SDR 모바일 플랫폼을 위한 SCA 소프트웨어 프레임워크

SCA Software Framework for SDR Mobile Platform

김창기(C.K. Kim) SDR연구팀 선임연구원
이찬용(C.Y. Lee) SDR연구팀 선임연구원
신연승(Y.S. Shin) SDR연구팀 책임연구원

본 논문은 SDR Forum에서 SDR용 모바일 플랫폼 구성을 위한 소프트웨어의 표준으로 인정된 차세대 개방형 소프트웨어 프레임워크인 SCA의 구조에 대해 기술한다. SCA 소프트웨어 프레임워크는 궁극적으로 단일 모바일 플랫폼 환경에서 하드웨어의 변경 없이 무선으로 다운로드한 소프트웨어의 동작으로 다양한 무선 접속환경을 구성할 수 있도록 하는 내장형 시스템 소프트웨어들의 표준을 추구한다. SCA는 분산 객체 모델의 표준인 CORBA Middleware와 더불어 다중 무선접속 응용프로그램을 구성하기 위하여 컴포넌트 구성물의 수행을 뒷받침하는 개방형 소프트웨어 구조로서 크게 RTOS, CORBA Middleware, CF, Portable Software Resource로 이루어진다. 본 고에서는 SCA 소프트웨어 프레임워크의 기본 구조와 기능들을 최신 표준 동향을 기반으로 분석하고, 개발 사례에 대해서도 간단히 소개한다.

I. 서 론

오늘날 통신 기술의 빠른 진보와 사용자들의 서비스 욕구로 인하여 기존의 하드웨어로만 가능했던 많은 기능을 소프트웨어로 구현 가능하게 되었다. 다양한 이동 통신 시스템을 하나의 단말을 이용해서 소프트웨어적인 접근으로 여러 통신방식을 이용할 수 있도록 하는 SDR 기술 역시 이러한 개념에서 출발하였다[1].

SDR(Software Defined Radio)은 하나의 디바이스 기능을 소프트웨어로 정의하고 제어함으로 인해 다양한 무선 접속환경과 새로운 기능들의 업그레이드를 할 경우에 하드웨어의 변동 없이 유연하게 변경 가능하도록 하는 기술이다[2]. 이러한 기능을 만족시키기 위한 소프트웨어는 계층적, 표준화 및 개방형 구조를 가져야 하는데, SDR Forum에서는 이를 위해 Joint Tactical Radio System(JTRS) Joint Program Office(JPO)에 의해서 정의된 SCA (Software Communications Architecture)를 SDR 모바일 플랫폼 구성을 위한 소프트웨어 프레임워크의 표준으로 삼고 있다[3].

SCA 기반으로 구성된 모바일 플랫폼은 사용되어지는 무선 도메인에 관계없이 공통된 개방형 프레임워크를 바탕으로 도메인간의 상호 운용성, 다양한 주파수, 소프트웨어의 이식성(Portability), 재사용성(Scalability) 등을 제공해 준다.

본 논문에서는 SCA 소프트웨어 프레임워크의 구조와 기능에 대해 중점적으로 살펴보도록 한다. 먼저 II장에서는 SCA의 소프트웨어 프레임워크의 배경과 목적 등에 대해서 간략히 소개하고, III장에서

는 SCA 소프트웨어 프레임워크의 주요 기능을 기본 아키텍처를 중심으로 기술하고, IV장에서는 최근 SCA 소프트웨어 아키텍처를 테스트베드를 통해 구현한 사례에 대해 간단히 살펴 보고 마지막으로 결론을 맺는다.

II. SCA 배경 및 목적

JTRS JPO는 원래 미국의 육, 해, 공군의 군사 전술 용도로 차세대 음성, 영상, 데이터 통신 시스템 개발을 목적으로 생성된 프로그램이다. 따라서 이 프로그램은 시스템의 상호운용성의 극대화, 비용의 절감, 운용 비용의 최소화는 물론이고, 기술적으로 유지보수 및 업그레이드의 용이함을 위해서 소프트웨어 programmable, multi-band, multi-mode, 모듈화가 가능한 통신 시스템을 개발하는 것을 추구하고 있다[3].

SCA는 JTRS의 목적을 이루기 위해 JTRS와 Modular Software-programmable Radio Consortium(MSRC)에 의해 만들어진 표준 소프트웨어 아키텍처이다. SCA는 특정 시스템에 한정되는 규격이 아니고 위의 목적을 만족시키는 통신 시스템을 만들기 위한 독립적인 시스템 디자인 프레임워크라고 말할 수 있다. SCA는 다음과 같은 몇 가지 기본 원칙을 가지고 구축되어야 한다[4].

- SCA 기반이 구현된 서로 다른 시스템 사이의 응용 소프트웨어의 이식성을 제공하여야 한다.
- 개발 비용의 초소화를 위해 상용 제품 표준을 최대한 사용하여야 한다.
- 모듈화 소프트웨어의 재사용을 통하여 새로운 시스템 개발의 시간을 단축시킨다.
- 상용 프레임워크와 구조를 발전시키는 것을 토대로 해야 한다.

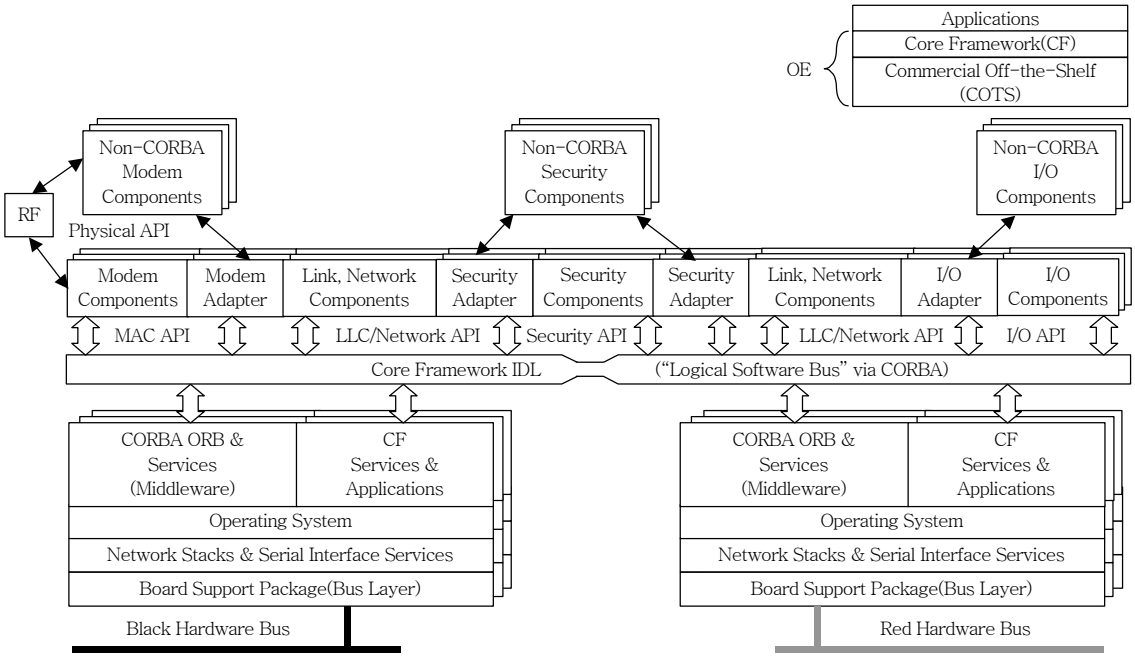
이런 원칙 하에 개발된 SCA의 소프트웨어의 프레임워크는 군사 전술용 응용분야에서 일반 상업용 응용분야에까지 이르게 되었고, 실제 이동통신 분야에서의 SDR 기술에 적용되는 단일 플랫폼 형성을 위한 내장형 제어 소프트웨어의 표준 규격으로 자리잡게 되었다.

한편 SCA가 정부에서 공식적인 상업용 소프트웨어의 표준 프레임워크로 공식화 될 전망과 더불어 많은 회사들이 SCA 프레임워크를 검증하고 개발하는데 참여하고 있다. JTRS JPO는 SCA Ver1.0 규격을 통하여 프레임워크 규격의 개발과 더불어 SCA 규격의 검증을 MSRC를 통하여 Raytheon을 비롯한 여러 회사에 각각 다른 프로토타입을 가지고 수행하게 되었다. 이를 통해 규격의 개발이 더욱 가속화가 되었고 현재 Ver2.2까지 문서로 나와 있다.

SCA 소프트웨어 프레임워크 기술 표준 문서는 크게 4개의 영역으로 나누어져 있는데, 기본적인 소프트웨어 아키텍처 규격(Architecture), 군사 보안용 보충 문서(Supplement), 응용 프로그램 인터페이스(Application Program Interface: API) 보충 문서 그리고 SCA SRD(Support and Rationale Document)로 구성된다. 소프트웨어 기본 아키텍처는 SCA의 전반적인 소프트웨어 구조를 설명하고 있으며, 군사 보안용 보충 문서는 SCA의 군사용도로서 사용될 때 추가로 필요한 요구사항 및 API들을 기술하고, API 보충 문서는 API 개발을 위한 요구사항들을 정의하고 있으며, SRD 문서는 SCA 아키텍처를 구현하기 위한 기술적 이론과 근거를 상세히 설명하고 있다. 본 고에서는 SCA의 기본 아키텍처 문서를 중심으로 SCA 소프트웨어 구조를 살펴 보기로 한다.

III. SCA 소프트웨어 기본 아키텍처

SCA 기본 아키텍처는 소프트웨어 자원간의 상호연결 및 관리를 위한 상용 프레임워크(OS, CORBA (Common Object Request Broker Architecture)) 기반의 개방형, 계층적, 분산 처리 환경을 제공한다. (그림 1)은 SCA 소프트웨어의 기본 아키텍처이다. (그림 1)에서 SCA는 크게 OE(Operating Environment)와 애플리케이션으로 이루어져 있으며, OE는 Commercial Off-the-Shelf(COTS: RTOS



(그림 1) SCA 소프트웨어 구조

(Real Time Operating System)와 CORBA)와 CF(Core Framework)를 말하며, 애플리케이션은 Portable Software Resource를 나타낸다. 즉 아래 4개의 주요 부분으로 구성되어 있다.

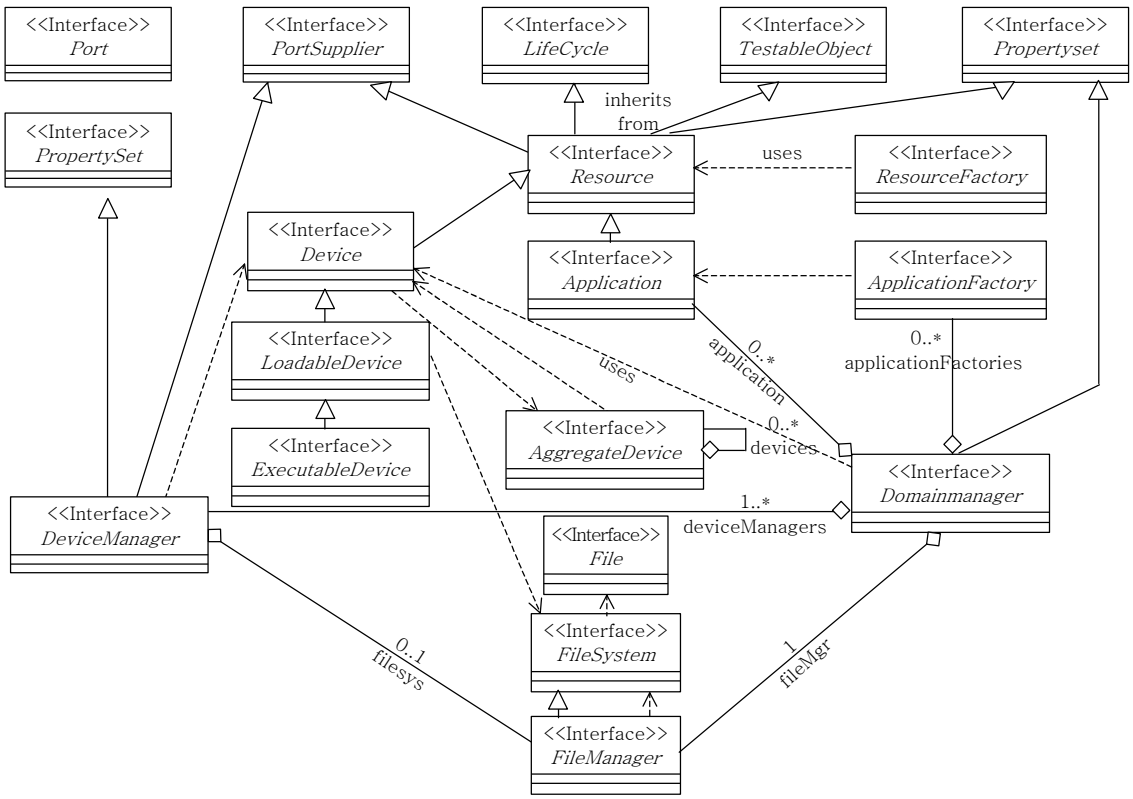
- A Real Time Operating System(RTOS)
- CORBA Middleware
- Core Framework(CF) Services & Application
- Non-core Application(Portable Software Resource)

1. RTOS & CORBA Middleware

RTOS는 소프트웨어 아키텍처의 운영체제로서 Portable Operating System Interface(POSIX) 기반의 처리 환경을 제공한다. SCA는 애플리케이션의 이식성 및 아키텍처의 범용성 등을 지원하기 위해 애플리케이션 환경 프로파일(Application Environment Profile: AEP)을 정의하고, OS는 AEP에서 필수적으로 제공하여야 하는 서비스를 지원하도록 규정하고 있다. POSIX는 IEEE에 의하여 정의된 현재 RTOS의 산업용 표준이다. POSIX 1003.13 표준은 애플리케이션의 모델에 따라서 PSE-51, PSE-52, PSE-53, PSE-54의 4개의 AEP를 정의하고 있다. SCA는 이중에서 SCA 요구사항을 만족시키는 최소한의 POSIX AEP인 Real-time System 프로파일을 제공하는 PSE-52를 채택하고 있다[5].

CORBA Middleware는 분산 처리를 위해 Object Management Group(OMG)에서 제정한 미들웨어의 표준이다. SCA에서는 애플리케이션들 간의 통신 및 애플리케이션과 CF, OS와의 논리적인 소프트웨어 버스(Logical 소프트웨어 버스)를 제공하여 OS로의 접근과 동시에 CF 메시지들을 전달하기 위해서 CORBA를 표준으로 정하여 사용하고 있다. 특별히 CORBA의 여러 표준 중에서 minimum-CORBA를 사용하고 있다[5].

한편 SCA에서는 OE의 요구사항 중의 하나인 Naming Service와 Log Service를 CORBA의 확장 영역에서 제공해주는 CORBA Naming Service와 Log Service를 사용한다. CORBA Naming



(그림 2) Core Framework UML Model 및 Relationship

Service는 ORB 상의 원하는 객체를 찾아 그 객체의 Object Reference를 얻어주는 역할을 한다. 또한 CORBA에서는 Event Service를 제공하여 consumer object와 producer object 사이의 커플링을 줄여줄 수 있다[6].

2. Core Framework Services & Application

Core Framework Services & Application(이하 CF)는 SCA 소프트웨어 아키텍처 중에서 가장 핵심이 되는 (“core”) 부분이다. 이는 core에 대해 상대적인 개념으로서의 “non-core”(non-CF) Software Radio Applications들에게 COTS 이하의 소프트웨어와 하드웨어에 대하여 추상화 기능을 제공하는 개방된 인터페이스와 서비스를 말한다. (그림 2)는 CF의 구성요소와 상호간의 관계를 Unified Modeling Language(UML)를 사용하여 나타낸 그림이다. CF는 기능과 역할에 따라 크게 4개의 부분으로 구성되어 있는데, 각각은 아래와 같다.

- 기본 응용 인터페이스(Base Application Interface): *Port, LifeCycle, TestableObject, PortSupplier, PropertySet, ResourceFactory, and Resource*
- 프레임워크 제어 인터페이스(Framework Control Interface): *Application, ApplicationFactory, DomainManager, Device, LoadableDevice, ExecutableDevice, AggregateDevice and Devicemanager*
- 프레임워크 서비스 인터페이스(Framework Service Interface): *File, FileSystem, FileManager, and Timer*
- 도메인 프로파일(Domain Profile)

<표 1> Base Application Interface

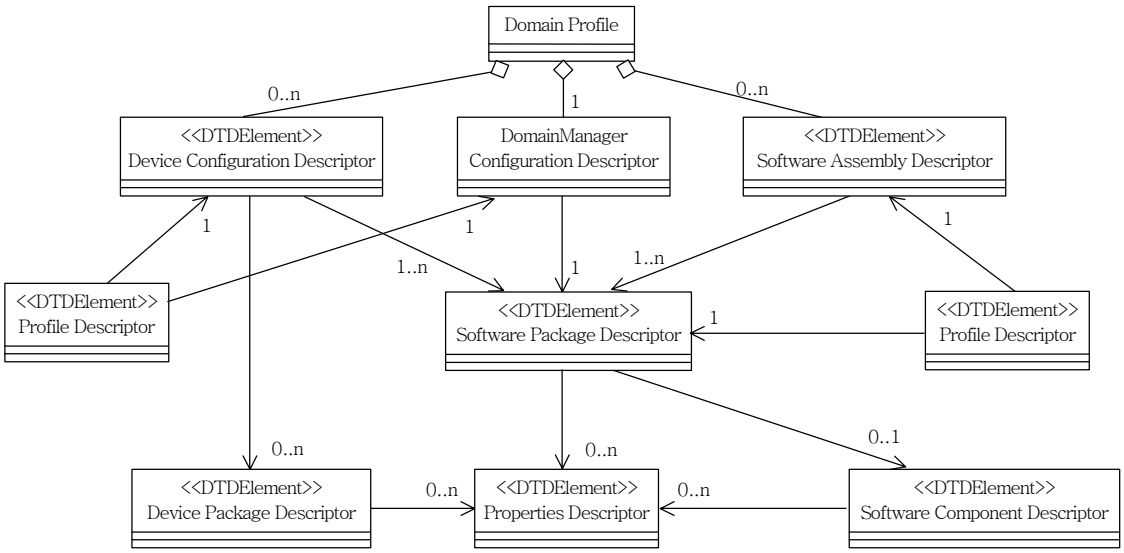
인터페이스	오퍼레이션 및 설명
Port	포트들간의 관리를 위한 오퍼레이션을 제공한다. connectPort(), disconnectPort()가 있다.
PortSupplier	특정 객체에 대한 포트의 참조를 얻을 수 있는 메커니즘을 제공한다. getPort() 오퍼레이션이 있다.
LifeCycle	특정 객체의 초기화 및 해제를 관리한다. CF의 ApplicationFactory를 사용하며 Constructor 이후 초기화 및 종료시 메모리 해제, File Close를 담당한다. initialize(), releaseObject()가 있다.
Testable-Object	특정한 객체에 대한 Built-In Tests(BIT)를 제공한다. 테스트 Id 및 Value는 도메인 프로파일의 Properties Descriptor에 따르고 runTest()가 있다.
PropertySet	특정 객체에 대한 PropertyXML 파일에 정의된 Configuration 파라미터의 값을 요구 내지 변경하는 메커니즘을 제공한다. query(), configuration()이 있다.
Resource-Factory	응용 Resource를 생성하고 삭제하는 데 사용. createResource(), releaseResource(), shutdown()이 있다.
Resource	컴포넌트의 제어와 구성을 위한 공통된 API를 제공한다. Application이나 Device를 설정하기 위해 HCI에 의해 사용된다. start(), stop()이 있다.

Base Application Interface는 모든 소프트웨어 애플리케이션에서 사용될 수 있는 기본이 되는 인터페이스이다. <표 1>은 Base Application Interface의 구성요소와 기본 오퍼레이션을 나타낸 것이다. Base Application Interface에 대한 세부적인 UML 다이어그램과 각각의 인터페이스에서 제공되는 오퍼레이션에 대한 설명은 SCA 소프트웨어 아키텍처 규격문서[4]를 참조하도록 한다.

Framework Control Interface는 (그림 2)에서 보듯이 크게 Domain Management, Device, Device Management Interface로 나눌 수 있다. *Applicaton, ApplicationFactory, DomainManager*로 구성된 DomainManagement는 하나의 도메인 내에서 도메인 구성요소(Application, Devices, Device Manager, Event Channel)들의 등록/등록해제, 설치/삭제, 검색 등의 서비스를 제공한다. Device Interface는 *Device, LoadableDevice, Executable-Device, AggregateDevice*로 구성되고, 도메인 내의 Logical Device를 관리하는 기능을 제공한다. DeviceManagement Interface는 *DeviceManager* 인데, 이는 Logical Device를 생성하고, 생성된 Logical Device에 서비스 애플리케이션을 시작시키는(Launch) 기능을 비롯한 노드를 관리한다. Framework Control Interface의 UML 다이어그램과 각각의 오퍼레이션 및 설명과 상호관계는 SCA 규격을 참조하도록 한다. 기본적으로는 Base Application Interface를 상속하고, 추가적으로 위에서 설명한 각각의 기능에 따른 오퍼레이션이 존재한다.

Framework Service Interface는 Core Application과 Non-core Application 모두에 서비스를 제공하는 인터페이스로, *File, FileSystem, FileManager*가 있다. *File*은 파일시스템 안의 파일에 대한 읽고, 쓰기(read, write, sizeOf, close, setFilePoint) 등의 기능을 제공한다. FileSystem은 CORBA 오퍼레이션 제공을 통하여 물리적으로 떨어진 파일시스템에 원격접속을 허용하고 기본적인 파일 액세스에 관련된 기능(remove, copy, exist, list, create, open, mkdir, rmdir, query)을 제공한다. *FileManager*는 한 개 이상의 분산된 파일 시스템을 관리하는 기능을 갖는데, 이 *FileManager*는 실제 파일 시스템이 두 개 이상으로 떨어져 있어도 하나의 파일시스템에만 존재한다. 이것을 “ferated file system”이라고 한다. *FileManager*는 이러한 파일 시스템을 관리하는 기능(mount, unmount, getmounts)을 제공한다[7].

마지막으로 Domain Profile은 하나의 SCA 시스템 도메인을 구성하는 하드웨어 디바이스와 소프트웨어 컴포넌트들을 잘 나타내주는 여러 개의 파일의 집합이라고 할 수 있다[3]. 도메인 프로파일은 하드웨어 디바이스와 소프트웨어 컴포넌트의 위치, 상호의존관계, 용량, 식별(Identity), 능력(Capability) 등을 기술하고 있으며, 여러 개의 eXtensible Markup Language(XML) 디스크립터(Descriptor) 파일들로 구성되고, 이들 파일들은 Document Type Definitions(DTD) 포맷을 따른다[3]. (그림 3)은 도메인 프로파일을 구성하는 XML의 타입과 상호관계를 나타낸다.



(그림 3) 도메인 프로파일 XML 타입과 상호 관계

(그림 3)에서 도메인 프로파일은 총 7개의 파일로 구성되어 있다. 각각의 종류와 구성내용을 살펴보면 아래와 같다.

• Software Packaged Descriptor(SPD)

SPD는 특정 컴포넌트의 구현정보를 기술한다. 이름, 저자, 속성파일, 구현 코드정보 그리고 하드웨어와 소프트웨어의 의존관계와 같은 소프트웨어 패키지에 대한 일반적인 정보를 포함한다. SCD의 인스턴스에 대하여 참조를 가질 수 있다.

• Software Component Descriptor(SCD)

SCD는 특정 컴포넌트(Resource, Resource-Factory, Device)에 대한 CORBA 인터페이스 정보를 기술한다. 컴포넌트가 상속/지원하는 인터페이스와 지원하는 포트도 포함된다.

• Software Assembly Descriptor(SAD)

SAD는 하나의 응용 프로그램을 이루는 소프트웨어 컴포넌트들의 배치와 연결 정보를 기술한다. 시스템에 응용프로그램을 인스톨하는 것은 하나의 SAD 파일을 인스톨하는 것으로 볼 수 있다. SAD는 해당 컴포넌트에 대한 배치 정보를 얻기 위해 SPD 파일의 인스턴스에 대한 하나 이상의 참조를 가진다.

• Properties Descriptor

Properties Descriptor는 소프트웨어 패키지와 디바이스 패키지에서 사용 가능한 속성(Properties)에 대한 정보를 포함하고 있다. 즉 컴포넌트의 속성 정보인 구성(configuration), 테스트(test), 실행(execute), 할당 타입(allocation types) 등을 포함하고 있다.

• Device Package Descriptor(DPD)

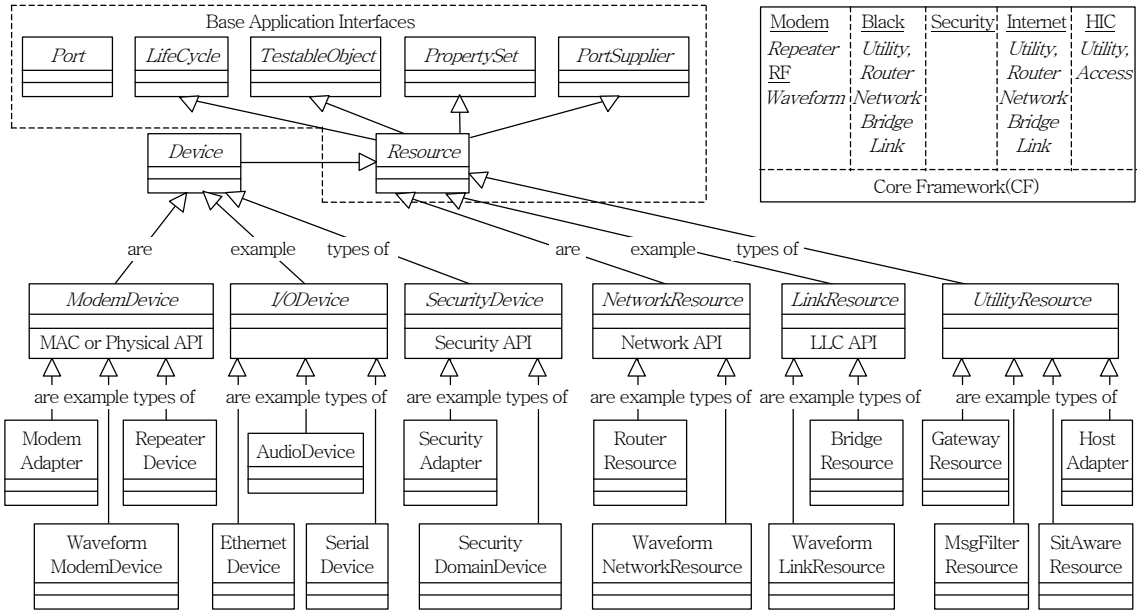
DPD는 Device Profile의 한 부분으로 Device의 특정 속성 즉 디바이스 종류, 시리얼 넘버, 메모리 크기 등의 정보를 포함하고 있으며, 이들 정보는 HCI를 통하여 시스템의 운용/유지 보수를 위한 운용자들에게 제공되는 정보이다.

• Device Configuration Descriptor(DCD)

DCD는 *DeviceManager* 인터페이스를 지원하는 컴포넌트를 포함하여 노드를 초기화시키는 데 사용되어야 할 컴포넌트들에 대하여 기술한다. 또한 DomainManager를 찾는 방법, 디바이스에 대한 구성 정보(Log, FileSystem)도 포함한다. DCD는 DPD의 인스턴스들에 대한 참조도 가질 수 있다.

• Profile Descriptor

Profile Descriptor는 SAD, SPD, DMD, DCD



(그림 4) Portable Software Resource 구조 및 CF 인터페이스와의 관계

파일을 식별하는 데 사용된다. 즉 위 파일들의 절대 경로를 포함한 파일명을 포함하고 있다. Profile Descriptor는 profile attribute를 가지는 모든 CF 인터페이스로 만들어 질 수 있다.

- DomainManager Configuration Descriptor (DMD)

DMD는 DomainManager의 구성정보(서비스와 DeviceManager의 SPD)를 포함하고 있다.

3. Non-core Application(Portable Software Resource)

SCA는 응용 소프트웨어의 이식과 동적 인스턴스(Dynamic Instantiation)가 매우 증대하기 때문에 CORBA 기반의 객체지향 오픈 아키텍처를 규정하고 있다. 즉 하드웨어에 독립적인 응용 소프트웨어를 위한 인터페이스를 지향하고 있다.

Portable Software Resource는 (그림 1)의 SCA 소프트웨어 구조에서 CF IDL 상위의 User-Orient한 "Non-core" Application을 말한다. 이는 "core"의 상대적인 개념으로서의 의미일 뿐이고 이 역시 중요한 부분이다. 이는 모뎀 디지털 신호처리(modem-level digital signal processing), 네트워크 프로토콜 처리(network-level protocol processing), 라우팅(interwork routing), 외부 액세스(external access), 보안(security), 내부의 유틸리티(embedded utility)를 포함하는 광범위의 통신 기능을 수행한다. 모든 Non-core Application은 CF의 *Resource*와 *Device* 클래스의 하부 클래스(subclass)들로 구성된다. 또한 *Resource* 는 2절에서 살펴본 바와 같이 CF의 기본 응용 인터페이스를 상속 받는다. (그림 4)는 Portable Software Resource의 구성요소 및 CF 인터페이스와의 관계를 나타낸 것이다.

다수의 *Resource*들로 구성된 "Non-core" Application은 하나의 무선 도메인에서의 Programmable Modular Communication System (PMCS) 참조 모델에 근거하여 기능별로 구분한다면 크게 (그림 4)에서와 같이 *ModemDevice, I/ODevice, SecurityDevice, NetworkResource, LinkResource, UtilityResource* 등으로 나눌 수 있으며, 각 Resource들의 기능은 아래와 같다.

• *ModemDevice*

*ModemDevice*는 모든 모뎀에 공통적인 물리적 인터페이스와 제어를 위한 표준을 제공한다. CF의 Base Application Interface는 MAC, LLC API를 통하여 *ModemDevice*에 확장된다.

• *NetworkResource and LinkResource*

*NetworkResource and LinkResource*는 네트워크 애플리케이션에 정보 전달 및 특정 서비스를 지원해 준다(예를 들면 Link-LLC API and Network-MAC API). CF의 Base Application은 API를 통해 확장된다.

• *I/ODevice*

I/O Device는 시스템 하드웨어와 외부 물리적 인터페이스로의 접근을 제공한다. I/O 인터페이스에서 제공해 주는 오퍼레이션은 하드웨어와 지원되는 외부 물리적 인터페이스에 따라 변할 수 있다. 전형적인 I/O Device로는 SerialDevice, EthernetDevice, AudioDevice 등이 있다.

• *SecurityDevice*

SecurityDevice는 CF의 Base Application Interface를 확장하여서 암호화, 복호화, 인증, 키관리 등의 보안서비스 등을 제공한다. Waveform 사이에서 요구되는 Transmission security(TRANSSEC)와 communication security(COMSEC)가 다르므로 보안의 범위 역시 waveform에 따라 달라질 수 있다.

• *UtilityResource*

UtilityResource는 CF의 Base Application Interface를 확장하여 메시지 변환, 네트워크 게이트웨이, 호스트 어댑터 등의 오퍼레이션을 수행한다. 역시 지원되는 내장형 애플리케이션과 호스트 인터페이스 프로토콜에 따라 변할 수 있다.

IV. 개발 사례

SCA는 SDR 기술의 핵심이 되는 표준으로 자리잡고 있는 가운데 캐나다 Communication Research Center(CRC)의 The Military Satellite Communication(RMSC) 그룹에서는 SDRF의 후원으로 SCA 규격의 참조 구현(Reference Implementation: RI) 프로젝트를 2000년에 처음으로 실시하였고(SCARI), 이를 통해 실제 구현 입장에서 여러 가지 문제를 고려하면서 SCA Core 프레임워크의 수정 보완을 추진하였다. SCARI의 목적은 아래와 같다[8].

- SCA 규격문서의 모호함을 줄인다.
- 예제를 통하여 SCA 소프트웨어 아키텍처의 이해를 증가시킨다.
- 하나의 구현 사례를 통해 SDR 기술의 상용화를 가속시킨다.
- 참조 구현을 통하여 개발자들로 하여금 잠재적인 상호 운용성을 증가시킨다.

SCARI를 통하여는 SCA 규격의 기술적인 부분의 구체적이고, 체계적인 분석과 인터페이스로 정의된 많은 부분들의 실제 behavior들을 정의하였다. SCARI 프로젝트는 SCA2.1 규격을 기반으로 만들어졌으며, 약 60,000라인의 코드와 30개의 예제 프로그램과 UML 시퀀스 다이어그램을 포함하는 300페이지 이상의 기술문서로 이루어져 있다. 또한 CRC는 SCARI를 통하여 SCA 규격의 CF의 문제점을 설명하고 변경 제안을 JTRS에 제출하였다. SCARI 프로젝트의 모든 결과(소스, 기술문서) 등은 CRC 홈페이지에서 무료로 이용할 수 있다. SCARI의 SCA CF 개발 환경은 아래와 같다.

Software

Operating System: Linux Redhat 7.2

Development Kit: GNU C++ part of Linux Redhat 7.2

CORBA Orb 2.3 compliant: javaIDL part of Sun Java JDK 1.4

XML support: Java API for XML part of Sun Java JDK 1.4

Hardware

1 or 2 Personal Computers equipped with Intel Pentium processors included

- 10/100Mb Network interface card
- Full-duplex SoundBlaster soundboard

SCARI 프로젝트는 현재 SCARI2, SCARI-Hybrid, SCARI++로 확장되어 계속해서 진행되고 있다[9]. 이는 개발 환경 및 개발 툴의 변화로 SCA의 CF 성능을 발전시켜 가기 위한 것이다. SCARI2는 현재 SCA 규격의 최종 버전인 SCA2.2를 기준으로 SCARI(V2.1)를 업그레이드 한 것이다. SCARI-Hybrid는 기존의 SCARI, SCARI2에서는 CF 자체의 개발에 중점을 두어 소프트웨어 개발툴을 Java를 사용한 것에 비해, CF의 성능을 고려하여 Waveform 애플리케이션을 Java Native Interface(JNI)통하여 Java Runtime Environment에서 프로세싱 해야 하는 문제점을 고려하여 CF의 구성요소 중에서 데이터 프로세싱과 직접관계가 있는 부분은 C++ 코드로 사용하여 개발한 것이다. SCARI++는 SCARI-Hybrid를 한 단계 더 발전시킨 것으로, 개발 소프트웨어를 C++를 사용하고, Real-time ORB를 사용하여 CF의 성능을 최적화 하였다.

참고로 SCARI, SCARI2, SCARI-Hybrid의 일부인 Java로 개발된 부분에서의 CORBA는 Java에서 지원되는 JDK를 사용하였으며, SCARI-Hybrid의 일부와 SCARI++의 C++로 개발된 부분에서의 CORBA는 TAO real-time CORBA를 사용하였다. 또한 SCA-Hybrid와 SCARI++는 기본 환경인 리눅스 뿐만 아니라 SCA 규격에서 요구하는 Real-time OS를 만족하기 위해 VxWorks에서도 테스트를 한 것으로 전해진다. 현재 SCARI2까지는 웹 사이트에서 오픈하는 것으로 되어 있고(현재는 SCARI만 공개됨), 나머지 SCARI-Hybrid, SCARI++는 상업용으로 알려져 있다.

V. 결론 및 향후 연구 방향

본 논문에서는 SDR 모바일 플랫폼 기술의 표준인 SCA 소프트웨어 프레임워크의 구조와 기능요소를 분석하였다. 그 중에서도 소프트웨어 기본적인 아키텍처에 대해서 중점적으로 살펴 보았다. 무선 도메인에 관계없이 공통된 개방형 프레임워크를 바탕으로 한 SCA 소프트웨어 기본 아키텍처는 도메인 간의 상호 운용성, 다양한 주파수, 소프트웨어의 이식성, 재사용성 등을 제공해 주기에 적합하다.

SCA 소프트웨어 기본 아키텍처는 크게 RTOS, CORBA Middleware, CF 그리고 Portable Software Resource로 구성되어 있다. 이중에서 Non-Core Application인 Portable Software Resource는 각 도메인마다 달라질 수 있는 응용 애플리케이션이다. 이에 반해 CF는 Base Application Interface를 비롯하여 몇 개의 Interface를 Non-core Application에게 제공해 주는 도메인에 관계없이 공통으로 제공되는 "core" 서비스이다.

한편 캐나다 CRC의 SCARI 프로젝트는 SCA 소프트웨어 기본 아키텍처를 처음 개발하는 데 도움이 되는 하나의 좋은 예를 보여 준다. 하지만 실제 SDR 기술을 구현하는 데 접목하기 위해서는 리눅스가 아닌 RTOS 상에서의 구체적인 검증이 있어야 할 것이며, 더 나아가 성능적인 측면에서도 한층 고려하여서 개발하여야 할 것이다[10].

향후 연구과제로는 CRC의 CF 구현을 바탕으로 실제 SDR 응용을 위한 Mobile 플랫폼이라는 상황을 고려한 RTOS와 minimumCORBA를 만족하는 범위에서 *ModemDevice*를 비롯한 Non-core Application을 접목한 구현을 시도하여야 하며, 또한 객체지향 시스템의 설계 및 구현의 관점에서 보면 SCA 규격은 실제 구현 레벨에서 중요한 부분을 차지하는 컴포넌트 및 배치 수준에서의 소프트웨어 재사용을 위한 설계에 대한 연구도 고려해 볼 만하다.

참 고 문 헌

[1] 황경호, 조동호, "Software Defined Redio 기술," *Telecommun. Review*, 제10권, 제1호, 2000. 1.~2., pp. 130-143.

[2] SDR forum: http://www.sdrforum.org.

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

[4] Modular Software-programmable Radio Consortium Software Communication Architecture Specification MSRC-5000SCA V2.2, Joint Tactical Radio System, Nov. 17, 2001.

[5] Modular Software-programmable Radio Consortium Support and Rationale Document for Software Communication Architecture Specification MSRC-5000SRD V1.2, Dec. 21, 2000.

[6] P.A. Eyermann and M.A. Powell, "Maturing the Software Communications Architecture for JTRS," *Proc. of MILCOM*, Vol. 1, Oct. 2001, pp. 28-31.

[7] Raytheon Company Joint Tactical Radio System (JTRS) SCA Developer's Guide, Rev1.1, 18 June 2002.

[8] CRC web site: http://www.crc.ca/

[9] K.V. Davis, "JTRS-an Open, Distributed-object Computing Software Radio Architecture," *Proc. 8th Digital Avionics Systems Conference,* Vol. 2, Oct. 1999, pp. 24-29.

[10] 김세화, 홍성수, 장래혁, "SDR 컴포넌트의 동적 배치를 위한 SCA 기반 컴포넌트 프레임워크의 설계," 한국과학재단 목적기초연구(R01-1999-00206), 2002.