Introduction to the Software Communications Architecture

Wireless Innovation Forum

May 26, 2011

Prepared by:

Research Advanced Radio Systems
Satellite Communications and Radio Propagation Research
Communications Research Centre Canada
Outline

- SCA is Component Based Development
- SCA Overview
- SCA APIs
- Summary
Outline

- SCA is Component Based Development
- SCA Overview
- SCA APIs
- Summary
Component Based Development Overview

- From a software development perspective, the SCA is a Component-Based Development architecture.

- What is Component-Based Development?
  - An architecture for the creation, integration, and re-use of software components.
  - A software development paradigm where the smallest unit of software is a component.
  - With CBD, an application is ‘assembled’ using software components much like a board is populated with hardware components.

- CBD is currently the most popular programming paradigm.
  - Microsoft’s CBD is the “.NET” framework.
  - Sun Microsystems’s CBD is the “EJB” framework.
  - OMG’s CBD is the “CCM” framework.
  - Software Defined Radio CBD is the “SCA” framework.
CBD’s goal is to apply the hardware development paradigm to software

- Select software components from a ‘spec-sheet’ catalogue
  - Describe how to influence behaviour (configuration properties)
  - Describe how to interface (ports)
  - Describe resource requirements (capability properties)
  - Describe resource consumption (capacity properties)

CBD Requirements:

- Component Model
  - Well defined standards and conventions
  - Design rules that must be obeyed by all components

- Component Framework
  - Support and enforce component models (development and runtime)
  - Support infrastructure (runtime)
  - Can be seen as a mini Operating System
Characteristics of a Software Component

- A small, reusable module of binary code that performs a well-defined function
- A black-box that hides the component internal implementation, but explicitly exposes its external interfaces
- Designed, implemented, and tested as a unit before it is used in an application
- Predictable, reusable, replaceable, upgradable, extendable, etc.

Interfaces of a Software Component

- Clean separation between interfaces and implementations
- Interfaces become contracts between components that ‘require’ access to certain operations and components that ‘implement’ them
Outline

- SCA is Component Based Development
- SCA Overview
- SCA APIs
- Summary
What is the SCA?

- The SCA was created for the US DoD Joint Tactical Radio System (JTRS) program
  - Created by the Modular Software-programmable Radio Consortium (MSRC): Raytheon, BAE Systems, Rockwell Collins, and ITT

- The goal of the SCA is to facilitate the reuse of waveform applications across different radio sets

- Central software piece in a radio, the “SDR operating system”

- The SCA is not a system specification!
  - Provides an implementation-independent set of rules that constrain the design of systems
Software Communications Architecture Overview

Application Domains

- The SCA is independent of the application domain

SCA Core Framework
Software Communications Architecture Overview

Application Domains

- The SCA is independent of the application domain
- Different applications are supported by domain-specific APIs
Software Communications Architecture Overview

Application Domains

- The SCA is independent of the application domain
- Different applications are supported by domain-specific APIs

SCA Core Framework

Automotive APIs

Robotic APIs

Base Station APIs

JTRS Waveform Applications

JTRS APIs
Software Communications Architecture Overview

SCA Functional Groups

SCA Core Framework (radio management)

SCA Applications

CORBA Middleware

SCA Devices

POSIX APIs SCA AEP

Real-Time Operating System (RTOS)

Digital Hardware

RF Hardware

Software Platform

Hardware Platform
Software Communications Architecture Overview

SCA Functional Groups

SCA Core Framework (radio management)

SCA Devices

RTOS (Real-Time Operating System)

CORBA Middleware

POSIX APIs

SCA AEP

SCA Applications

Hardware Platform

Digital Hardware

RF Hardware

Software Platform
Software Communications Architecture Overview

**SCA Functional Groups**

- Software components are partitioned into three groups: Radio Management, Devices and Applications
  - 1. Radio Management
    - DomainManager, DeviceManager, ApplicationFactory, Application
    - Used to install/uninstall/deploy/configure applications, health monitoring, introspection, etc.
    - Used to enforce the lifecycle of SCA Components
  - 2. Devices
    - Device, LoadableDevice, ExecutableDevice (provide access to hardware components)
    - DeviceManager (used to control radio nodes)
    - Devices can be started, stopped, configured, queried, tested, etc
  - 3. Applications
    - An application is composed of Resources
    - Deployable software implementation of communication standards: FM LoS, EPLRS, Link 16, etc
    - Applications can be started, stopped, configured, queried, tested, etc
Outline

- SCA is Component Based Development
- SCA Overview
- SCA APIs
- Summary
SCA APIs – Application Control

SCA applications

SCA Core Framework (radio management)

Real-Time Operating System (RTOS)

Digital Hardware
RF Hardware

SCA Devices
SCA AEP

POSIX APIs
SCA APIs

Software Platform

Hardware Platform

CORBA Middleware
An application can be viewed as a single **Resource**:  
- It has a certain combination of input, output, and control ports that need to be connected to other components  
- Its behavior can be altered through configuration properties
SCA Resource

- A *Resource* is much like a hardware component
  - It has a certain combination of input, output, and control ports that need to be connected to other components
  - To use a *Resource* in an application, it must be deployed onto a *Device*
  - A *Resource* has requirements a *Device* must meet
    - Capability requirements: OS, Processor, etc.
    - Capacity requirements: MIPS, kilo bytes of memory, etc.

- The behavior of a *Resource* can be altered by changing the value of its configuration properties
  - Ex: code rate
SCA Resource (cont)

- Simple or complex signal processing function

- Granularity: fine or large
  - An FFT Resource or a DAB™ Resource

- A Resource is always a port supplier
  - API to get a port
  - A port is identified by a named string
  - May provide ports for connection to other components
  - May provide ports to give access to a particular API
SCA Resource (cont) API

- `PortSupplier`
  - `getPort()`

- `LifeCycle`
  - `initialize()`
  - `releaseObject()`

- `PropertySet`
  - `configure()`
  - `query()`

- `TestableObject`
  - `runTest()`

- `Resource`
  - `identifier : string`
  - `start() : void`
  - `stop() : void`

Inherits from:

- `PortSupplier`
- `LifeCycle`
- `PropertySet`
- `TestableObject`
Digital Hardware

RF Hardware

Real-Time Operating System (RTOS)

SCA Core Framework (radio management)

SCA Devices

SCA applications

SCA Core

CORBA Middleware

POSIX APIs

SCA AEP

Hardware Platform

Software Platform

Device Overview

Communications Research Centre Canada • Centre de recherches sur les communications Canada • www.crc.ca
**Device Overview**

- **Devices** are software proxies for hardware devices and/or can be target for software deployment

- There are three kinds of *Devices*:
  - **Device**:  
    - This kind of *Device* is generally used for getting access to autonomous hardware device (ex: Modem, Audio, GPS receiver)
  - **LoadableDevice**:  
    - This kind of *Device* is generally used as a proxy for a device like an FGPA
  - **ExecutableDevice**:  
    - This kind of *Device* is generally used as a proxy for GPP
There are three kinds of Devices:

- **Device:**
  - This kind of *Device* is generally used for getting access to autonomous hardware device (ex: GPS receiver)

- **LoadableDevice:**
  - This kind of *Device* is generally used as a proxy for a device like an FGPA

- **ExecutableDevice:**
  - This kind of *Device* is generally used as a proxy for GPP
Device API

<<Interface>>

Resource

<<Interface>>

Device

- usageState : UsageType
- adminState : AdminType
- operationalState : OperationalType
- softwareProfile : string
- label : string
- compositeDevice : AggregateDevice

allocateCapacity()

deallocateCapacity()
ExecutableDevice API

```
<<Interface>>
loadableDevice

  load()
  unload()
```

```
<<Interface>>
executableDevice

  execute(name : in string, options : in Properties, parameters : in Properties) : ProcessID_Type
  terminate(processId : in ProcessID_Type) : void
```
SCA APIs – Radio Management Control

SCA Radio Management

- SCA Core Framework (radio management)
- CORBA Middleware
- POSIX APIs SCA AEP
- SCA Devices
- SCA applications
- RTOS
- Digital Hardware
- RF Hardware

Software Platform

Hardware Platform
Radio Management Components

- The Radio Management is performed by four components:
  - DomainManager, ApplicationFactory, Application, and DeviceManager

- DomainManager is the controller of the radio

- ApplicationFactory is used to instantiate an application and to provide an Application component

- Application is used to control a deployed application
DomainManager

- DomainManager is the central component for radio management
  - Accepts registration of DeviceManagers and Devices
  - Performs interconnections between node components when specified by a DeviceManager
  - Responsible for the installation of applications
  - Offers introspection services
  - Used by UI to control/monitor the radio
SCA APIs – Radio Management Control

DomainManager (cont)

- Can be started at any time during the boot of a radio
  - Can be started using a script or a DeviceManager
  - There is no SCA requirement regarding the boot sequence of a Domain Manager
  - SCARI allows the DomainManager and the DeviceManager to be booted in parallel

- Used to install applications
  - Validates application package for existence of all files referred
  - No standard for the format of an application package
    - SCARI approach uses an archive file (e.g. zip)
  - Creates an ApplicationFactory for the installed application
SCA APIs – Radio Management Control

DomainManager (cont)

- Receives registration of every DeviceManager, Device, and Service

- Provides two event channels
  - One that every Device must use for reporting state changes
  - One that reports radio management events

- Used for introspection of the radio
  - Listen to events being generated
  - Get list of installed applications
  - Get list of instantiated (deployed) applications
  - Provide a list of registered DeviceManagers
  - Etc.
SCA APIs – Radio Management Control

DomainManager API

```java
<<Interface>>
PropertySet

configure()
query()

<<Interface>>
DomainManager

identifier : string
deviceManagers : DeviceManagerSequence
applications : ApplicationSequence
applicationFactories : ApplicationFactorySequence
fileMgr : FileManager
domainManagerProfile : string

registerDevice/registeringDevice : in Device, registeredDeviceMgr : in DeviceManager) : void
registerDeviceManager/registerDeviceManager : in DeviceManager) : void
unregisterDeviceManager/unregisterDeviceManager : in DeviceManager) : void
unregisterDevice/unregisterDevice : in Device) : void
installApplication/install(Application/profileFileName : in string) : void
uninstallApplication/uninstallApplication(applicationId : in string) : void
registerService/registerService : in Object, registeredDeviceMgr : in DeviceManager, name : in string) : void
unregisterService/unregisterService : in Object, name : in string) : void
registerWithEventChannel/registerWithEventChannel : in Object, registeringId : in string, eventChannelName : in string) : void
unregisterFromEventChannel/unregisterFromEventChannel(unregisteringId : in string, eventChannelName : in string) : void
```
**ApplicationFactory**

- **ApplicationFactory** is responsible for instantiating specific types of applications
  - Deploys an application
  - Creates an Application object for each instance (deployed) application
  - Application object is used as a proxy to the deployed application
DomainManager creates an ApplicationFactory for each type of installed application

- Each ApplicationFactory manages one type of application
  - Ex: FMLoS, EPLRS, Link 16, etc.

ApplicationFactory can create several instances of a same type of application

- Ex: FMLoS 1, FMLoS 2, etc.
SCA APIs – Radio Management Control

ApplicationFactory API

```
<<Interface>>
ApplicationFactory

- name : string
- identifier : string
- softwareProfile : string

create(name: in string, initConfiguration: in Properties, deviceAssignments: in DeviceAssignmentSequence): Application
```

`<<CORBATypeDef>>`
DeviceAssignmentSequence

`<<Interface>>`
Application

`<<CORBAEnum>>`
ErrorNumberType
An Application component is used to:

- Start/stop the signal processing (delegates to Assembly Controller)
- Change the behavior of the application through configuration (delegates to Assembly Controller)
- Release the application which terminates components and frees acquired capacity
SCA APIs – Radio Management Control

Application API

```
<<Interface>>
Application

profile : string
name : string
componentNamingContexts : ComponentElementSequence
componentProcessIds : ComponentProcessIdSequence
componentDevices : DeviceAssignmentSequence
componentImplementations : ComponentElementSequence
```

Resource
SCA APIs – Radio Management Control

**DeviceManager**

- A radio is composed of many nodes
- Each node runs a *DeviceManager*
  - CRC has submitted a change proposal to allow any number of nodes to report to a same *DeviceManager*
- Each *DeviceManager* is responsible for booting a node as described in the node assembly descriptor
- Each *DeviceManager* must register to their *DomainManager*
**DeviceManager (cont)**

- **A Radio Node**
  - Provides access to a set of collocated *Devices*
  - For example, a PowerPC board in a Compact PCI Chassis can be considered a radio node in a SDR platform
  - In fact, any device capable of running some CORBA enabled code upon power-up can be considered a node

- **A *DeviceManager* is responsible for a single radio node**
  - Launch/shutdown node components (*Devices* and *Services*) as the *ApplicationFactory* does for the components (*Resources*) of an application
  - Registers to a *DomainManager*, making its node components known to the radio where they become available to applications
Booting a node:

- Reads the node assembly descriptor and launches all components of the node
- DeviceManager waits for registration of the launched Devices and Services components
- DeviceManager reports to a DomainManager
  - Can wait for its DomainManager to be started
  - Accepts registration of its components even when not registered with its DomainManager
DeviceManager API

- `configure()`
- `query()`
- `getPort()`

- `deviceConfigurationProfile : string`
- `fileSys : FileSystem`
- `identifier : string`
- `label : string`
- `registeredDevices : DeviceSequence`
- `registeredServices : ServiceSequence`

- `registerDevice(registeringDevice : in Device) : void`
- `unregisterDevice(registeringDevice : in Device) : void`
- `shutdown() : void`
- `registerService(registeringService : in Object, name : in string) : void`
- `unregisterService(registeringService : in Object, name : in string) : void`
- `getComponentImplementationId(componentInstantiationId : in string) : string`
Outline

- Component Based Development Overview
- SCA is Domain Independent
- SCA APIs
- Summary
The SCA is a Component-Based Development architecture

SCA components are ‘assembled’ in SCA applications

The SCA is independent of the application domain

The goal of the SCA is to facilitate the reuse of waveform applications across different radio sets

Software components are partitioned into three functional groups:

- Devices, Applications and Radio Management

Devices are software proxies for hardware components
Please submit comments and/or questions about this session to:

sca.mentoring@crc.gc.ca

Thank You!