USE OF DEVICE, MANAGING DEVICE-RELATED INTERFACES IN SCAV2.2

Yusuke KANAHASHI; Isao TESHIMA; Satoru NAKAMURA;
Mitsuyuki GOAMI; Takuzo FUJII
Communication Systems Development Group, Hitachi Kokusai Electric Inc.,
Tokyo, Japan; email:kanahashi.yusuke@h-kokusai.com

ABSTRACT

In "Development of Software Radio Prototype" [1] presented during the 2002 SDR Forum Technical Conference, we reported on a prototype SDR which provides various types of analogue/digital modulation, two full duplex channels, and a frequency range of 2-500MHz, evaluating hardware design approaches for multi-band, multi-mode SDR. Succeeding this, in "Software Architecture and Waveform Applications of Software Defined Radio Based on SCA v2.2"[2][3] presented during the 2003 same conference, we reported on an improved prototype with upgraded hardware and adoption of the SCA v2.2, examining our software architecture approach and waveform applications. There, we commented "Having some doubts about the necessity of the interface definition for devices which generally do not provide software portability, we reserve application of these interfaces defined in SCA v2.2 until it becomes clear".

In order to solve this issue, SDR Forum is studying HAL (Hardware Abstraction Layer)[4]. We reconsidered necessity of: By adaptively managing device-related interfaces, the devices can be used even when internal or external devices have no software portability due to differences in compilers, or in memory maps. In order to realize this consideration in the next SDR prototype, study was carried out. This paper reports this study result.

The device-related interfaces defined in SCAv2.2 include: *Device/LoadableDevice/Executable Device* interfaces that are needed when devices are used, and *aggregateDevice* interface that controls relations between above interfaces. And *DeviceManager/DomainManager* interfaces manage above all interfaces. The devices as the objects of these interfaces are CPU, DSP, FPGA, and firmware connected to SDR. However, for the sequences of procedure for using devices through these interfaces, only a part is shown in SCAv2.2.

Our study results include:

1) For *Device* / *LoadableDevice / ExecutableDevice /aggregateDevice* interfaces:
- The cases where these interfaces are applied.
- The relation between these interfaces and physical interfaces including; not defined in SCAv2.2.
- Necessary considerations when these interfaces are applied.

2) For Management of the above interfaces by *Device Manager/DomainManager* interfaces.
- Proposed description for management to be added to JTRS SCAv2.2.

Our study results show that: Device-related interfaces defined in SCAv2.2 become effective for use of devices, by managing with *DeviceManager/DomainManager* interfaces according to additional procedures we propose.

1. INTRODUCTION

We have developed hardware platform that provides high flexibility and adaptability in SDR. And the prototype presented during the 2002 SDR Forum Technical Conference was developed to adopt the software architecture of SCA v2.2, examining flexibility of *Waveform Applications* for the software execution environment and the expandability of functions. Since it was unclear for us how *Device*-related Interface for hardware contributes to the portability of the *Waveform Applications* at that time, we reserved application of these interfaces. Later, we recognized that: "assuring portability of the *Waveform Applications* for hardware" and "application procedure and management of hardware" should be considered as independent issue. That is; although SCA v2.2 made abstraction of how to use or manage logical hardware and software, in order to contribute to the portability of *Waveform Applications* for hardware, it is necessary to make abstraction of the executive environment for *Waveform Applications.* It is concluded with the both abstractions the portability is available.

2. STATUS OF *DEVICE/LOADABLE DEVICE/EXECUTABLE DEVICE* IN SCA V2.2

SCAv2.2 defines *Device* related Interfaces for abstracted functions for hardware in the core framework where many functions independent of the utilization gathers (Fig.1). These hardware related Interfaces, that are software, are called *Logical Device* and include three kinds classified by its operation as described below:

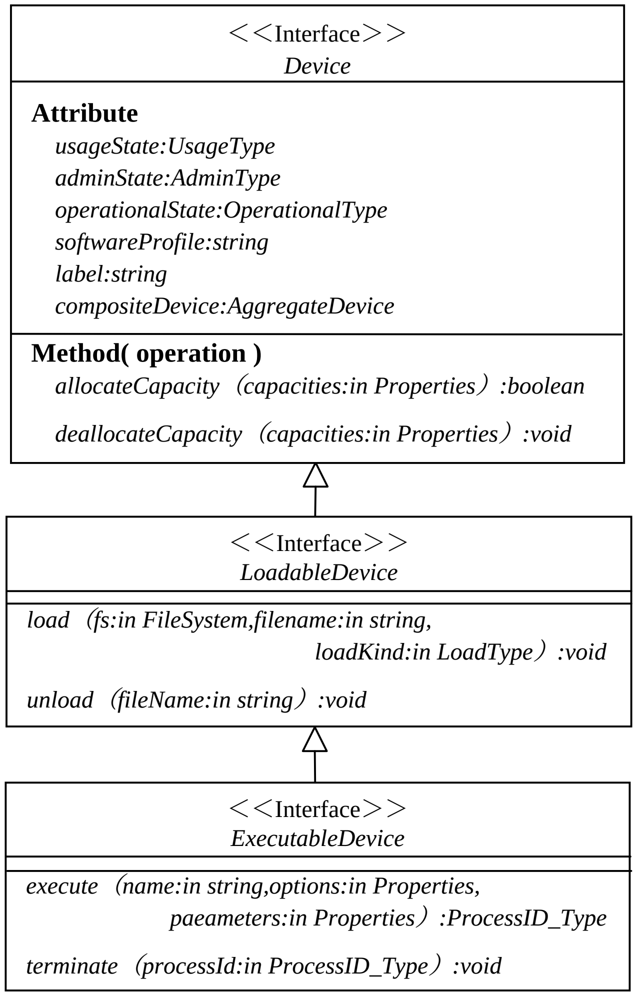
a*) Device* —*allocateCapacity*:
Judges availability of resource of hardware function.
deallocateCapacity: Returns the resource.

b) *LoadableDevice* —*load/unload*:
Load/unload of information or software on the hardware.

c) *ExecutableDevice* —*execute/terminate*:
Execute/terminate of the software loaded on the hardware.

Device has the following attributes:

- *UsageState*/*adminState*/*operationalState*: Returns the status.
- *SoftwareProfile*: Returns the XML software description for device.
- *Label*: device's label.
- *CompositeDevice*: Returns the *ObjectReference* of *aggregateDevice*



<<Interface>>
Device
Attribute
usageState:UsageType
adminState:AdminType
operationalState:OperationalType
softwareProfile:string
label:string
compositeDevice:AggregateDevice
Method(operation)
allocateCapacity（capacities:in Properties）:boolean
deallocateCapacity（capacities:in Properties）:void
<<Interface>>
LoadableDevice
load（fs:in FileSystem,filename:in string, loadKind:in LoadType）:void
unload（fileName:in string）:void
<<Interface>>
ExecutableDevice
execute（name:in string,options:in Properties, paeameters:in Properties）:ProcessID_Type
terminate（processId:in ProcessID_Type）:void

Figure 1. Abstracted interfaces of hardware

The *Device* related Interfaces are CORBA interfaces that command hardware according to the above classification. These interfaces are involved in *DeviceManager* and *Domain Manager*, and the existence of these interface are registered to the *DeviceManager* and *Domain Manager*.

3. STUDY RESULTS ON USE OF *DEVICE*, BY MANAGING *DEVICE*-RELATED INTERFACES IN SCAV2.2

3.1. Application to Hardware of *Device, Loadable Device, Executable Device, Aggregate Device*

In order to meet the timing issues in such as sampling, data processing of SDR, DSP and FPGA are widely used, though CORBA Bus installation to these devices is difficult. In order to cope with this issue, SCA v2.2 provides interfaces so called *adopter* that works for connection between the Software Bus and non-CORBA Bus of DSP and FPGA.

The following are our study results on application to the hardware:

CPU inherits *ExecutableDevic* behaving software execution.In general, *GPP Device*, which is the interface of CPU (GPP), inherits *ExecutableDevice*, and will take a sequence shown in Fig. 2 for the software allocated to CPU. The *allocate Capacity* Judges availability of CPU for the software to be allocated, from factors such as, CPU load factor, use factor of the operation memory, and number of software to be executed on the CPU. When it is available, software is permitted to use the CPU and executes being deployed on operation memory of the CPU.

Although DSP and FPGA operate software execution similar to the CPU, they inherit *LoadableDevice.* DSP and FPGA behave *load* of software and configuration data similar to CPU, but do not behave execute/terminate of software without OS. Usually DSP and FPGA do not provide Linux-like OS. However, they execute software that is read out from memory where executing software and configuration data are clearly allocated. Therefore, it is convenient to use DSP and FPGA on operation of *load method* for allocation and reading out of software to the memory.

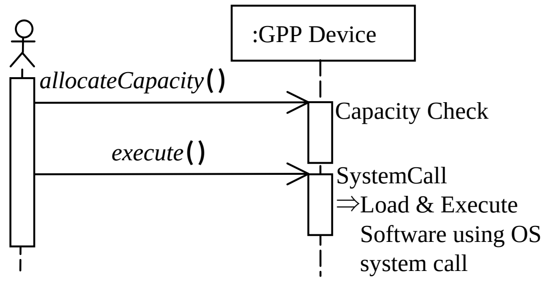


Figure 2. Sequence for software allocation on GPP

Table 1. Application Method of *Device*-related Interfaces

Hardware Device	Interface	*Method*	Operation to be Applied	Application Method of Operation
CPU (GPP)	*Executable Device*	*allocate Capacity*	・Judges margin of CPU load, CPU occupancy time-rate, etc. ・Increments number of Executed Software, etc.	Judges availability of the device by: ・Obtaining data by OS System Call, and ・Managing number of executed Software in GPP Device Class.
		deallocate Capacity	Decrements number of the Executed Software, etc.	
		load/ execute	Load and Execution of the Software.	Executed by OS System Call that depends on the Load Types.
		unload/ terminate	Termination of the Software	Terminates by using Process ID of executed Software used by OS System Call .
DSP	*Loadable Device*	*allocate Capacity*	・Judges margin of available DSP Resource, Memory size, etc. ・Increments number of DSP, renews memory size, etc.	Judges availability of the device by managing DSP resource and memory size in DSP Device Interface.
		deallocate Capacity	・Decrements number of DSP, renews memory size, etc.	
		load	・Writes in Software to memory. ・Interruption of reboot to DSP.	・Memory write in / erase out, by using the device driver. (The directions to a device driver depends on arg *LoadType*.) ・Reset/reboot interruption to DSP by using the device driver.
		unload	・Interruption of reset to DSP. ・Erase Software from memory.	
FPGA	*Loadable Device*	*allocate Capacity*	・Judges margin of available FPGA, memory size, etc. ・Increment number of FPGA, etc.	Judges availability of the device by managing FPGA resource in FPGA Device Interface.
		deallcate Capacity	・Decrements number of FPGA, etc.	
		load	・Writes in configuration data to memory to be loaded. ・Interruption of reboot to FPGA.	・Memory write in / erase out, by using the device driver (The directions to a device driver depend on arg *LoadType*.) ・Reset/reboot interruption to FPGA by using the device driver.
		unload	Erases configuration data from memory. ・Interruption of reset to FPGA.	
LAN /Serial	*Device*	*allocate Capacity/ deallcate Capacity*	・Judges available number of resources (e.g. LAN/Serial port). ・Increments/Decrements number of resources, etc.	Judges availability of I/O device, by managing number of Ether/Serial resources, and memory size.
Firm Ware	*Loadable Device* or *Executable Device*	*allocate Capacity*	Increments number of resources (e.g. number of RF Channels) in use, etc	Manages number of Firmware resources (e.g. RF Channel etc.).
		deallcate Capacity	Decrements number of resources, etc.	
		load	Transmits and commands configuration information or software.	Transmits load/unload command according to the interface specifications, by using device drivers or OS services (May transmits execute command simultaneously).
		unload	Transmits deletion command.	
		execute	Transmits execute command.	Transmits execute/terminate command according to the interface specifications by using device drivers or OS services. (May transmits load command simultaneously).
		terminate	Transmits terminate command.	

Each interface of CPU, DSP, FPGA is allocated as shown in Fig. 3 so that each can be commanded through CORBA.

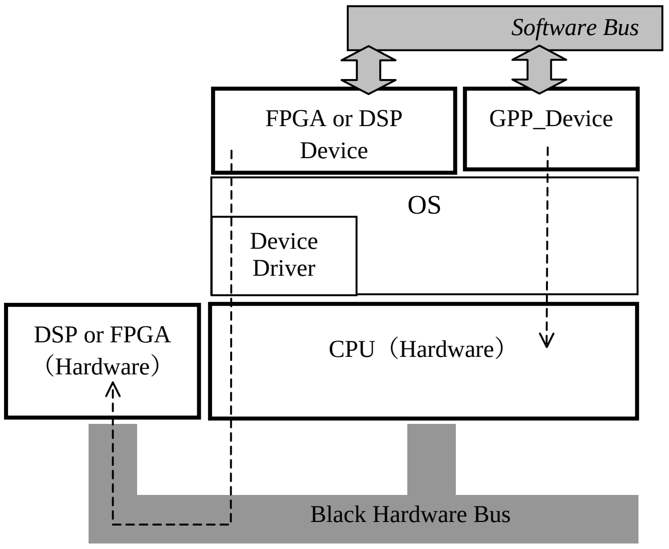


Figure 3. Distribution example

Command from each interface to hardware is included in Table 1.

According to the application methods indicate in Table 1, it is possible to use functions uniformly as CORBA interfaces of *Device/ loadableDevice/ExecutableDevice* regardless of control object, DSP/FPGA or external interface/firmware.

Linux-like OS can make *socket* with system call as an application of network device (e.g. Ether).

Since restriction of use of hardware device by *allocate Capacity* depends on *capacity*, interface or method, which treats approval of software execution, need to be added as described in the following Section.

Aggregatedevice is the interface that commands the combination of devices (e.g. combination of DSP and FPGA) when such combination is used.

3.2. Necessary consideration for application of *Device, Loadable Device, Executable Device, Aggregate Device*

Necessary consideration for application of *Device*-related interface is shown in Table 2. In particular consideration for Execution ability is important.

SCA v2.2 does not explain protection method for combination of unauthorized or unidentified software and hardware. In order to cope with this issue, we defined following to the framework: interface for acceptance and detailed behavior, for the software execution on device.

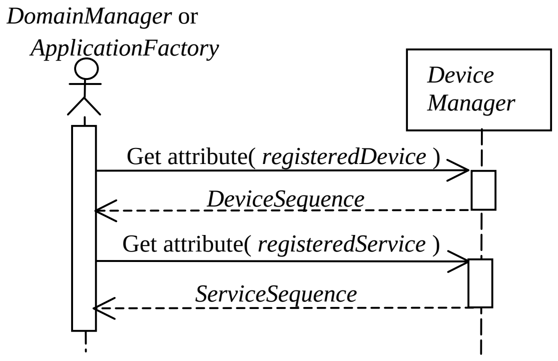
Table2. Necessary considerations for application of *Device*-related interface

Consideration item	Necessary consideration
Kind of applied interface	･Availability of major functions of devices. ･Appropriate operation sequence for devices. ･Exchangeability of device software
Allocated position	Availability for connection of *device Interface* to both software bus and hardware.
Execution ability	･Judgment of applicable software to device. ･Command to hardware on: loading, execution/stop, and deletion of software.
Expandability	Ability of addition/removal of *Device* interface for hardware expansion/reduction

3.3 Management by *Device* Manager and use for *Device*

Based upon the partial descriptions on this theme in SCA v2.2, we understand as follows: In order to use *Device* related interface, *DeviceManager* is provided. It is a software interface that indicates available software resource that inherits the *Device* interface. *DeviceManager* provides *method* for registration/removal of *ObjectReference* of *Device* and *Service*. *DeviceManager* adds the registered *ObjectReference* to the attribute value. As shown in Fig. 4, the software can obtain *Object Reference* of available *Device* and *Service* from its attribute value identified by accessing to the attribute of *DeviceManager*.

The sequence of registration by *Device*, using *DeviceManager*, from start of hardware operation until completion of the registration, is shown in SCA v.2.2.



DomainManager or ApplicationFactory
Device Manager
Get attribute(registeredDevice)
DeviceSequence
Get attribute(registeredService)
ServiceSequence

Figure 4. Use of available Device and Service *ObjectReference*

3.4. Management and use of *Device* by *DomainManager*

As described in SCA v2.2, *DoaminManager* manages the resource in *Domain* as the whole. That is, all the object references of *Device* and *Service*, registered by *Device Manager* as described in section 3.3, are registered in *DomainManager*. *Object References*, such as *Application*, are registered also in *DomainManager*.

For the management and the usage of *Device* that are not described in SCA, we used the following methods:

- For *ObjectReference* of available Device: To obtain based on the attribute of *DomainManager* which includes all available *Device Managers* in a domain.
- For hardware availability and expandability: To perform registration operation to *DomainManager* by allocating one or more *DeviceManager*/*Device* to the additional CPU board and additional hardware.
- For hardware reduction or removal: By the sequence shown in Fig.5.

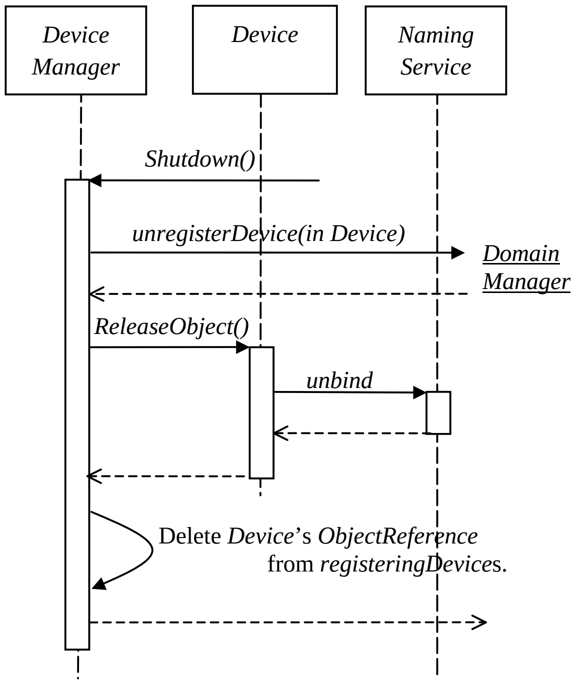

Device Manager
Device
Naming Service
Shutdown()
unregisterDevice(in Device)
Domain Manager
ReleaseObject()
unbind
Delete Device's ObjectReference from registeringDevices.

Figure 5. Sequence for hardware reduction or removal

- Construction of *Application* (*Domain* formation) by *ApplicationFactory*: Information of available *Device* resource for construction of *Application* is obtained from *Object References* managed by *DomainManager*. Based on this information, *ApplicationFactory* accesses to *Device* necessary for construction of *Application* (Fig.6).

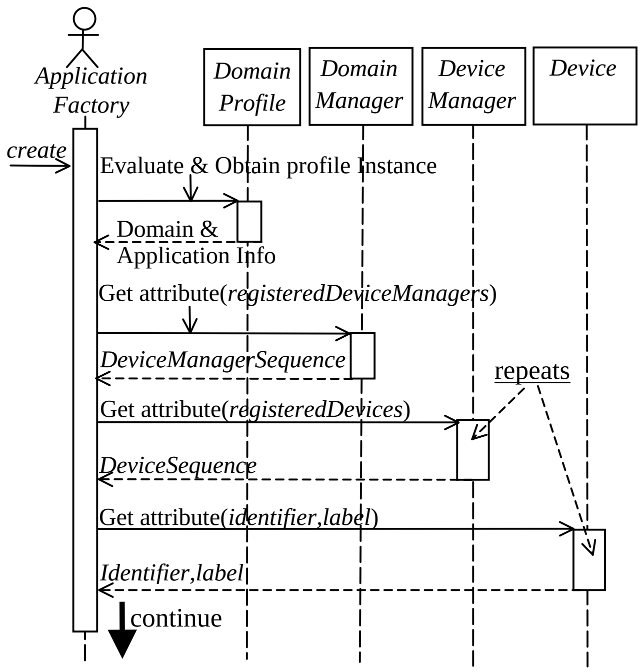

Application Factory
Domain Profile
Domain Manager
Device Manager
Device
create
Evaluate & Obtain profile Instance
Domain & Application Info
Get attribute(registeredDeviceManagers)
DeviceManagerSequence
repeats
Get attribute(registeredDevices)
DeviceSequence
Get attribute(identifier,label)
Identifier,label
continue

Figure6. Uses Device, DeviceManager and DomainManager for DomainCreate

3.5. Arrangement and command for *Device*-related interface, *DeviceManager*, and *DomainManager*

We have coped with this issue that is not described in SCA v2.2, as follows:

Device-related interface, *DeviceManager*, and *Domain Manager* are arranged as shown in Fig. 7.

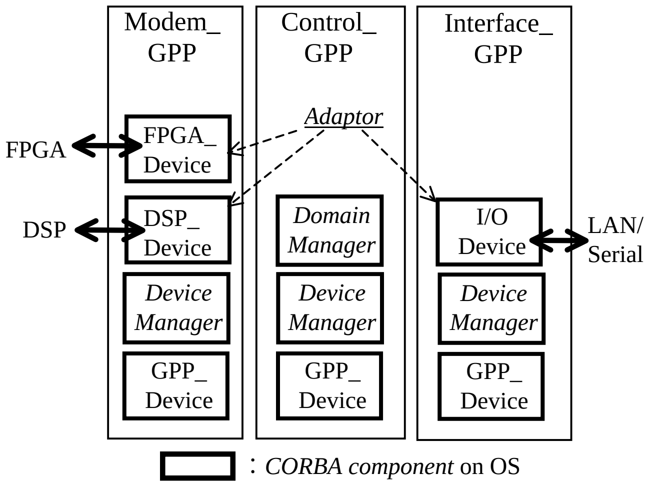


Figure7. *Device*/*DeviceManager*/*DomainManager* deployment

With this arrangement, each interface is called through the software bus, so that management and use of *Device* can be made corresponding to the following various cases:

- Loading and execution of software for the hardware device where *Device* delegates its function,
- Addition or removal of hardware device can be performed freely by operation of a *Device* related interface that delegates its function to the hardware device. This makes system flexible for system expansion or reduction.

3.6. Management procedures to be added to SCAv2.2

Device-related Interfaces defined in SCAv2.2 are abstracted interfaces that can correspond to various physical interfaces or use cases for application to a hardware device. And combining with a *Manager* interface, addition or removal of a hardware device can be performed flexibly. That is, *Device* is sufficiently managed.

However, as described in the previous Sections, the following management procedures should be added to SCAv2.2 and standardized.

1) Software approval sequence (See 3.1):
   The software executed on GPP board should be approved one for each GPP board (CPU) operation.
2) Operation for obtaining the *ObjectReference* of arranged *Device* in *creating* operation of *ApplicationFactory*. (See 3.4):
   (By standardizing the method of obtaining *ObjectReference* of *Device* that arranges software, compatibility of *Application Factory* with the platform can be increased)
3) The arrangement rule of a *Device* interface (See 3.5):
   Arrangement of *Device* interface should be described
   (Depend upon the Arrangement of *Device*, some isolated interfaces may exist which can not be called from software bus.)

4. CONCLUSION

We have carried out our study considering that:
By adaptively managing *Device*-related interfaces, the *Devices* can be used even when internal or external devices have no software portability due to differences in compilers, or in memory maps.

As the results, we have confirmed appropriateness of the above consideration, as described in Section 3. That is:
With the following countermeasures, *Device*-related interface of the framework in SCAv2.2 is sufficiently effective:

- Appropriate consideration for application hardware of *Device*-related interface (Table 1 and 2).
- Appropriate arrangement and direction system of *Device Manager* and *DomainManager*. And management and use of *Device* by *DeviceManager* and *Domain Manager* (Fig.3, and 3.5)
- Appropriate management procedures to be added to SCAv2.2 (3.3, 3.4, and 3.6)

In the above, we proposed management procedures for *Device*-related interface to be standardized (Fig. 5 and 6). We hope that: These standardizations are put into practice and the portability in the field of management procedure progresses. Furthermore, by SCA with HAL, portability of execution environment is further progressed, and obstacle to standardization on execution environment is eliminated.

We will further continue our studies including HAL. However, above study results will be introduced to our recent SDR proto type, aiming at SDR where *Device*-related Interfaces and *Managers* are fully applied.

Our goal is production of SDR that is fully application compatible. SCA should further be improved. We will also continue our study according to that.

5. REFERENCES

[1] I.Teshima, K.Takahashi, Y.Kikuchi, S.Nakamura, and M. Goami, “Development of Software Radio Prototype”, Proceedings of the 2002 Software Defined Radio Technical Conference Volume 1, pp.169-174, November 11-12, 2002.

[2]Y.Kanahashi, I.Teshima, S.Nakamura, M.Goami, and T.Fujii “Software Architecture and Waveform Applications of Software Defined Radio Based on SCAV2.2”, Proceedings of the 2003 Software Defined Radio Technical Conference Volume 2, PST-2, November 18-19, 2003.

[3]Software Communications Architecture (SCA) Specification MSRC-5000SCA, V2.2,17 November 2001

[4] Software Communications Architecture (SCA) Specification JTRS-5000SCA, V3.0, 27 August 2004