DOI: http://dx.doi.org/10.26483/ijarcs.v8i9.5064

ISSN No. 0976-5697

Volume 8, No. 9, November-December 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

ENHANCEMENT OF BACKWARD COMPATIBILITY IN SOFTWARE COMMUNICATION ARCHITECTURE (SCA) STANDARDS AMONG SCA COMPLAINT SOFTWARE DEFINED RADIOS (SDR)

N Thinakaran
Research Scholar
Dept of ECE, PMU
Thanjavur, India

Dr D Kumar
Head Nano, and Professor
Dept of ECE, PMU
Thanjavur, India

Dr P Saikrishnan
Associate Professor
Dept of Mathematics, NIT
Trichy, India

Dr N Vetrivelan
Principal and Professor
Dept of MCA, Dhanalakhsmi Group of Institutions
Perambalur, India

Dr C Vimala
Head of Mathematics and Associate Professor (SG),
Dept of Mathematics, PMU
Thanjavur, India

Abstract: Since World War-II, world is developing newer communication radio continuously by spending efforts, money and assets. After digital communication systems are invented, the development is rapidly progressing at every country. With globalization in mind, country-to-country starts interacting with each other and begins to share their radios too for operations. In military-to-military contact, there exists a problem of incompatibility of radios between friendly countries. With the Software inventions and mathematical processor improvements, Software Defined Radio (SDR) started appearing in both military and commercial world. We explore the SDR and suggest some Universal standards in Software Communication Architecture (SCA) which can be followed for mitigating the interoperability problem. With SCA-compliant-SDR, the radio can be reconfigured with high flexibility for multi-band-multi-mode capability with maximum portability.

Keywords: Components, JTRS, SDR, SCA, UML and Waveform.

1. INTRODUCTION

All The term Software Radio was coined by Joe Mitola in 1991, for referring the class of reprogrammable or reconfigurable radio [1]. SDR is a radio, in which some or all of the physical layer functions in OSI (Open System Interconnection) network model, are software defined. It means that different waveforms can be supported by modifying the software or firmware, but not changing the hardware. Waveform here means a signal with specific values for all the parameter such as carrier frequency, data rate, modulation, coding etc., Due to advancements in waveform signal processing and hardware processing capacity, radio communication systems were becoming more of software processing and controlling. Each radio is becoming more of mission specific and is not able to communicate with joint forces in military tactical operation or disaster relief. With this as compulsion, Unites States, Department of Defence (DoD) initiated a program in 1990 called Joint Tactical Radio System (JTRS). As a result a prototype named SPEAKeasy was created in various phases [2]. Other programs such as PMCS (Programmable Modular Communication System), WITS(Wireless Information Transfer System) by Motorola, SpectrumWare at MIT by DARPA, CHARIOT(Changeable Advanced Radio for Inter-Operable Telecommunications) at Virginia Tech as part of DARPA's GloMo programs were developed to make a unified Software Communication Architecture. Thus in year 2000, SCA was created by SDR Forum comprising US DoD and civil telecom companies and SCA version 1.0 was released [2]. By the end of year 2001, the SCA framework for development of SDR version 2.2 was released [3]. Over a period of five years of rigorous discussion by SDR forum among US DoD, various telecom manufacturing companies and academicians around the world arrived at a major revision of SCA version 2.2.2 in the year 2006. This version has enhanced the interoperability of communication systems by leveraging the benefits of technology advances in commercial standards for military applications [4]. Michael L Dickens et al.,[5] has designed and implemented a Portable Software Radio in the year 2008. In the next six years of hard work the JPO has released SCA version 4.0 dated 28 Feb 2012[6], by enhancing the SDR in deployment, management, interconnection, and intercommunication of software components in embedded, distributed-computing communication systems. Now a days the Joint Tactical Networking Centre(JTNC, part of US DoD) is managing the SCA. This revision supports lightweight, disadvantaged platforms like handheld radios. But the backward compatibility with products compliant with SCA ver 2.2.2

waveforms was missing. This was not fully addressed in SCA version 4.0 which was released in 28 Feb 2012. We would like to address these backward compatibility problems in SDRs.

Our scope in SDR is limited to the framework of Software Communication Architecture (SCA) which is prevailing in the world and imbibed by SDR Forum, in collaboration with IEEE P1900.1 Working Group. Initially the US DoD has introduced the SCA framework through SDR forum, to resolve the interoperability of radio among military, which is now re-coined as Wireless Innovation Forum (WInnF). We have recommended the backward compatibility issue between SCA 2.2.2 waveforms and devices to run on current SCA versions 4.X. In the Section 2, we have briefly introduced the concepts of SCA alongwith interoperability problem of SDR. In the Section 3, we have explored the likely solutions to SCA standards in the form of modifications in *Base* Component, *Device* Component, *Application Manager* Component, *Application Factory* Component and *Device Manager* Component..

2. CONCEPTS OF SCA

A. Interoperability Problem of SDR in existing SCA Standards

Due to advancements in waveform signal processing and hardware processing capacity, radio communication systems are becoming more of software processing and controlling. Each radio is becoming more of mission specific and is not able to communicate with joint forces in military tactical operation or disaster relief. Hence there was a need for creating an unified communication architecture called Software Communication Architecture (SCA). Dept of Defence (DoD) in USA initiated a program in mid 1990s called Joint Tactical Radio System (JTRS) Program Office(JPO) to pursue the development of future communication systems, with benefits of advance technology. This type of radio was to enhance interoperability, reduce development & deployment cost, upgradeable (modular), scalable, backwards-compatible and reconfigurable. This software programmable radio will also enhance real-time information exchange among friendly forces such as ground troops to combat fighter aircraft.. This will have combat edge among all fighting elements. Such radio is called Software Defined Radio (SDR) which can accommodate multi-service and multi-national capabilities. Such radio is easily upgradeable with latest versions through the standard Application Program Interface (API). Hence SCA was created. Till 2015 SCA version 4.1 has been released for the public to adhere while buying / developing SCA compliant SDR. After thirty years of Evolution, SDR is now a dominant industry standard in radio domain, from military-tactical radios to cellular handsets.

JTRS envisions a radio that will support Operating frequencies from 2 MHz to 2 GHz, be reconfigurable through waveform software, support voice, video and data applications, be scalar in both software & hardware, leverage COTS components for affordability and be interoprable with different waveform, with legacy equipment, and with radios designed for different domains. JTRS program covers five unique domains such as airborne, fixed / maritime, vehicular, dismounted, and hand-held. JTRS is designing its SCA to meet the goals such as "function as Multiband-Multimode-Radio", "be interoperable with all domains", "be compatible with legacy system", "support insertion of new technologies", "support advent networking features" and "use primarily COTS components".

B. Core Framework (CF)

Role of SCA is to provide common infrastructure for managing the software and hardware elements present in a system and ensuring that their requirements and capabilities are commensurate. SCA accomplishes this function by defining a set of interfaces that isolate the system applications from the underlying hardware. This set of interfaces is referred to as the Core Framework (CF) of SCA[6]. In a distributed architecture, functionalities such as deployment, management, interconnection and intercommunication can be achieved by this core framework interfaces and services. Portability can be achieved among SCA compliant radio / platform using CF interfaces with some limitations.

CF Interface can be grouped into three major types viz., *Base Application Interfaces, Framework Control Interfaces and file service Interfaces.* Base Application Interfaces provide a common set of interfaces for developing software application components and exchanging information between them. They are *Port, Life cycle, Testable Object, Property set, Port Supplier, Resource factory and Resource, Controllable component.* Framework Control Interfaces provide the means for control and management of hardware assets, applications and domain(system). They are further sub-divided as *Device Interfaces, Device management Interfaces, Domain Management Interfaces.* File access services in a radio can be achieved using File Service Interfaces in Core Framework of SCA standards. There are three services viz., *File* service, *File system* interface and *File Manager* interface. The CF interface is depicted in the form of Unified Modeling Language (UML) at figure.1.

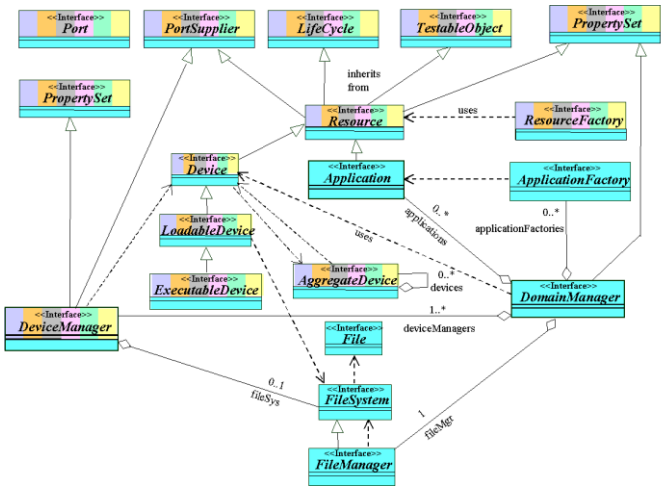

Fig 1 . CF Interfaces Top-Level View.

C. Application Architecture

The application architecture of SCA deals with Application-Layer-Software partitions which gives details about how waveform might be implemented using the SCA. Applications such as modem, Link, Network, security & I/O components perform user level functions. These components require CF interface & services through MAC API,

LLC/Network API and I/O API. These applications have direct OS access limited to SCA POSIX (Portable Operating System Interface) Profile. The application architecture of SCA is given below in figure.2.

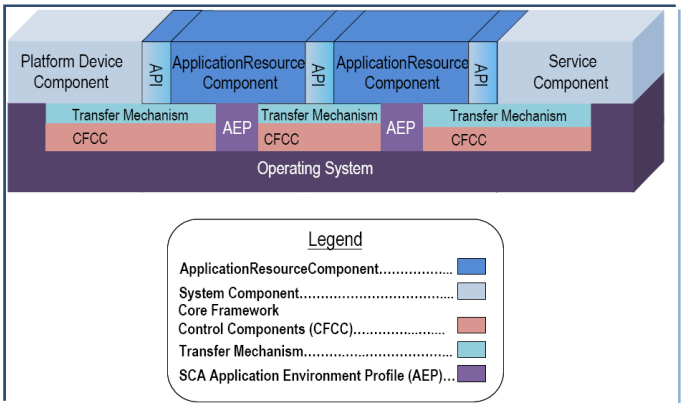

Fig. 2. Application Use of Operating Environment.

D. SCA Software Architecture

The SCA Software Structure can be broadly divided as Application Layer and Infrastructure Layer. The software architecture resides over a hardware. The details of SCA software architecture are shown in figure.3 given below.

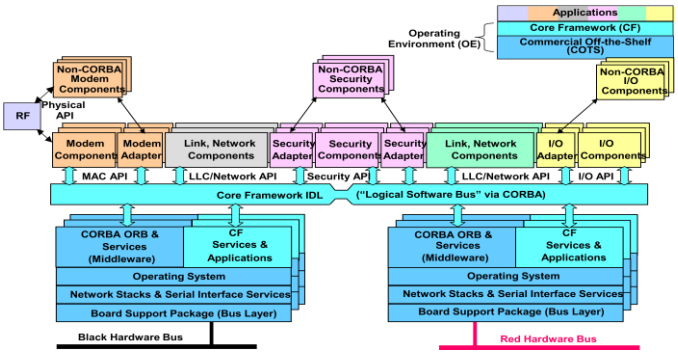

Fig.3. SCA Software Structure.

The interoperability problem between a radio which follows SCA 4.X standards and another radio which follows SCA 2.2.2 version can be solved by making modification in SCA 4.X Base component. For this, the SCA 2.2.2 *Resource* interface and SCA 4.X *BaseComponent* can be compared and analysed as shown in figure 4 and 5 respectively[3][6]. While the software development is developing the migratory version from SCA 2.2.2 to SCA 4.1, the team has to envisage the changes in interfaces, requirements and designs. If a comparison is made between *BaseComponet* of SCA 4.1 and *Resource* of SCA 2.2.2 standards, it will be equivalent. *Resource* interface is a set of interfaces needed to initialise, configure, control, and tear-down a component as given in fig 4. It will have a start function and stop function for controlling the error which may occur. The Resource interfaces are depicted with CORBA interface in Unified Modelling Language in the fig 4 below..

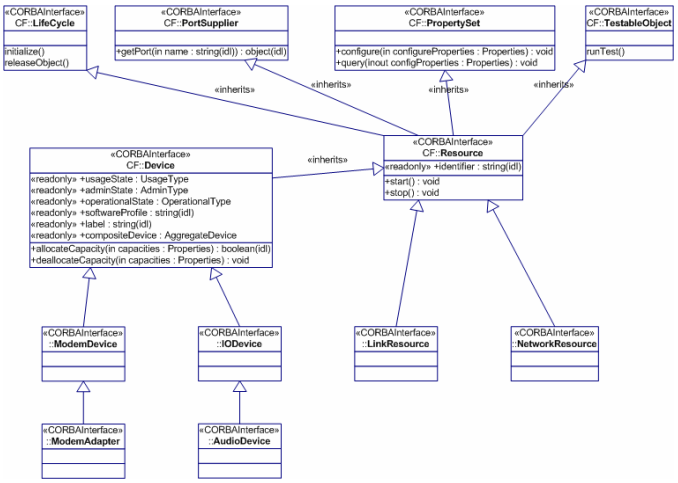

Fig. 4. SCA 2.2.2 Resource relationships in UML.

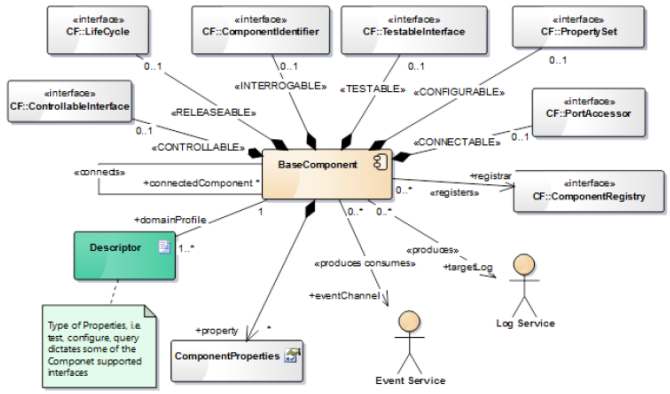

Fig. 5. SCA 4.X Base Component

3. ENHANCEMENT OF SCA STANDARDS[7][8]

SCA standards are to be enhanced to cater for backward compatibility by modifying the *Base* Component, *Device* Component, *Application Manager* Component, *Application Factory* Component and *Device Manager* Component as shown in the following paras.

E. ***Modification in Base Component***

By analysing the *Resource* interfaces of SCA 2.2.2, following changes in *Resource*, *LifeCycle*, *PropertySet*, *PortSupplier*, and *TestableObject* are suggested for mitigating the backward compatibility problem of SDRs (figure 6).

Since the componentization is the biggest modification required for backward compatibility, it is recommended to create new interface called *ComponentIdentifier* and *ControllableInterface*, in place of Resource. A new *started* attribute is recommended for introduction within *ControllableInterface* interface. Accordingly the *ControllableInterface* is to be inserted in place of *Resource* interface.

SCA 2.2.2

```
interface Resource : LifeCycle
 TestableObject, PropertySet, PortSupplier
                    {
          exception StartError {
             CF::ErrorNumberType
       errorNumber, String msg;
                    };
          exception StopError {
             CF::ErrorNumberType
       errorNumber, String msg;
                    };
    readonly attribute string identifier;
             void start()
                    raises
      (CF::Resource::StartError);
             void stop ()
                    raises
      (CF::Resource::StopError);
             };
```

SCA 4.X

```
interface ControllableInterface {
        exception StartError {
    CF::ErrorNumber Type errorNumber,
          String msg;
                };
        exception StopError {
    CF::ErrorNumber Type errorNumber,
          string msg;
                };
   readonly attribute Boolean started;
          void start()
                raises
  (CF::ControllableInterface::StartError)
          void stop()
                raises
  (CF::ControllableInterface::StopError)
          };
   interface ComponentIdentifier {
    readonly attribute string identifier;
          };
```

SCA 2.2.2

```
interface PortSupplier{
  exception UnknownPort {
          };
  Object getPort (in string name)
          Raises
(CF::PortSupplier::UnknownPort);
};
interface Port {
exception InvalidPort {
   unsigned short errorCode;
  string msg;
          };
exception OccupiedPort{
  };
void connectPort (
   in Object connection,
   in string connectionidd)
   raises
(CF::Port::InvalidPort,
CF::Port::OccupiedPort);

  void disconnectPort (
   in string connectionid)
   raises (CF::Port::InvalidPort);
}
```

Fig. 6. Comparison of Migration of Resource Interface.

LifeCycle and *PropertySet* interfaces are retained identical. *PortSupplier* interfaces are compared and analysed as given below at figure 7. Following are the changes suggested to mitigate the interoperability problem. The functionality of the *Port* and *PortSupplier* interfaces is recommended for merging. Same can be unified as with the new interface called the *PortAccessor* interface. Exceptions in *OccupiedPort* is suggested for removal. This functionality can be combined with *InvalidPort* Exceptions. The new variable called *ConnectionErrorType* can contain this exception for backward compatibility. With this all the port operations can be called on a single execution for simultaneous multiple communications of SDR radio.

```
SCA 4.X
interface PortAccessor {
   struct ConnectionidType {
            string connectionid;
            string portName;
            };
  typedef sequence <ConnectionidType>
Disconnections;
 struct ConnectionType {
            ConnectionType portConnectionid;
            Object portReference;
            };
  typedef sequence <ConnectionType>
Connections;
  struct ConnectionErrorType {
            ConnectionType
portConnectionid;
 unsigned short errorCode;  };
  exception InvalidPort {
ConnectionErrorType invalidConnections;   };
void connectUsesPorts(
in CF::PortAccessor::Connections
portConnections)
  raises(CF::PortAccessor::InvalidPort);
void disconnectPorts(
  in CF::PortAccessor::Disconnections
portDisconnections)
 raises(CF::PortAccessor::InvalidPort);
 void getProvidesPorts(
 inout CF::PortAccessor::Connections
portConnections)
raises(CF::PortAccessor::InvalidPort);
 };
```

Figure. 7. Port Interfaces Comparison

In *TestableObject*, replace *TestableObject* interface with *TestableInterface* as given in figure 8 comparison.

```
SCA 2.2.2
interface TestableObject {
    exception UnknownTest {
            };
   void runTest (
            in unsigned long testid,
            inout CF::Properties
            testValues)
            raises
(CF::TestableObject::UnknownTest,
CF::UnknownProperties);
 };
```

```
SCA 4.X
interface Testableinterface {
  exception UnknownTest {
            };
   void runTest (
            in unsigned long testid,
            inout CF::Properties
            testValues)
            raises
(CF::TestableInterface::UnknownTest,
CF::UnknownProperties);
 };
```

Fig. 8. Test Interface Comparison.

Within both SCA 2.2.2 and SCA 4.X support the same core capabilities of Configuration management, Operations Management, Life Cycle Support, Connectivity management and Test management.

F. ***Modification in Device Component***

In order to support the Device Component functionality with all types of radio, all version will have to possess Capacity, Configuration, Operations, connectivity and test management with Life Cycle supports. All the devices such as *Loaded* and *Executable* can be replaced with respective components such as *DeviceComponent*, *LoadableDeviceComponent* and *ExecuatbleDeviceComponent.* With this modification, the *Device* Interface and its Device *scoped attributes and exceptions* can be eliminated in the newer versions of SCA. In the exception classes, the *Invalidstate* can be moved to the *ComponentType*. *SoftwareProfile* in the newer versions can be moved to *ComponentType*. Refactor *Device* Interface into three new interfaces *AdministrableInterface*, *CapacityManagement* and *DeviceAttributes*. Accordingly the names and data types has to be replaced. Device Component SCA standards is recommended to have UML as shown in figure 9.

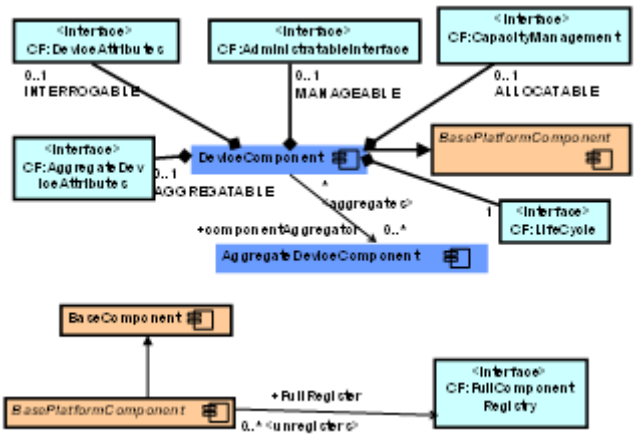

Fig. 9. Proposed Device Component Model.

G. ***Modification in Application Manager Component***

In the older SCA versions, applications are realized through the *Resource* Interface. After the removal of the same in the new SCA versions, it is suggested to rename all the application components into the *ApplicationFactory*

Components. Appropriately the naming conventions can be modified to suit all the applications envisaged in backward combatibility. The attributes pertaining to each application has also to be relocated including *Profile*, *componentDevices* and the *process identities*.

H. ***Modification in Application Factory Component***

Application Factory undertakes the application deployment, Component connection, initiation and configuration of applications. For backward compatibility, the naming services in the form of identifier and its attributor are to be removed. Then create the componentization of *ApplicationFactory*. The application has to be instantiated into *ApplicationManager* through its proxy interface. For this appropriate entry is to be made in the *ComponentRegistry*. In addition, the "create Operation" has to be modified to hold the *ComponentType*, *deploymentDependencies* and *executionAffinityAssignments* parameters. Then the naming service of *ApplicationFactory's* association has to have an entry in the component registry. In overall the *componentRegistry* will act as a central repository for registration of each deployed components in the radio. All these components are to be stored with appropriate information inside the *ApplicationFactoryComponent*. These are periodically updated and executed from the radio platform with a passing reference through a *componentRegistry*. Update any use of the *ResourceFactory, ExecutableDevice, LoadableDEvice, Device and Resource* interface to refer to a *ComponentFactoryCcomponent, ExecutableDeviceComponent, LoadableDeviceComponent, DeviceComponent and ManageableApplicationComponent* references respectively. As and when new events are added, these are to be notified through an unique id for extending *ApplicationFactoryComponent* to the *DomainManagementObject* too. With the above mentioned modifications, all the application can have backward compatibility.

I. ***Modification in Device Manager Component***

Device Managers assist us in new Device and service deployment and will manage all the nodes in that communication network. First and foremost, remove the *DeviceManager* interface in lieu of a new interface, which can be user-defined interface. For inheriting non-CORBA application interface, few modifications are to be suggested. As suggested similar to other above components, for Device Manager too, a new Component Type has to be created. Then move the *deviceConfigurationProfiles* inside the newly created component. Since each device has its file of its type, it is prudent to move the *filesys* and *identity* of each device to the Component Container. Each device has to be shutdown and initialized as and when it comes to network or if its demanded for interface. Hence the shutdown operation has to be controlled by the *ReleasableManager* interface. The attribute pertaining to this operation is to be made available inside the Component Container Registry.In order to pave entry for Componetisation, all the Devices and Services which are likely to be encountered in the Software Defined Radio network is be removed from registration and un-registration kind of operations. As soon as the devices and services are componetised, next is the logic, behind which these devices and services are to be operated, require migration into *componentRegistry*. Accordingly all the information about the components is to be stored inside the Component Container, including the *DeployementAttributes*. Since the SCA version of 2.2.2 has *getComponentImplementationId*, this needs to be removed for the componentisation of all interfaces within the *ComponentType* Container. All the data are to be ensured for its availability in the container.

J. ***Modification in Domain Manager Component***

Among the six component modification, Domain manager is one among them, which needs few changes. Domain Manager undertakes functions such as Application installation and management, Component registration and unregistration, and management of application factories and all device managers. Like the other modifications, this domain manger also needs to be componetised. The *DomainManager* interface is to be divided into two parts viz., *DomainInstallation* and *EventChannelRegistry*. Similar to the above modifications, the exceptions envisaged during registration and un-registration is to be readjusted into appropriate Component Registry. For identification, the *ComponentIdentifier* interface has to be used. In addition, the installation and un-installations exceptions are to be moved to the *DomainInstallation* interface.

By carrying out the above modifications SCA 4.X standard SDR will have backward compatibility with SCA 2.2.2 standard SDR products. Thus performance of SDR can be enhanced.

4. CONCLUSION

Backward compatibility Problem is existing in SCA standards 4.0. Backward compatibility issues are related to the *Base* Component, *Device* Component, *Application Manager* Component, *Application Factory* Component and *Device Manager* Component. This has been addressed by us in the form of modification in the standards. By analysing the *Resource* interfaces of SCA 2.2.2, following changes in *Resource, LifeCycle, PropertySet, PortSupplier,* and *TestableObject* are suggested for mitigating the backward compatibility problem of SDRs (figures 6 to 8). The other modifications are suggested in *Device* Component, *Application Manager* Component, *Application Factory* Component and *Device Manager* Component at para 3. By carrying out the above modifications SCA 4.X standard SDR will have backward compatibility with SCA 2.2.2 standard SDR products. Thus performance of SDR can be enhanced.

5. ACKNOWLEDGMENT

We thank the Wireless Innovation Forum (WInnF) for their great service in standardizing the SCA. Their documents have helped us to work and enhance the SDR performance further.

REFERENCES

[1] Joseph Mitola, III, "Software Radio Architechture: Object Oriented Approaches to Wireless Systems Engineering", John Wiley and Sons, 2000.

[2] Jeffrey Reed, 'Software Radio: A Modern Approach to Radio Engineering', Prentice Hall Communications Engineering and

Emerging Technologies Series, New Jersey 07458, ISBN 0-13-081158-0, May 1991.

[3] "Software Communication Architecture version MSRC-5000 SRD 2.2" by JTRS Joint Program office, San Diego dated 19 Dec 2001.

[4] "Software Communication Architecture Specification Version 2.2.2" dated 15 May 2006, by Joint Program Office(JPO), San Diego, for the Joint Tactical Radio System(JTRS).

[5] Michael L Dickens et al., "Design and Implementation of a Portable Software Radio" Norte Dame, IN, 10 Jun 2008.

[6] "Software Communication Architecture Specification Version 4.0" dated 28 Feb 2012, and its Appendices "A" to "F" version 4.0.1 dated 01 Oct 2012" by Joint Program Office(JPO), San Diego, for the Joint Tactical Radio System(JTRS).

[7] Program Executive Office Command Control Communication Tactical 's Notes , "JTNC SCA 4.1 New Features in JTNC Standards", Webinar Series lecture dated 18 Feb 2015.

[8] LeePucker, "SDR 4.1 Draft Specification Release" Webinar Series lecture by Wireless Innovation Forum dated 18th Feb 2015