 표준 스트링을 자신의 시스템에 맞는 데이터 표현 방식으로 변환시키고, 객체 어뎁터에게 적당한 서버 객체를 찾아 이를 처리하도록 요청한다. 객체 어뎁터는 서버 측 CORBA 객체의 규모성(scalability)을 가능하게 하는 것으로, 실제 프로그램상에서 객체의 명령을 수행하는 부분(서번트, servant)과 클라이언트에게 보이는 서버 객체를 분리하여 관리하는 역할을 수행한다. IDL skeleton은 클라이언트 객체의 IDL stub과 유사하게 구현 언어를 추상화시켜주는 역할을 수행하며 객체 어뎁터와 서번트를 연결시켜준다. 구체적으로 C 언어의 경우에는 해당 서번트에 대한 함수 포인터, C++이나 java 언어의 경우에는 서번트 클래스가 상속한 기반 클래스가 된다.

4.4. SDR을 위한 SCA 코어 프레임워크

SDR의 소프트웨어 표준인 SCA에서는 미들웨어로서 CORBA와 더불어, 컴포넌트들을 합성하여 응용 프로그램을 구성하기 위하여 컴포넌트의 구성물의 수행을 뒷받침하는 하부 구조로서 (1) 코어 프레임워크 인터페이스와 (2) 도메인 프로파일을 제공하고 있다. 코어 프레임워크 인터페이스들은 CORBA 미들웨어 위에서 수행되는 응용 객체들을 관리하는 관리 객체들을 위한 인터페이스들로서, 통상적으로 디바이스 제어 프로그램과 응용 프로그램으로 이루어진 내장형 시스템 소프트웨어의 디자인 패턴을 잘 활용한 프레임워크를 제공하고 있다.

도메인 프로파일은 SDR 시스템의 하드웨어와 소프트웨어의 구성 정보를 기술하기 위한 XML 디스크립터 파일들로 구성된다. 이는 설치된 하드웨어와 소프트웨어 컴포넌트의 정보를 알아내는데 사용될 뿐만 아니라, 새로운 소프트웨어를 설치할 때 컴포넌트들의 조합에 대한 설치 정보를 기술하는 부가 정보로서 사용될 수 있다.

5. 결 론

본 고에서는 SDR 시스템을 위한 RTOS와 미들웨어의 설계에 대하여 논하였다. 구체적으로 이들의 개념과 SDR 시스템에서의 필요성과 역할에 대하여 살펴보고, SDR 시스템에 특화되기 위해 필요한 요건, 관련된 표준들에 대하여 살펴보았다. RTOS는 다양한 하드웨어 디바이스 위에서 SDR의 실시간 프로토콜을 신뢰성 있고 안정적으로 처리하는 동시에 탄력적이고 유연성 있게 구동하기 위하여 최소한의 기본적인 추상화 계층으로서 필요하다. 또한 미들웨어는 SDR 분산 노드들의 다양한 이종성을 숨기고 분산 응용의 부분들이 유연하게 상호 작용할 수 있도록 분산성을 숨기는 추상화 계층으로서 필요하다.

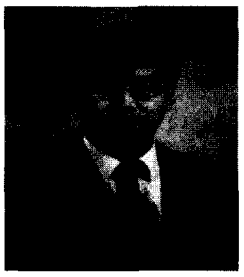
SDR 포럼에서도 SDR 시스템의 소프트웨어 구조의 표준인 SCA에서 RTOS와 미들웨어와 응용 프로그램으로 이루어진 계층적인 소프트웨어의 구조를 제시하고 있다. SDR 시스템은 내장형 실시간 무선 응용 시스템으로서 RTOS와 미들웨어에게 경량성과 실시간성을 중요하게 요구한다. 또한 SDR 시스템은 다양한 구성과 응용을 요구하고 재구성성이 매우 중요하다. 따라서 RTOS가 다양한 방식으로 구성될 수 있도록 컴포넌트 기반 RTOS 기술이 더욱 중요해 질 것이며, 미들웨어에 있어서도 유연한 동적 컴포넌트 배치를 지원하는 컴포넌트 프레임워크의 기술이 더욱 중요해 질 것으로 보인다. 여기에는 컴포넌트의 비기능적 측면인 수행 시간, 고장 감내, 보안 같은 QoS에 대한 명세와 이를 검증하는 도구에 대한 연구도 포함될 것이다.

참고 문헌

[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.



홍성수

1986년 2월: 서울대학교 컴퓨터공학과 졸업 1988년 2월: 동대학원 석사 1994년12월: University of Maryland at College Park, Dept. of Computer Science 박사 1988년 2월~1989년 7월: 한국전자통신연구원 연구원 1994년 12월~1995년 3월:Faculty Research Associate (University of Maryland at College Park) 1995년 4월~1995년 8월: Silicon Graphics Inc. 연구원. 1995년 9월~1997년 9월: 서울대학교 전기컴퓨터공학부 전임강사 1997년 10월~2001년 10월: 서울대학교 전기컴퓨터공학부 조교수 2001년 10월~현재: 동학부 부교수.

관심분야는 실시간 시스템, 실시간 운영체제, 정보가전, 실시간 시스템 설계 방법론, 멀티미디어 시스템, 소프트웨어 공학