

# NATO RTO/IST RTG ON SDR: DEMONSTRATING PORTABILITY AND INTEROPERABILITY OF SCA-BASED WAVEFORMS

Sarvpreet Singh, Marc Adrat, Markus Antweiler

Fraunhofer Institute for Communication, Information Processing & Ergonomics (FKIE),  
Wachtberg, Germany

[sarvpreet.singh@fkie.fraunhofer.de](mailto:sarvpreet.singh@fkie.fraunhofer.de), [marc.adrat@fkie.fraunhofer.de](mailto:marc.adrat@fkie.fraunhofer.de)

## ABSTRACT

Among various features offered by the SDR technology, two important aspects related to it are portability of waveform applications and interoperability between SDR platforms. The NATO RTO/IST Research Task Group on Software Defined Radio (RTG on SDR) is working on these aspects in a *Software Communications Architecture* (SCA) based environment on a mutually agreed test waveform of STANAG 4285. This paper presents the steps taken in order to port the STANAG 4285 waveform from one SCA Operating Environment (OE) to another. This means, moving from one Core Framework (CF), Object Request Broker (ORB) etc to another using the processing element as a General Purpose Processor (GPP). Finally, we present a working demonstration of the STANAG 4285 waveform interoperability between the different OE implementations.

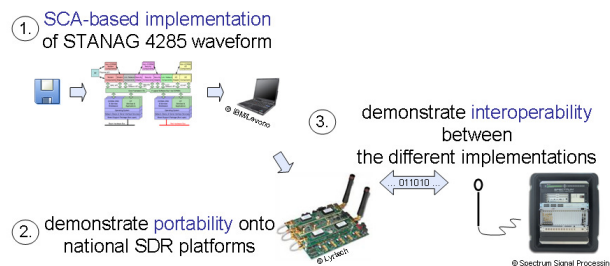
## 1. INTRODUCTION

Established in the year 2007, the NATO RTO/IST Research Task Group [1] on Software Defined Radio is working towards gaining knowledge and experience in the SCA/SDR domain. There are about 10 active participating nations sharing their experiences in the development of SCA based waveforms. As a three folded approach (shown in Fig 1) to acquire such knowledge, the group aims at:

1. Implementation of an SCA based waveform
2. Demonstrating the porting of an SCA based waveform on different SDR platforms
3. Demonstrating the interoperability between different implementations of the same SCA based waveform

The group has jointly agreed on the waveform of STANAG 4285 [2] for its development work. An example implementation of the waveform in ANSI-C code has been provided by *Telefunken Racoms* [3]. The different member

nations are using different commercial tools for implementing the SCA version of the waveform. We (at FKIE, Germany) decided to use the commercial *Software Communications Architecture Reference Implementation* (SCARI) *Software Suite* from *Communications Research Centre* (CRC), Canada [4] for development purposes. Various SCA based implementations of different granularity level for the STANAG 4285 waveform have been developed and analyzed by us with the CRC tools [5]. For testing portability, we are also using the *Open Source SCA Implementation - Embedded* (OSSIE) [6] tools from *Virginia Tech*. This paper focuses on demonstrating the portability and interoperability results achieved using the SCARI and OSSIE tools at FKIE. In addition to these in-house developments, various interoperability tests between different member nations have been performed successfully in the past.



**Fig 1: Target Workflow**

Section 2 discusses the porting efforts required to move an SCA based waveform from a machine running one OE (OSSIE Tools) to a machine using another OE (SCARI Tools) and vice versa. It also explores the various approaches taken in order to achieve portability. Section 3 presents the results of interoperability between two machines running the same waveform together under different Operating Environments (OE). A live demonstration displaying the interoperability results is presented at the final conference.

This project was performed under contract with the Technical Center for Information Technology and Electronics (WTD-81), Germany.

## 2. PORTING WAVEFORMS

A formal definition of Porting can be given as *the ease with which a system or component can be transferred from one hardware or software environment to another* [7]. From the point of view of the NATO group, porting is required in order to exchange various SCA based waveforms/components among partner nations. This means that an SCA based component of a waveform working on the processing element (e.g. GPP) of one nation should also be able to work on the processing element of another nation. Thus, the efforts taken to achieve such targets should be minimal. In this paper we focus only on porting waveforms between different Operating Environments (OEs) running on GPP. Porting waveforms on different processors can have several other issues. Table 1 shows the two different Operating Environments discussed in this paper to port waveforms.

**Table 1: OE Comparison**

	Operating Environment (OE)	
<b>Core Framework</b>	OSSIE (v0.7.0), Virginia Tech, USA (OWD v1.0.16)	SCARI, CRC Canada (SCA Architect v1.1.14)
<b>Operating System</b>	Linux (Fedora 9)	Linux (Ubuntu 8.04)
<b>CORBA</b>	omniORB	TAO ORB

### 2.1. Porting from OSSIE to SCARI

In order to achieve minimal porting efforts for moving an SCA based waveform from a system with one OE to another, various approaches are tested. This sub section describes the issues faced and the proposed solution to port a waveform from an OE using OSSIE to another using SCARI.

In this paper, we avoid the use of term Base and Target Waveform for demonstrating portability. As per the NATO guidelines [8], a Base Waveform is defined as a software which acts as a starting point to develop Target Waveform. It should be the generic code independent of any particular platform. The Target Waveform is defined as the complete executable code (XML, source code, binaries etc) that will run on a particular target platform. In our case we are porting the Target Waveform from one OE to another. Thus, we use the term Base OE and Target OE to avoid confusion.

#### 2.1.1. Base OE (SCARI)

As a first step, the SCA based Transmitter component of the STANAG 4285 waveform is implemented with the OSSIE development tools on one machine. The tool generates necessary SCA based domain profiles in XML and the SCA

based C++ source code for the component as shown by step 1 in Fig 2. This C++ source code takes care of the correct CORBA interfaces for communications etc. The waveform developer has to add additionally, the signal processing code. The several generated domain profiles contain information like Properties, OS, Processor etc used by the component (e.g. Software Package Descriptor (SPD)). As shown in step 2 in Fig 2, it is a good practice for the developer to make a static library of the signal processing code and call it from the generated C++ code. In this way, the signal processing code remains separate and independent of the generated code. The complete source code for SCA and the signal processing is then compiled along with the necessary shared libraries (for Core Framework, ORB etc) to generate a binary. The next step is to port the waveform XML and executables to a Target OE.

#### 2.1.2. Porting to Target OE (OSSIE)

In an ideal case, to achieve portability, the developer must only copy the XML files, the executable and the signal processing library of the base waveform to another machine to make it work. But in real case, at least two issues arise:

1. The generated XML files are not interpreted as expected because it is stated in [6] that the domain profiles of components/applications do not entirely conform to the SCA specifications.
2. The generated executable does not run error free. E.g.: it does not find the shared libraries linked during the compilation.

For instance, the SCARI tools do not interpret the XML files directly because of different expectations in the implementation of XML files. Certain modifications are required in the XML files in order to make them readable for the SCARI tools keeping the SCA specifications in mind.

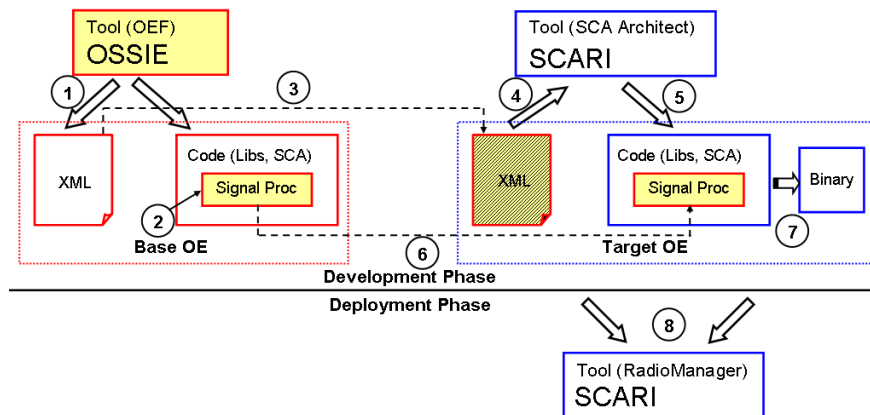
#### 2.1.3. Issues Faced

Following are the changes needed to be made in the different domain profiles/XML files generated by OSSIE:

#### PRF (Properties Descriptor) File

1. The Properties Descriptor file details component and device attribute settings [9 (Section D.4.1)]. Elements like “simple”, “simplesequence”, “test2”, “struct” or “structsequence” are used to describe the attributes of a component. It is observed that in case of defining a component with a “simple” property by OSSIE, the “description” element is created as one of the daughter elements of the “simple” element even though no text is entered while defining that component:

```
<simple>
  <value>number</value>
  <description>desc_name</description>
  .....
</simple>
```



**Fig 2:** Porting steps from OSSIE to SCARI

In the XML file generated by OSSIE, the “*description*” element is placed after the “*value*” element. The SCARI XML parser also checks for correct positioning of the elements according to the SCA defined Document Type Definition (DTD). The DTD defines “*description*” as the first daughter element of “*simple*”.

```
<simple>
  <description>desc_name</description>
  <value>number</value>
  .....
</simple>
```

Therefore, in order to make the SCARI tools read this file, the “*description*” element is moved before the “*value*” element in the OSSIE XML file.

#### SCD (Software Component Descriptor) File

2. This descriptor file is based on the CORBA Component Descriptor specification [9 (Section D.5)]. The root element of this file is “*softwarecomponent*” which in turn has “*interfaces*” as a daughter element. The “*interfaces*” element must have one or more “*interface*” elements. The “*interface*” element describes an interface that the component, either directly or through inheritance, provides, uses, or supports [9 (Section D.5.1.5)]. The SCD file generated by OSSIE contains several “*interface*” elements. As per the SCA specifications, the “*resource*” interface inherits from four other interfaces namely PortSupplier, LifeCycle, TestableObject, PropertySet [10 (Section 3.1.3.1.6)]. In the OSSIE implementation, the inherited interfaces are not declared additionally as “*interface*” elements. The inherited “*interface*” elements have to be added additionally in order to make the SCARI tools accept this file without error.

An example of the additional “*interface*” elements added to this file is:

```
<interface name="LifeCycle"
  repid="IDL:CF/LifeCycle:1.0"/>
<interface name="TestableObject"
  repid="IDL:CF/TestableObject:1.0"/>
<interface name="PropertySet"
  repid="IDL:CF/PropertySet:1.0"/>
<interface name="PortSupplier"
  repid="IDL:CF/PortSupplier:1.0"/>
```

#### SPD (Software Package Descriptor) File

3. In the SPD file generated by OSSIE, again the “*description*” element has to be moved at the right place in the SCA defined XML file according to the DTD, like in the case of prf file.

```
<description>desc_name</description>
```

In the OSSIE implementation of the SPD, the “*description*” element is placed before the “*author*” element inside the “*softpkg*” element. But as per the SCA defined DTD [9 (Section D.2.1)], SCARI parser requires the “*description*” element after the “*author*” element. Thus the element is moved to the right place.

4. As per the SCA specifications [9 (Section D.2.1.6)], the “*implementation*” element needs at least one declaration of either “*os*”, “*processor*” or “*dependency*” element. The SPD file generated by OSSIE contains only the “*processor*” element. But the SCARI tools also need the operating system “*os*” element inside the “*implementation*” element for its internal functioning:

```
<os name="Linux"/>
```

Therefore, this element is added manually in the SCA defined SPD file to run with SCARI tools. This element gives information about the target OS being used to run the particular component.

### SAD (Software Assembly Descriptor) File

5. The SAD file is used to describe the assembled functional application and the interconnection characteristics of SCA components within that application. As per the SCA defined “connectinterface” element of SAD DTD [9 (Section D.6.5.1)], it must contain one “usesport” element and one of the “providesport”, “componentsupportedinterface”, or “findby” element. The DTD specifies that the “usesport” element is written before any of the other three elements. But the OSSIE implemented SAD file writes “providesport” element before “usesport”. Therefore, the order of the port description is changed manually for correct interpretation by SCARI.

```
<usesport>.....</usesport>
<providesport>.....</providesport>
```

6. According to the SCA, the root element of SAD is “softwareassembly” which has two attributes, “id” and “name” [9 (Section D.6)]. The “name” is defined as CDATA in the DTD. This means that name must be a character data and that text will not be parsed. In case of OSSIE, the SAD file writes the name of the application in the “softwareassemblyid” element as OSSIE::saname (saname is user defined text).

```
<softwareassembly id=".." name="OSSIE::saname">
```

The double colons present between OSSIE and saname is not accepted by SCARI tools. Thus, the double colons have to be removed in order to make the SCARI tools read this file.

```
<softwareassembly id=".." name="OSSIEsaname">
```

### File Paths

7. The XML files generated by OSSIE contain hard coded paths for the location of several other files. For Example: In the SPD file of the component, the path for the DTD file containing the schema and the path for property and SCD files are hard coded. The paths are specific to the folder location where they are generated. Some examples are:

```
<!DOCTYPE softpkg SYSTEM “../dtd/softpkg.dtd”>
<localfile name=”xml/comp_folder/comp.scd.xml”>
```

These paths have to be changed in order for the SCARI tools to read the XML files from correct location.

Running the waveform after making all the XML modifications still leaves the developer with the issue of missing shared libraries. In order to overcome that issue, a proposal process is shown in the following subsection.

#### 2.1.4. Proposed Process

The solution to the above mentioned issues is shown step by step in Fig 2. In addition to making modifications in the

XML files, they are collected and zipped for the SCARI tools as shown in step 3 in Fig 2. Next, the XML files generated by OSSIE on Base OE are read by the SCARI tools to model the component again. In other words, the SCA Architect tool of SCARI reads the XML files. This can be seen from step 4. It allows the developer to use the same specifications defined using the Base OE in the XML files for the Target OE. The modeling tool allows the developer to regenerate the SCA specific source code of the component. As a next step, the source code is generated with the new tools shown by step 5. Now, the signal processing part is added to the source code. This is done by using the static library depicted in step 6. This static library containing the signal processing part of the waveform is called from the new generated code. The source code along with the static library is then recompiled with necessary dependencies. This results in the generation of a new binary for the Target OE as shown in step 7. The fresh compiled binary for the particular component is now able to run without problems on the new machine with new tools. Similar steps are carried out on all the components of the waveform. Then as shown in step 8, the complete waveform is able to run with the Radio Manager tool of SCARI. With this approach, one can also avoid problems with different ORBs etc.

The drawback of this approach is that the developer needs to know the signal processing part at his end. The advantage which he has with this approach is that he does not need the component specifications and this frees him of any associated code dependencies.

### 2.2. Porting from SCARI to OSSIE

Investigation into the porting from the OE using SCARI to OSSIE led to some similar issues as described in the previous sub section.

#### 2.2.1. Base OE (SCARI)

Similar to the process mentioned in Section 2.1, again the Transmitter part of STANAG 4285 waveform is implemented. This time, the SCARI tools are used to model the waveform. The SCA Architect tool of SCARI generates the domain profiles (XML files) and the SCA based source code for the waveform. As mentioned in Section 2.1, the signal processing code is compiled into a static library. This library is then called from the SCA source code. These steps are similar to the ones described in Section 2.1.

#### 2.2.2. Porting to Target OE (OSSIE)

Directly porting the waveform developed by SCARI to OSSIE gives up similar issues as discussed before in the previous section. Thus, changes have to be incorporated in order to run the waveform on Target OE.

It is observed that once OSSIE is installed on a system, it creates specific default folders in the file system where

different files related to all the waveforms are kept. The “xml” folder under “sdr” is used to collect all the XML files. This means that for each component build by OSSIE, a separate folder with the component name is created under the “xml” folder. The three domain profiles namely, the SPD, SCD and PRF for each component are kept in corresponding folders. Another folder called “waveforms” is also created under “sdr”. This location is used to keep the domain profiles for the different assemblies. Thus, the SAD for each assembly is placed under a separate folder under “waveforms”. An important point to note here is that this folder also contains a DAS file for the assembly. This file is implemented and used internally by OSSIE and is not SCA specific. This DAS file defines the connection between each component of the assembly to the GPP used with OSSIE. An example DAS file from an original OSSIE application is used as a reference to build the DAS file for the application which is ported. In order to port the waveform from Base OE, careful steps have to be carried out to place the SCARI XML files at correct places in the OSSIE file system.

### 2.2.3. Issues Faced

After placing the XML files generated by SCARI at correct places within the OSSIE file system, we try to read them with OSSIE. Like in the case of SCARI, modifications are needed this time in the SCARI generated XML files to make them readable for OSSIE. Following are the changes to be made:

#### File Paths

1. The file paths in the XML files generated by SCARI do not match the paths expected by OSSIE. Since the XML files are kept according to the OSSIE file structure explained in Subsection 2.2.2, their defined paths are therefore changed. For Example, in the SPD file of a component, the file path:

`<localfile name="comp_name.scd.xml">`

is changed to:

`<localfile name="xml/comp_folder/comp.scd.xml">`

OSSIE does not validate the XML files with the SCA DTD. Thus, no DTD dependent errors are encountered. But again we observe that even after making the XML files readable by OSSIE, we face the same issues of missing shared libraries used by SCARI.

### 2.2.4. Proposed Process

In order to successfully port the waveform to OSSIE, a similar solution is proposed as stated in Section 2.1. The modified XML files are used to remodel and regenerate the SCA source code with OSSIE. The source code is then compiled with the signal processing static library. The new generated binary runs without any problem with OSSIE. Similar steps are carried out on each component of the

waveform in order to successfully achieve waveform portability.

### 2.3. Other Experiences

Similar trials to determine the porting efforts between other OEs are also explored by different NATO nations. In a similar experimental setup as explained in the previous section, tests were performed to port a waveform from OSSIE tools to the development tools from Zeligsoft [11]. The results show that similar XML file modifications are also required in the case of Zeligsoft as presented in the previous section. The point numbers 4, 5, 7 from Section 2.1.3 are also observed in the case of Zeligsoft tests. In addition to that it is also seen while porting the waveform from OSSIE to Zeligsoft that the connections between the components defined by the SAD file have to be modified. OSSIE declare the connections between components using the “findby” element (in case of “usesport” and “providesport”).

`<findby>.....</findby>`

It is observed that after porting the waveform to Zeligsoft tools, the port connection between components is freestanding. The connections between ports are made statically in order to run the waveform successfully on Zeligsoft tools.

## 3. INTEROPERABILITY

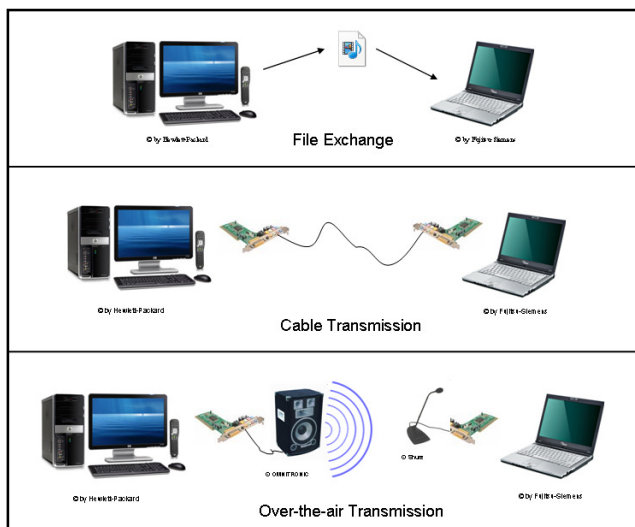
A formal definition of interoperability defines it as *the ability of two or more systems or components to exchange information and to use the information that has been changed* [7]. A real life example of interoperability is the communication between different systems like SDR to SDR, SDR to legacy equipments. For the NATO Group, it is important to know whether the partner nations working on different hardware and software are able to interoperate the STANAG 4285 waveform with each other.

The interoperability between the partner nations is tested in 3 steps:

1. Exchanging data files
2. Transmission through cable
3. Transmission over the air

The three approaches are shown in Fig 3. In the case of “File Exchange” approach, interoperability is achieved by the Receiver decoding either baseband I/Q samples (inphase/quadrature) or IF (intermediate frequency) values written by the Transmitter into a file. In case of the “Cable” and “Over-the-air” approach, the transmission is performed using the IF values. It is possible to use IF values because STANAG 4285 operates in the bandwidth range of 3 kHz with the intermediate frequency of 1.8 kHz. Since these values are in the audible range, we can use “cable” and “over-the-air” transmission with a standard PC sound card as transceiver.





**Fig 3:** Different Interoperability Approaches

Using all the above mentioned modes for testing, the results were positive between the partner nations. For Example: the encoded data sent by the SCA based Transmitter implementation of the waveform by one nation was successfully decoded by the SCA based Receiver of another nation. Similar results were also achieved after performing tests between all the other partner nations.

This paper however focus on the interoperability between the SCA based OSSIE implementation of STANAG 4285 with the SCARI implementation.

### 3.1. Test Setup

The idea behind the test setup is to run the SCA based STANAG 4285 Transmitter functionality on one machine and the Receiver on the other. Instead of writing the IF values given by the Transmitter in an external file, the encoded data is given to the sound card through additional code. The loud speaker jack of the Transmitter machine is connected to the microphone jack of the Receiver machine via an external audio cable. In case of “over-the-air” transmission, a speaker is connected to the Transmitter side and a microphone is connected to the Receiver side. In this way, the sound card of the Receiver machine receives the data coming from the Transmitter machine without the cable. In order to successfully test interoperability, the SCA based Receiver must read this data and decode it. In our implementation the decoded result is then printed out on the console. The live demonstration of the interoperability tests will be presented at the conference.

The STANAG 4285 waveform offers different modes operating at different speeds between the ranges of 75 bits/sec to 2400 bits/sec. All the modes of operation are tested between the Transmitter and Receiver side. Some problems occur while testing the highest mode. The highest

mode is also the least robust mode. The problems can be contributed to the noise level in air and also to the level of sound at which the sound card of the machine is working.

## 4. CONCLUSION

Some important conclusions can be drawn from the porting and interoperability experiences described in the above sections. In the case of portability, we can see that the modified SCA defined XML files help us in modeling the target waveform without the necessity of separate details of the base waveform. Similarly, the approach of making a separate library for the signal processing code helps in protecting the proprietary rights of the vendor. In addition, it frees the waveform developer of going into the details of the signal processing code thus reducing the development time. The interoperability results are proposed to be shown as a live demonstration at the final conference.

## 10. REFERENCES

- [1] “RTO Group” <http://www.rta.nato.int/>
- [2] NATO, Military Agency for Standardization (MAS), “STANAG 4285: Characteristics of 1200/2400/3600 Bits Per Second Single Tone Modulators/Demodulators for HF Radio Links”.
- [3] “Telefunken Racoms” <http://www.tfk-racoms.com/>
- [4] “Communications Research Centre (CRC), Canada” <http://www.crc.gc.ca/>
- [5] S.Singh, M.Adrat, S.Couturier, M.Antweiler, “SCA Based Implementation of STANAG 4285 In a Joint Effort Under the NATO RTO/IST Panel”, Software Defined Radio Technical Conference (SDR’08), Washington D.C, USA, November 2008.
- [6] Open Source SCA Implementation - Embedded, “OSSIE”, <http://ossie.wireless.vt.edu/>
- [7] Standards Coordinating Committee of the IEEE Computer Society, “IEEE Standard Computer Dictionary: A compilation of IEEE Standard Computer Glossaries”, 1990.
- [8] NATO, Software Defined Radio Users Group, “Policy and Framework for Sharing Common Waveforms”, 2009.
- [9] Modular Software-programmable Radio Consortium for JTRS, “Software Communication Architecture Specification, Appendix D. Domain Profile”, Version 2.2, <http://sca.jpeojtrs.mil/>
- [10] Modular Software-programmable Radio Consortium for JTRS, “Software Communication Architecture Specification, Version 2.2”, <http://sca.jpeojtrs.mil/>
- [11] Leif Hanssen, NATO RTO/IST RTG on SDR Meeting, Budapest, June 2008.