Acquiring and Sharing Knowledge for Developing SCA Based Waveforms on SDRs

S. Singh, M. Adrat, M. Antweiler
Fraunhofer FKIE, Wachtberg
GERMANY

sarvpreet.singh@fkie.fraunhofer.de

T. Ulversoy, T.M.O. Mjelde, L. Hanssen
FFI, Kjeller
NORWAY

tore.ulversoy@ffi.no

H. Ozer, A. Zumbul
TUBITAK, Gebze
TURKEY

hozer@uekae.tubitak.gov.tr

ABSTRACT

The NATO Research and Technology Organization (RTO) / Information Systems Technology (IST) Research Task Group (RTG) on Software Defined Radio (SDR), works on the issues concerning SCA based implementations of waveforms on SDRs. This paper gives a review of the motivation, workflow and results of this international working group. With a potential target of building a Waveform Library in the future, it is important that all the partner nations collaborate and work together. In our contribution, we present some of the important results achieved by national research labs in Germany (FKIE), Turkey (TUBITAK), and Norway (FFI) during the tenure of this Group. Firstly, we present the SCA-based implementation results of STANAG 4285 waveform and the effect of increasing the granularity of the SCA waveform application on the system overhead. After that, we show the portability approach taken to port a waveform between different Operating Environments (OEs) and platforms. The portability efforts between the partner nations are also presented. Finally, the interoperability results show how a waveform running on two different OEs of different partner nations is still able to exchange data.

1.0 INTRODUCTION

Over the last few years, Software Communications Architecture (SCA) [1] is gradually playing a major role in the future Software Defined Radios (SDRs) particularly in the military domain. Since the SCA had been initially developed by the US [2], less experience was available among other NATO nations. In order to gain more knowledge of using SCA with SDRs, initially a NATO Exploratory Team (ET) was formed in 2005 which continued to work till 2007. To follow-up the activity of the ET, the NATO Research and Technology Organization (RTO) / Information Systems Technology (IST) Research Task Group (RTG) on Software Defined Radio was established in 2007 with the same people, motivation and workflow, but different (i.e. higher) "status" within the RTO/IST. There are about ten active participating nations both from industry and research labs sharing their experiences in the development of SCA based waveforms in this Group. As a three folded approach (shown in Figure 1) to acquire such knowledge, the group aims at:

- Implementation of an SCA based waveform
- Demonstrating the porting of an SCA based waveform to different SDR platforms
- Demonstrating the interoperability between different SCA based implementations of the same waveform

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
SEP 2010	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Acquiring and Sharing Knowledge for Developing SCA Based Waveforms on SDRs	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Fraunhofer FKIE, Wachtberg GERMANY**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADA568727. Military Communications and Networks (Communications et reseaux militaires). RTO-MP-IST-092

14. ABSTRACT
The NATO Research and Technology Organization (RTO) / Information Systems Technology (IST) Research Task Group (RTG) on Software Defined Radio (SDR), works on the issues concerning SCA based implementations of waveforms on SDRs. This paper gives a review of the motivation, workflow and results of this international working group. With a potential target of building a Waveform Library in the future, it is important that all the partner nations collaborate and work together. In our contribution, we present some of the important results achieved by national research labs in Germany (FKIE), Turkey (TUBITAK), and Norway (FFI) during the tenure of this Group. Firstly, we present the SCA-based implementation results of STANAG 4285 waveform and the effect of increasing the granularity of the SCA waveform application on the system overhead. After that, we show the portability approach taken to port a waveform between different Operating Environments (OEs) and platforms. The portability efforts between the partner nations are also presented. Finally, the interoperability results show how a waveform running on two different OEs of different partner nations is still able to exchange data.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**SAR**	**37**	

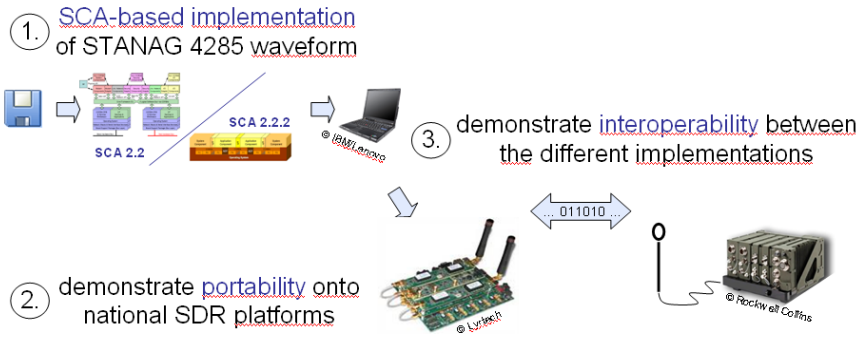


Figure 1: Target Workflow.

The group has jointly agreed on the STANAG 4285 Waveform (HF Medium Data Rate Waveform) [3], for its development work. An example implementation of the waveform in ANSI-C code has been provided by one of the Group members, Telefunken Racoms [4] which is used by FKIE and FFI for research purposes. TUBITAK has implemented their own waveform from scratch using the STANAG 4285 standards. As a first step, an SCA based version of this waveform is implemented. This is explained in Section 2.0. The member nations are using different commercial, open-source or self developed tools for implementing the SCA version of the waveform. As a second step these implementations are ported to different SCA platforms by the participating nations as described in Section 3.0. The last step demonstrates the interoperability between the different implementations on different platforms. Section 4.0 presents these interoperability results. In this way, expertise and common understanding of the various aspects of SCA is built between the participating nations without a high initial cost. The lessons learned during the working of this Group are highlighted in Section 5.0. Finally, Section 6.0 gives a final summary on the paper.

2.0 IMPLEMENTATION OF SCA BASED WAVEFORMS

The SCA is specifically designed as an Operating Environment (OE) for communication devices. The OE consists of Core Framework (CF) services and infrastructure software (including board support packages, operating system and services, and CORBA Middleware services). The software partitions defined by the architecture that illustrate applications are typical of how waveforms might be implemented using the SCA.

For the development of SCA based waveforms for SDRs, the developers usually need some tools to assist them in their work. There can be different levels of support for system/management functions, for standardized software interfaces etc. These tools are required to do the rudimentary work of SCA implementation required again and again for each waveform development.

2.1 SCA Development Tools

FKIE decided to use the SCA Reference Implementation (SCARI) Software Suite from CRC, Canada [5]. In addition to this, FKIE also use the open Source Open-Source SCA Implementation - Embedded (OSSIE) [6] tool from Virginia Tech only for performing porting tests. FFI is using the Component Enabler tool from Zeligsoft [7] as well as the OSSIE tool. In order to model the SCA waveforms, TUBITAK uses the Zeligsoft modeling tool in combination with its self developed SCA Core Framework called TUBITAK – UEKAE Radio Control Software (TURCOS). Another group member, Indra (Spain) is using the Spectra development tool from Prismtech [8]. Using different tools among partner nations allows better analysis and understanding of SCA based waveform implementation.

For the basic implementation of the STANAG 4285 waveform, two SCA based resources are implemented using the development tools, one each for Transmitter and Receiver. Since one of the goals of the NATO Group is to find the cost/overhead of increased number of SCA resources on the system, we need several resources of the waveform to analyze it. Keeping this in mind each of the nations developed various SCA implementations of the STANAG 4285 waveform with multiple resources by increasing the granularity of the waveform implementation.

2.2 Granularity of SCA Applications

Figure 2 shows the block diagram of STANAG 4285 Transmitter [9]. It shows two parts: *Frame Collection* and *Frame Processing*. These names are used in the Telefunken Racom code description used by FKIE and can be named differently in the implementation from TUBITAK. The Transmitter as a single SCA Resource is made by including the complete functionality in one Resource. To increase the granularity, two SCA Resources are made, each with the functionality of *Frame Collection* and *Frame Processing*. A higher level of granularity is achieved by reusing the *Frame Collection* Resource and making new SCA Resources for the divided *Frame Processing* part: one responsible for FEC encoding, the second part for interleaving and the third part for modulation. These are the different levels of granularity achieved at FKIE.

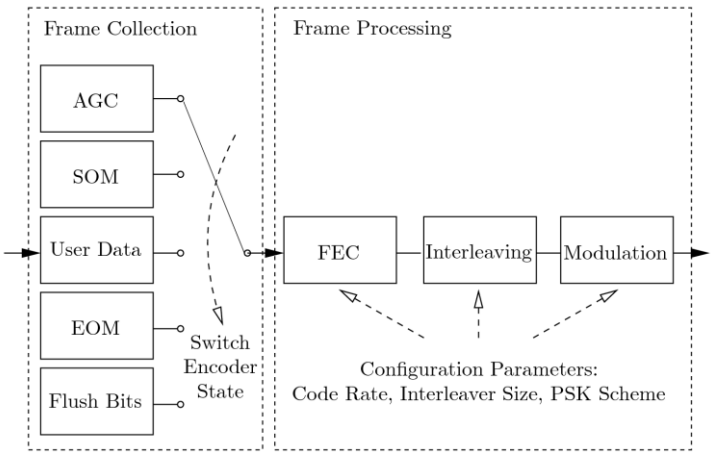


Figure 2: Block Diagram of STANAG 4285 Transmitter.

FFI achieved even higher level of granularity [10]. The Transmitter functionality is divided further over several SCA based Resources. In one implementation, the modulation part is further divided into three SCA Resources handling separate functionalities (see Figure 4). The *Data Source* Resource of FFI is similar to the *Frame Collection* Resource of FKIE. In another implementation, for analysis of SCA overhead, four dummy SCA Resources were added (see *Forwarder* Resource in Figure 4). These dummy Resources were used to forward data to the next SCA Resource.

2.3 Profiling Results

The profiling of the above mentioned SCA based implementations by FKIE and FFI are explained below in separate subsections.

2.3.1 Results at FKIE

In order to analyze the overhead of running multiple SCA based Resources on the processor; various profiling tests are investigated at FKIE [9]. FKIE used a profiling tool called Callgrind, available from the

Valgrind [11] tool suite to profile the SCA Resources for the different test implementations. To perform profiling, a text is read from a file and then used by the Transmitter Resource for encoding. The profiling results include the waveform specific overheads of STANAG 4285 for transmitting the EOM (End Of Message), SOM (Start Of Message), Flush Bits associated with the Transmitter data. Keeping the mode of STANAG 4285 same at 75 bits/sec, various tests are made by transmitting different amount of data which in turn leads to different number of frames transmitted [9]. Some terms used with respect to the profiling tool are defined:

- The *Cost* of a function is defined by the event counts of a particular function. This means the number of instructions/data accesses. It also tells us of the instructions that do/do not reference memory.
- The *Self Cost* is the cost of the function itself.

In this section we focus on the *Self Cost*. Whenever the term 'cost' is used in this section, we refer to the *Self Cost*.

After examining the data from the profiling results, the total cost of running an implementation is classified on the basis of ELF (Executable and Linking format) objects. The major contributions towards the total cost made by the ELFs are under the Executable, ACETAO (Related to CORBA), SCA, C/C++ and ld.so categories. The results for the different implementations of STANAG 4285 with different granularity levels give different results. It is seen that in case of a multi SCA Resource implementation, there are some Resources whose cost of running the Executable is less than the cost of total overheads involved. This leads to an important conclusion that while increasing the granularity of an SCA based waveform application, it is important to make sure that each SCA Resource is doing considerable amount of signal processing. Otherwise, the cost of overheads will be more than the cost of running the SCA Resource functionality itself.

Figure 3 shows the overall cost comparison of SCA based implementations of STANAG 4285 at different granularity levels. The left part of the Figure shows the number of frames on the X-axis and the Self Costs on Y-axis. The dashed curves show the total sum of overheads for running the Transmitter as a single SCA Resource, as two SCA Resources (Frame Collection and Frame Processing) and as four SCA Resources. The solid curves show the combined cost of the 'Executable' of the Transmitter at different levels of granularity. The lower two solid curves show the cost of the 'Executable' of one Resource and two Resource implementations. There is not much difference in the curves since dividing the complete Transmitter functionality into *Frame Collection* and *Frame Processing* required only minor changes in the C-code. There is however a higher cost of ‘Executable’ for a four Resource implementation because the separation of *Frame Processing* into FEC (Forward Error Correction), Interleaving and Modulation required a reorganization of the C-code leading to a higher signal processing cost. The dashed curves show that there is an increase in the total cost of overhead when functionality is divided into several SCA Resources. We can also observe by extrapolation of this graph that at one point splitting an SCA Resource into more number of SCA Resources (e.g. more than five Resources) will lead to a higher cost of overheads than the 'Executable'.

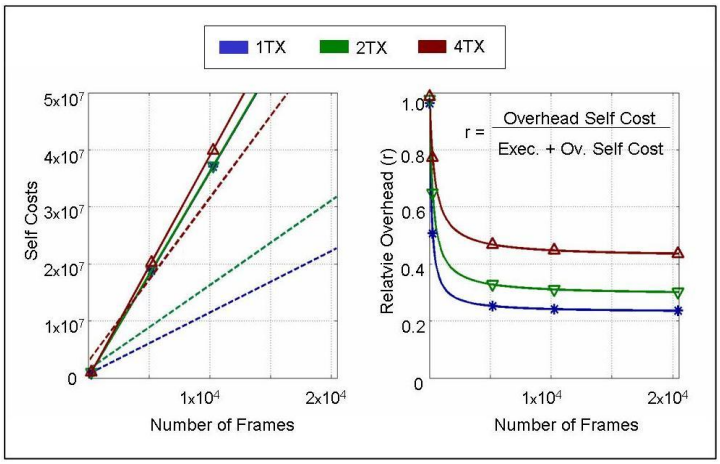


Figure 3: Overall Cost Comparison for TX.

The right side of Figure 3 shows the comparison of the relative overhead cost with the number of frames processed. It can be concluded from the graph that increasing the number of SCA Resources for an implementation leads to increase in the relative cost. It can also be seen that the relative overhead cost is more for processing less number of frames. It becomes less and almost constant for processing higher number of frames. More details can be found in [9].

Testing the SCA based Transmitter implementations at different modes of the STANAG 4285 (e.g. 2.4 kbits/sec) also give similar results as mentioned above. In addition to the Transmitter, several implementations of the Receiver at different granularity levels are also tested at FKIE which gives supporting evidence as in the case of the Transmitter [12].

2.3.2 Results at FFI

In order to investigate into SCA based application granularity when running on a CORBA-capable GPP, different versions of the STANAG 4285 Transmitter application are made, each split into a different number of components.

The experiment is run on a Pentium M 1.86GHz with Linux 2.6.9-34EL and using the OSSIE Core Framework version 0.6.0. Since the Transmitter application represents a low workload for this processor at the original symbol rate of 2400 symbols per second, the processing is run at an increased symbol rate of 25600 symbols per second, in order to get more significant CPU workload readings. The measurements are made using the performance monitoring tool 'SYSSTAT sar' [13], and measured as an average over 5 periods of 40 seconds.

Figure 4 shows the total measured CPU workload for a 2-component, 7-component and 11-component version of the waveform application. This is compared also to the measured workload for a non-SCA 'C'-language implementation of the same waveform. It is seen that the more components the application is split into, the higher is the workload. In [10], it is shown that a major portion of the workload overhead of the multi-component-versions in this case is caused by the inter component communication through CORBA. The losses will be relatively less significant, however, when the useful processing in each component represents a higher workload. The losses can be reduced also by reducing the rate of the inter component communication.

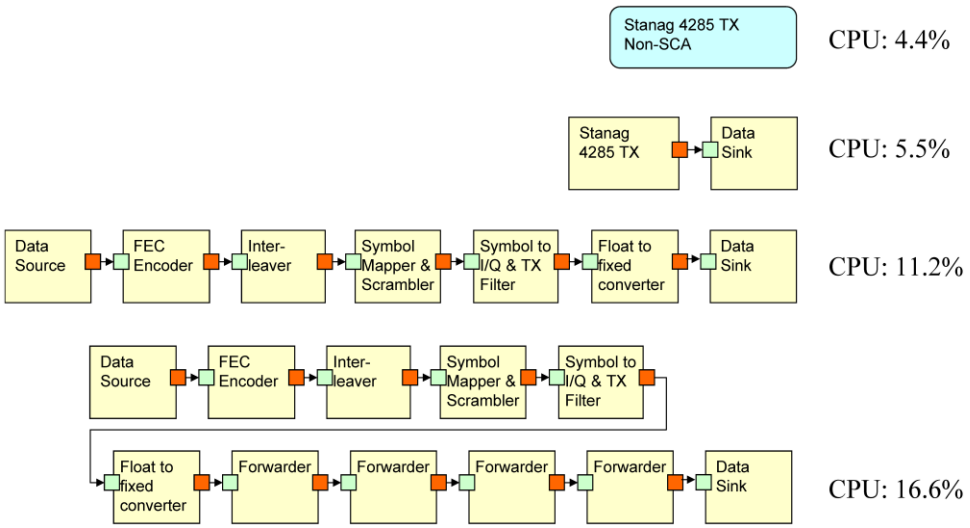


Figure 4: User+system CPU % for the non-SCA 4285 TX version, compared to three SCA-based versions with different granularity. Measured at an increased symbol rate of 25600 symb/sec).

3.0 PORTABILITY RESULTS

Porting can be defined as *the ease with which a system or component can be transferred from one hardware or software environment to another* [14]. Here we discuss only the technical aspects. The security aspect is out of scope in this study. It is anticipated that in future there will be a Waveform Library where all the member nations can add and share SCA based waveforms. This means that one nation can add an SCA based waveform to this library and the other nations can use this shared waveform and run it on their platforms. Therefore, the Group members are interested in investigating the portability of waveforms. Since FKIE, FFI and TUBITAK are using different Operating Environments (OEs), the portability tests are a good starting point to investigate.

3.1 In-House Porting Efforts (FFI)

3.2.1 From OSSIE to WDS

The initial SCA based STANAG 4285 implementation at FFI is implemented using the OSSIE CF (version 0.6.2) and the OSSIE Waveform Developer (OWD) tool. This implementation is then ported onto the Spectrum's Waveform Design Studio (WDS) platform. The WDS has a Harris SCA CF and a TAO ORB for CORBA. The Zeligsoft Component Enabler (CE) is the tool used on the WDS for SCA waveform modeling, for generating skeleton code and XML-files as well as for application deployment and runtime management.

For the initial porting effort and since the STANAG 4285 has low processing workload requirements, the application is ported onto GPP-implementation components only. This subsection describes the lessons learned from this porting from OSSIE onto WDS.

While compiling and using the C++ code level component files directly on WDS ideally should have been possible, it turns out that OSSIE and WDS has different naming conventions in their auto-generated skeleton code, thus using the code level files directly would not work and manual editing would be tedious. Instead that, a three-step procedure for the porting is executed: First, the SCA structure XML files (the Domain Profile files) are moved over to WDS and loaded into the CE tool. Second, the code generation tool in CE is used to generate new component skeleton code. Third, the actual component processing code is moved into the new component implementations.

An empty project is created on the Spectrum platform, onto which the SCA XML (domain profile) files can be imported. Several unexpected issues are discovered in this import operation [16]. The next obstacle encountered is that OSSIE's SAD files describe all component ports as *dynamic*. This implies that the connections are to be resolved at run-time. For reasons not identified, this does not work on the WDS. The workaround is to delete all dynamic connections and reconnect all interfaces with static connections.

After having generated the new component skeleton code from CE, the processing code is copied over from the respective OSSIE-generated components. A subtle difference between OSSIE and CE here is that the OSSIE framework depends on a separate processing thread where the components' processing code is run, but CE had no such prerequisite. To avoid changing the structure of the code, processing code is inserted in the component's start-up methods for creating a thread that can run the processing code in the same manner as in the OSSIE-generated components.

In summary, the porting is not a very difficult operation but it involves a lot of tedious fault finding. Some of the issues are caused by subtle details not adhering strictly to SCA, while others are due to differences in approaches taken in the OSSIE and CE tools.

3.2.2 From GPP Implementation to DSP Implementation on WDS

As an experiment, one of the components, the FEC Encoder, is ported onto one of the DSPs of the WDS. Since the WDS platform does not have CORBA-enabled DSPs, a GPP proxy component is made that communicates with the DSP-implemented component through Spectrum's 'quicComm' interface commands. This proxy relays the through-CORBA input port data to the DSP, using quicComm memory write commands. It also fetches the DSP-processed data by memory read commands, and forwards the data on the output port. The proxy obtains a reference to the DSP by inquiring the DSPs logical device.

The actual porting of the functional processing code from the FEC Encoder GPP component to the DSP is found to be a low level effort. Since this processing code is already written in 'C', only the quicComm input/output communication code has to be added. The SCA-level changes, e.g. adding the DSP component and implementation and inputting the correct properties and dependencies are found to be relatively a much more labor some job, requiring precise string-inputs and being difficult to debug.

Since the proxy to DSP worker communication in this case uses proprietary quicComm API commands, the resulting application obviously is not readily portable to other platforms without at the same time changing the code. The SCA, however, is still found to be of some limited benefit in terms of making the application portable. The SCA approach helps in concentrating the platform dependent code, such that it is easy to locate and change when needed. It also helps in providing a handle to the actual DSP where the target code is deployed, and in loading the code onto the target DSP.

3.2 In-House Porting Efforts (FKIE)

FKIE tested the portability in-house by porting waveforms developed with OSSIE development tools onto the SCARI development tools [15]. Both the tools provide two different Operating Environment (OE) which helps in understanding the porting efforts involved. An approach is proposed in this Section as shown in Figure 5.

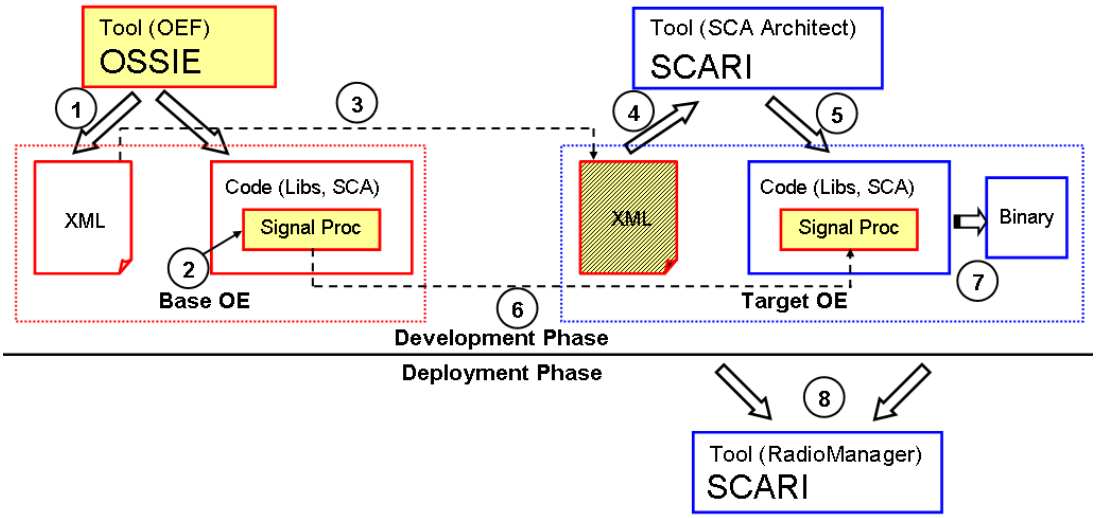


Figure 5: Porting Steps from OSSIE to SCARI.

In this case we start by developing an SCA based application with the OSSIE tools. The OSSIE tools generate SCA defined XML Domain Profiles and the skeleton code for a Component as shown by Step 1 in Figure 5. All the necessary interfaces according to the SCA specifications and CORBA interfaces are implemented in the source code. The waveform developer has to only add the necessary signal processing code inside this source code. As a good programming practice, the developer can compile the signal processing code in static libraries. These libraries can then be called from the SCA based skeleton code of the Component as shown in Step 2. This way there is a clear separation between the signal processing code and the SCA based code. The complete SCA based source code is compiled together with the signal processing static libraries and the necessary shared libraries (associated to the Core Framework, ORB etc) to generate a library. This way we can make an SCA based application. Now in an ideal case, we must only copy the necessary SCA defined XML files along with the executables and signal processing library of the Components to another machine to port this application to another OE. But we are faced with at least two issues. In our case the SCARI tools are not able to read the XML Domain Profiles directly because of different implementation of these XML files. Some changes are needed to be done in the XML files to enable error free reading by SCARI tools [15]. In addition to that, the Executable does not run error free on the different OE (due to the missing shared libraries required while compiling).

In order to overcome the above issue, a porting process is proposed. As shown in the Step 3 of Figure 5, the necessary changes are made to the XML files. In the next Step 4, these XML files are read by the SCARI tools to model the components again. Using this approach, the developer using SCARI can use the same specifications defined during modeling with OSSIE. In Step 5, new SCA based source code is generated for the components using the SCARI tools. The developer can now add the signal processing code in this code as shown in Step 6. This requires only using the same static library for the signal processing code and calling it from within the SCA based code. The source code along with the static library is then recompiled with necessary dependencies. As a result, a new binary is generated as shown in Step 7. 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.

The drawback of this approach is that the developer needs to know the signal processing part at his end. On the other hand the advantage is that the developer does not need the component specifications and this frees him of any associated code dependencies. The above process is only proposed for porting a waveform from one GPP based OE to another. It is possible that additional efforts are needed to port waveforms between different processor architectures. More details can be found in [15].

3.3 In-House Porting Efforts (TUBITAK)

TUBITAK also performed various in-house tests between several OSs, ORBs and CFs. VxWorks, Linux and Integrity are the operating systems which are used by TUBITAK. SCARI and TUBITAK's own SCA Core Framework called TURCOS (TUBITAK – UEKAE Radio Control Software) are used as SCA Core Frameworks. In addition to that several efforts are taken to perform CF porting to different operating systems. Table 2 summarizes some of the waveform porting efforts at TUBITAK.

Table 2: Platform Configurations.

Waveforms	Ported Platforms
STANAG 4285	TURCOS + ACETAO + Linux (Pardus) @ x86
	TURCOS + ACETAO + Linux (Ubuntu) @ ppc
	TURCOS + ORBExpress + VxWorks @ ppc
	SCARI GT + ORBExpress + Integrity @ ppc
TUBITAK Reference Waveform (TRWF)	TURCOS + ACETAO + Linux (Pardus) @ x86
	SCARI-Open + ACETAO + Linux (Pardus) @ ppc

STANAG 4285 and TUBITAK Reference Waveform (TRWF) [17] (It can be freely downloaded from the Wireless Innovation Forum [18]) have been ported to several platforms. Both waveforms are ported among different Core Frameworks, ORBs, Operating Systems, and GPP targets. The Zeligsoft tool is used as modeling tool. The general procedure is as follows, the model is developed on Zeligsoft tool, and the released IDL are used for interfaces. Once the specifications about target platforms (OS, ORB, CF etc.) are set, then the skeleton code and XML files are generated. The functional code is inserted into skeleton code and executable files are generated. The executables and XML files are then loaded using a human machine interface to the platform.

For porting the developed waveform to a new platform, same model is used. The model is opened with the modeling tool and then the specifications are set for the new target platform. The remaining process is done as given above for the new platform. The main issue is that the Zeligsoft tool should have the target platform specifications. If it does not have specification option for the object platform, a patch for the target specification should be developed or ordered to the modeling tool developer company and added to the tool.

TUBITAK implemented its own code for the STANAG 4285 waveform. The SCA implementation of the waveform is also ported to a SCA certified platform (i.e. SDR 4000 [19]). The platform has Integrity OS, SCARI GT Core Framework and ORBExpress ORB. Because of its low processing requirement, baseband processing of the waveform is run only on the GPP of the platform.

The following issues have been encountered during the porting stages of these waveforms:

- **Core Framework Incompatibilities:** The SCA specifications do not oblige every detail of implementing a core framework. Some of the details are left to developers in order to make SCA easier to understand and to allow developers to customize their core frameworks according to customer requirements. The following list contains the main incompatibilities between TURCOS, SCARI-GT and SCARI-Open:

- **Port Names:** SCA specifications do not define a naming standard for SCA Ports in device and service implementations. Therefore, porting a waveform from one Core Framework to another requires changing these names in the XML files of the waveform.
- **Behavior Differences:** Each CF may have different behaviors. For example, SCARI-Open and SCARI-GT register the devices with the naming service and therefore connections between waveform components and Core Framework devices can be established over naming service; however TURCOS does not register devices with naming service for security reasons, therefore connections in XML files have to be modified accordingly. Another example is the AudioDevice implementation differences between SCARI and TURCOS. In TURCOS, AudioDevice automatically starts pushing packets when a waveform which requires audio is installed and started, however in SCARI-Open AudioDevice has to be started manually by the installed waveform which requires additional CORBA connection to AudioDevice over Resource interface.
- **Installation Procedures:** Each Core Framework has its own procedure to install applications. Even different versions of the same core framework may require different steps. For example, SCARI-Open installs zipped waveform files with a jar extension, SCARI-GT installs zipped waveform files with a zip extension, whereas TURCOS does not require any zipping operation and it can install waveforms from a folder.

- **Code Generation Differences between the Tools:** Component Enabler from Zeligsoft and SCA Architect from CRC are the tools that are used for modeling SCA applications. These tools have some differences in terms of generating code and XML files. With this regard, SCA Architect tool is more compatible with CRC's core framework but the generated code is less portable due to linking requirement with CRC dependent libraries. On the other hand, Zeligsoft Component Enabler requires target specific code generation files. Regenerating the XML files by reading the model made by the other tools, mostly do not work in the first attempt. Some dependency problems and naming incompatibilities exist between these tools. Solving these problems require modifications to the auto-generated XML files which is an error-prone process.

- **Differences between Operating Systems:** Each OS comes with its own running mechanisms and development tool chains. Porting a source code from one operating system to another is always a risk in case of portability. It is strictly dependent on the code implementation. A common rule states: a more independent code is the most portable code. OS dependent libraries and system calls prevent portability. Although POSIX standard guarantees a set of common system functions, there exist some exceptions to that between different operating systems. Linux, Integrity and VxWorks are three different operating systems on which TUBITAK has ported waveforms. The main difference between these operating systems is memory management. Integrity and VxWorks are Real Time Operating Systems (RTOS) and provide process separation concept. In addition, stack and heap size dedicated to each executable component of the waveform differs between these operating systems. Determining optimum stack and heap size for each waveform component and preventing memory leaks is crucial to achieve successful portability.

- **Differences between ORB Implementations:** Although CORBA is a well-defined standard, all ORBs are not implemented in the same way. There exist some incompatibilities between them. For example, ACETAO and ORBExpress have some non-portable macro definitions. In order to implement portable code these macros must be avoided. Also they have some differences in CORBA string and memory management operations.

- **Differences between Target GPP Architectures:** x86 and power pc (ppc) are the two different GPP architectures on which TUBITAK has ported waveform. These processors are different in terms of being big-endian or little-endian. Also, processor speeds may differ between the ported platforms. Porting a waveform on a slower processor may cause performance problems.

4.0 INTEROPERABILITY RESULTS

Interoperability can be defined as *the ability of two or more systems or components to exchange information and to use the information that has been changed* [14]. A real life example of interoperability is the communication between different systems like SDR to SDR, SDR to legacy equipments. Since interoperability is also one of the points of interest 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. For example: If the Receiver part of the STANAG 4285 developed on one implementation is able to decode the data encoded by the Transmitter part running on a different implementation. For the following tests with the STANAG 4285, we focus on the baseband signal (I/Q-values) and the IF-Signal (carrier freq. 1.8 kHz, bandwidth 3 kHz). We do not mix the signal up into the RF-domain (HF: 1.5 MHz - 30 MHz). The interoperability for the STANAG 4285 is tested in 3 steps:

- Exchanging data files
- Transmission through cable
- Transmission over the air (acoustics)

Exchanging data files means that on one PC, the TX of the waveform runs and encodes data to a wave file; the created wave file is then copied to another PC, where it is decoded by the RX. The tests by exchanging files between FKIE, TUBITAK, FFI showed that all combinations of implementations and modes work without any problems or en-/decoding errors. With these results, the offline interoperability by file exchange approach is proven for all different implementations.

Cable based approach can be considered as having an almost perfect transmission channel. For these tests, two PCs running the SCA based STANAG 4285 are connected via an audio cable. On TX side, the cable is plugged into the line-out port and at the RX side it is plugged to the line-in port.

Over the air approach can be considered as a real channel with challenges like multipath, attenuation etc. For these tests, the cable connection between TX and RX is replaced by a loudspeaker and a microphone. The line-out port of the PC running TX is connected to a loudspeaker and the line-in port of the RX side is connected to a microphone.

4.1 In-House Interoperability Tests (FKIE)

In addition to the implementation of SCA based implementations of the STANAG 4285, several non SCA based implementations are also developed. A plain C code of the Transmitter and Receiver is tested with the SCA based implementations. A Matlab implementation of the waveform is also tested. All these tests gave positive results for interoperability. This means that the encoded data by the Transmitter implementation is successfully decoded by the Receiver implementation by using all the three approaches mentioned above.

Interoperability tests are also conducted between the two SCA based implementations of STANAG 4285 using two different OEs. For testing purposes, FKIE also implemented the waveform using the OSSIE tools in addition to the SCARI implementations. Since OSSIE and SCARI tools use different SCA Core Framework implementations, CORBA Orbs, on different distribution of Linux, it presents a more concrete proof of the interoperability tests. The interoperability tests were positive also in this case using all the three approaches of testing.

A point to note here is that in case of using the sound card to transfer data between two PCs, the success of interoperability tests also depend on the quality of hardware used. A better quality sound card with good quality speakers and microphone give better results.

4.2 In-House Interoperability Tests (FFI)

Two different implementations of STANAG 4285 have been developed at FFI, one using the Zeligsoft tools running on the Spectrum SDR-2000 platform and the other using OSSIE tools which are done in several versions. These implementations have been tested against each other successfully. The tests were conducted over a wire and over the air from a speaker to a microphone with the process discussed above.

Since both implementations are based on the same code provided by Telefunken, the Zeligsoft implementation was tested with a legacy HF modem (HARRIS RF-5710A modem). This test also worked successfully and confirmed that the SDR implementations met the specifications given in the STANAG. The tests were performed both in synchronous and asynchronous way (using the RS-232 standard interface).

4.3 In-House Interoperability Tests (TUBITAK)

TUBITAK tested the self implemented SCA based waveforms (STANAG 4285) also (see above) with a legacy HF modem (HARRIS RF-5710A modem). The interoperability tests are done in three phases. First, the recorded IF data files are exchanged between the two systems and then decoded. In the second phase, the IF output of the modem is connected to the SCA based platforms (PC and SCA certified system). Then by running RX and TX mode of the waveform, data is transferred from one side to the other side in online manner. At the last phase over-the-air communication is tested. By using HF radio systems (HF radio and antennas), the SCA based waveform is tested with the legacy modem mentioned above.

4.4 International Interoperability Tests

Various tests are performed between FKIE and TUBITAK with the SCA based implementations of STANAG 4285 waveform to check for Interoperability. Both the *cable based* approach and the *over the air* approach were tested between different OEs. Table 1 shows the difference in the two OEs used for testing. The left column of Table 2 describes the OE used by FKIE and the right column used by TUBITAK.

Table 2: OE Comparison.

	Operating Environment (OE)	
Core Framework	SCARI, CRC Canada (SCA Architect v1.1.14)	TURCOS
Operating System	Linux (Ubuntu 8.04)	Linux (Pardus[20])
CORBA	ACE TAO	ACE TAO

In this case, in addition to a difference in the OEs, there is also a difference in the implementation of the STANAG 4285 waveform. FKIE uses the implementation from Telefunken Racoms and TUBITAK uses its own self developed implementation of the STANAG 4285. Using the above mentioned three approaches, further tests proved successful interoperability.

5.0 LESSONS LEARNED

There are a number of lessons learned during the duration of this working group regarding the implementation of SCA-based waveforms. Some of them are presented in this Section.

5.1 Implementation

- The signal processing functionality should be distributed efficiently among the SCA Resources. Each SCA Resource should perform considerable amount of signal (payload) processing so that the overheads involved in inter-resource communication and context switching are insignificant relative to the workload represented by the signal processing part.
- Having such a high granularity that a high number of SCA-Resources need to be run on the same processing element should be avoided from a computational efficiency viewpoint. In such a case, the granularity of the application should be considered reduced, with fewer SCA-Resources implementing the processing functionality.
- There is usually a trade-off between computational efficiency and reusability of an SCA Resource. Having small, well defined resources increases the potential of being able to reuse them in a different waveform. At the same time, too small resources may lead to increased computational overheads, as mentioned above.

5.2 Portability

- Several SCA based development tools are available in the market. Although the SCA is a fixed standard followed by different tool vendors, there still exist some differences in the code generated by these tools. Therefore, special attention must be given to these differences during the porting efforts.
- Differences also exist between different implementations of SCA Core Framework, Operating Systems, and Processing elements. Porting a waveform is not a single step process and needs prior knowledge of these differences for successful porting.
- The library approach is helpful in creating the SCA based Resources which have to be ported. Keeping the signal processing code separate in a static library which can be called from the SCA based source code provides more flexibility. This way the SCA based source code is separate from the signal processing code. The developer only needs to reuse this library and call it from the new SCA based Resource.

5.3 Interoperability

- Various interoperability results have shown that there is little impact of SCA implementation on waveform applications. The issues which come up during the tests are more related to the hardware. For example: In our tests the quality of the sound card and its correct settings were important for achieving correct interoperability results.

6.0 CONCLUSIONS

The paper presents an overview of the work done in the NATO Research and Technology Organization (RTO) / Information Systems Technology (IST) Research Task Group (RTG) on Software Defined Radio. The various steps of implementation, porting and interoperability of SCA based waveforms carried out by FKIE, TUBITAK and FFI are discussed briefly. These steps have helped in providing the Group members with better understanding of SCA. It is learned that the SCA Resources should perform significant signal (or payload) processing relative to the associated computational overheads. A balance is needed between ease of reuse of SCA components, for which a high granularity of SCA components in an application is favorable, and that of low computational overheads, which points towards a low granularity. It is seen that porting is not a single step process and requires considerable knowledge of the differences in the implementations of SCA Core Frameworks, Operating Systems, SCA development tools etc for efficient porting of SCA based waveforms. And interoperability of SCA based waveforms can be achieved successfully without any major effect of SCA on the waveform. These important lessons learnt during the work done in this Group help in more efficient implementation of SCA based waveforms.

7.0 REFERENCES

[1] Website of Software Communications Architecture (SCA) *"http://jpeojtrs.mil/"*.

[2] Website of Joint Program Executive Office (JPEO) Joint Tactical Radio System (JTRS) *"http://sca.jpeojtrs.mil/"*.

[3] NATO, Military Agency for Standardization (MAS), *"STANAG 4285: Characteristics of 1200/2400/3600 Bits Per Second Single Tone Modulators/Demodulators for HF Radio Links"*.

[4] Website of Telefunken Racoms *"http://www.tfk-racoms.com/"*.

[5] Website of Communications Research Centre (CRC), Canada *"http://www.crc.gc.ca/"*.

[6] Website of Open Source SCA Implementation - Embedded, OSSIE *"http://ossie.wireless.vt.edu/"*.

[7] Website of Component Enabler, Zeligsoft, *"http://www.zeligsoft.com/"*.

[8] Website of Spectra Tools, Prismtech, *"http://www.prismtech.com/"*.

[9] S. Singh et al., "SCA-based Implementation of STANAG 4285 in a Joint Effort Under the NATO RTO/IST Panel", SDR Forum Technical Conference (SDR'08), Washington D.C, November, 2008.

[10] T. Ulversoy, O.Neset, "On Workload in an SCA-based System, with Varying Component and Data Packet Sizes", RTO-MP-IST-083 Military Communications with a Special Focus on Tactical Communications for Network Centric Operations, Apr 2008.

[11] Website of Valgrind Tool Suite, *"http://valgrind.org/"*.

[12] S. Singh, M. Adrat, "Waveform Development for SCA-based Software Defined Radios", FKIE Report-185, November 2009.

[13] Website of SYSSTAT *"http://pagespersoorange.fr/sebastien.godard/"*.

[14] Standards Coordinating Committee of the IEEE Computer Society, *"IEEE Standard Computer Dictionary: A compilation of IEEE Standard Computer Glossaries"*, 1990.

[15] S. Singh et al., "NATO RTO/IST RTG On SDR: Demonstrating Portability and Interoperability of SCA-based Waveforms", SDR Forum Technical Conference (SDR'09), Washington D.C, December, 2009.

[16] J. O. Neset, "Software Defined Radio - Analysis of portability issues using Open Source SCA Implementation - Embedded," Master's thesis, NTNU, June 2008.

[17] A. Zumbul, *"TUBITAK Reference Waveform"*, SDR Forum Technical Conference (SDR'09), Washington D.C, December, 2009.

[18] Website of Wireless Innovation Forum, *"http://www.wirelessinnovation.org/"*.

[19] Website of Spectrum Signal Processing by Vecima "*http://www.spectrumsignal.com/".*

[20] Website of Pardus "http://www.pardus.org.tr".

NATO OTAN

North Atlantic Treaty Organisation

Research & Technology Organisation (RTO)

NATO RTO/IST RTG on SDR:

Acquiring and sharing knowledge for developing SCA-based Waveforms on SDRs

S. Singh, M. Adrat, M. Antweiler
Fraunhofer FKIE, Germany

T. Ulversoy, T.M.O. Mjelde, L. Hanssen
FFI, Norway

H. Ozer, A. Zumbul
TUBITAK, Turkey

NATO Information Systems Technology Symposium (IST - 092/RSY - 022)
Wroclaw, Poland, September 28th – 29th, 2010

Outline

- NATO Group and Motivation
- Waveform Implementation
- Portability Approach
- Interoperability Approach
- Conclusions

NATO RTO/IST RTG on SDR

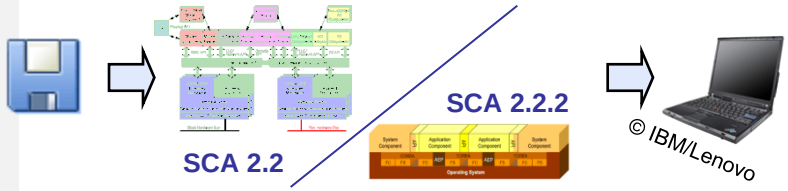
- Research Task Group on SDR
  - was founded below the NATO *Research and Technology Organisation* (RTO) *Information System Technology* (IST) panel
  - the team consists of experts from government and industry from CA, DK, FI, GE, HU, IT, NL, NO, SP, TU, US and: SDR-Forum
  - official duration from July 2007 until October 2010
- Main objectives of the group
  - share knowledge & experience of (multi)national SDR/SCA developments
  - report on possibilities of sharing waveforms and waveform components

→ demonstrate portability and interoperability of SCA-based waveforms

Motivation

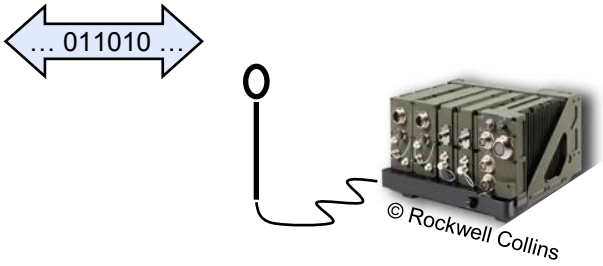
- Work flow

1. SCA-based implementation of STANAG 4285 waveform



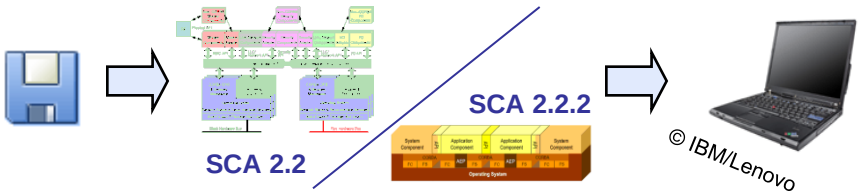
2. demonstrate portability onto national SDR platforms

3. demonstrate interoperability between the different implementations



Some Details on Step 1.

1. SCA-based implementation of STANAG 4285 waveform



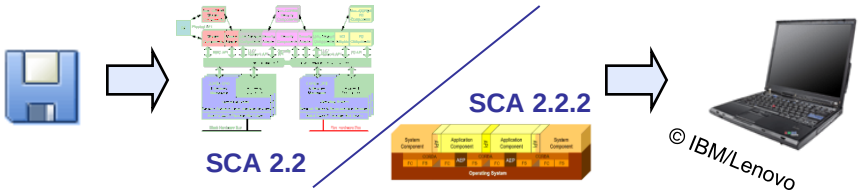
- What is the purpose of the SCA (Software Communications Architecture)?

> „The main purpose of the SCA specification is to define the Operating Environment software, also commonly referred to as Core Framework, which implements the core management, deployment, configuration, and control of the radio system and the applications that run on the radio“ [Bard,Kovarik]

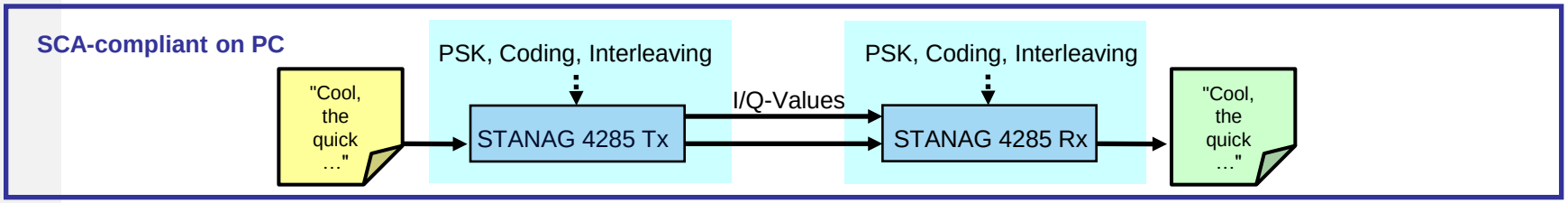
- Some Key Elements
  - SCA XML Domain Profiles (SAD, SPD, SCD …)
  - CORBA IDL Interfaces (connection between SCA Resources)

Some Details on Step 1.

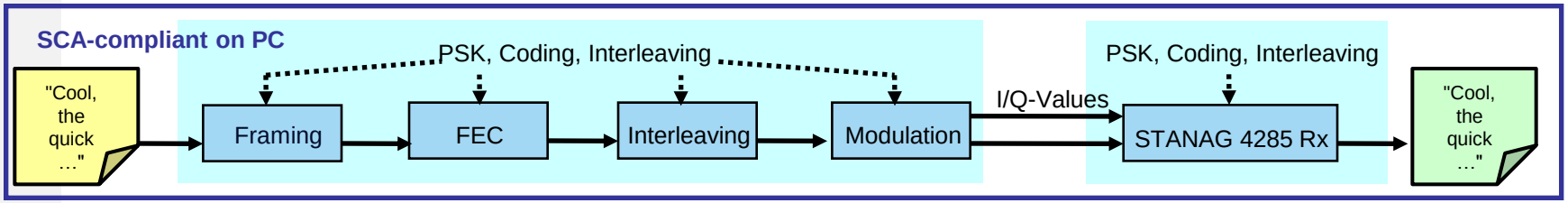
1. SCA-based implementation of STANAG 4285 waveform



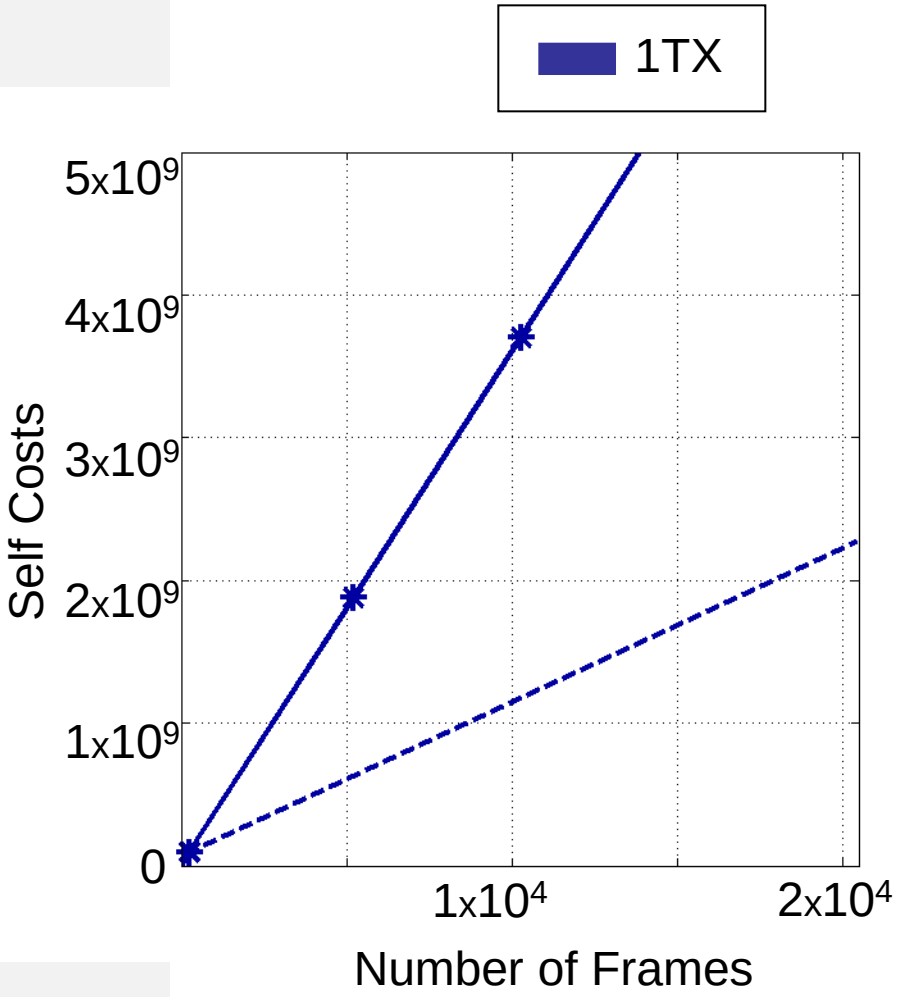
- transmitter (TX) and receiver (RX) as one SCA resource each

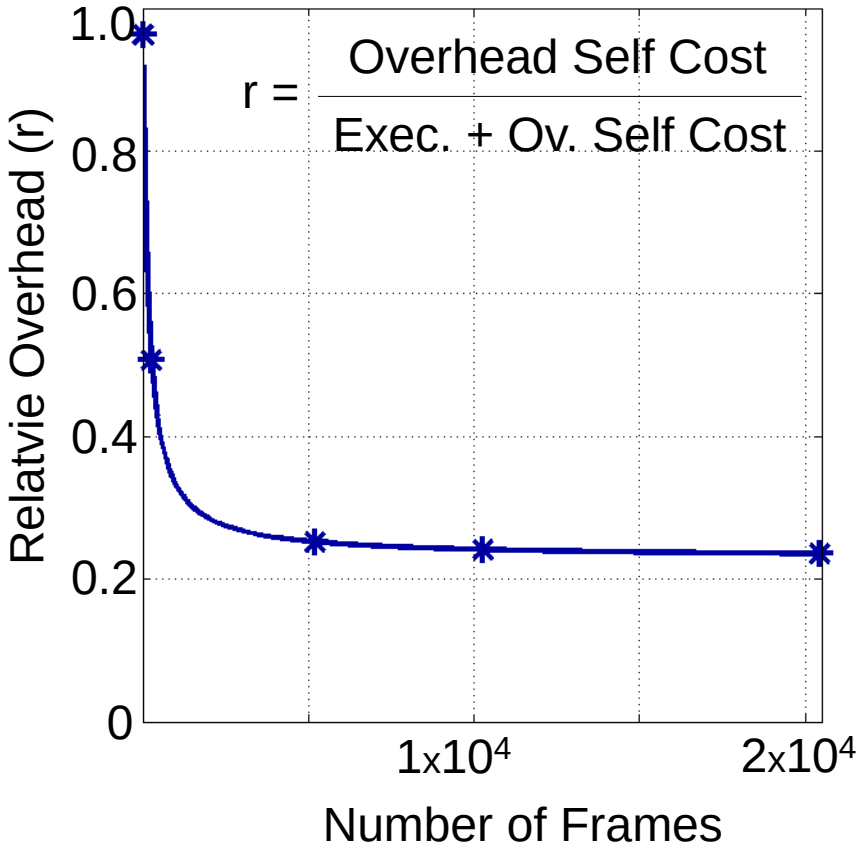


- TX split into four SCA resources and RX still as one SCA resource



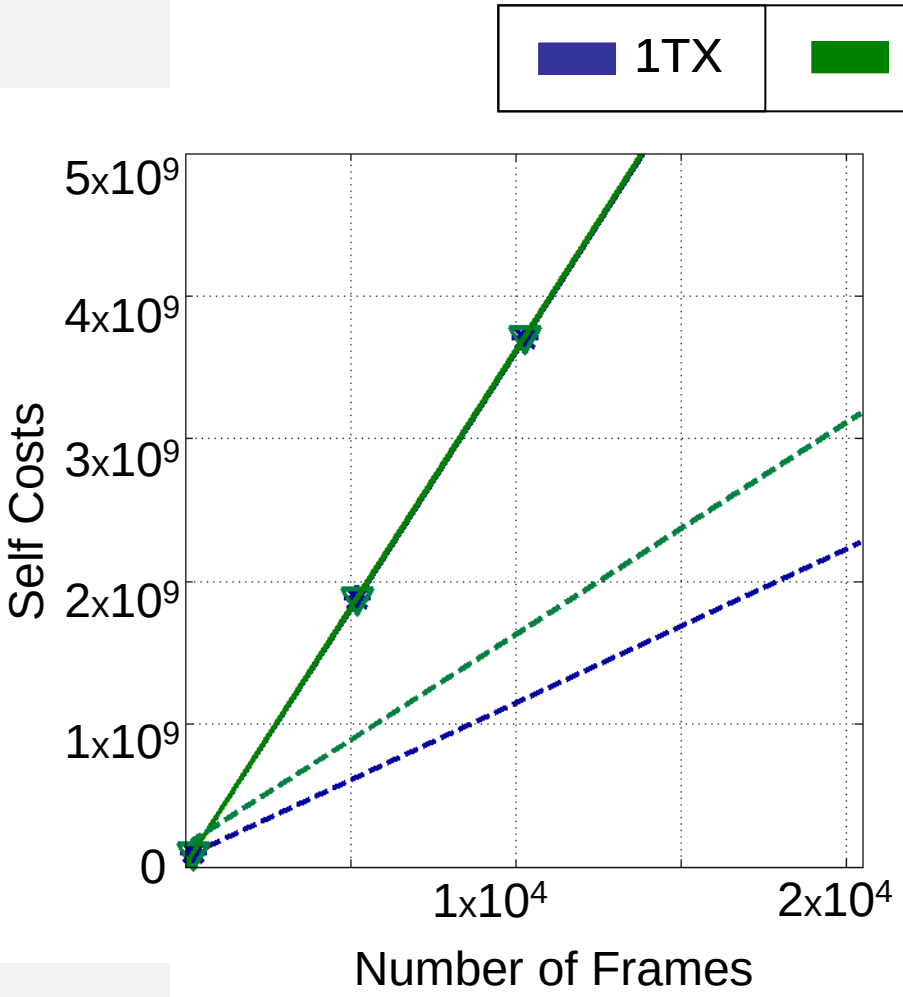
Profiling Results Comparison

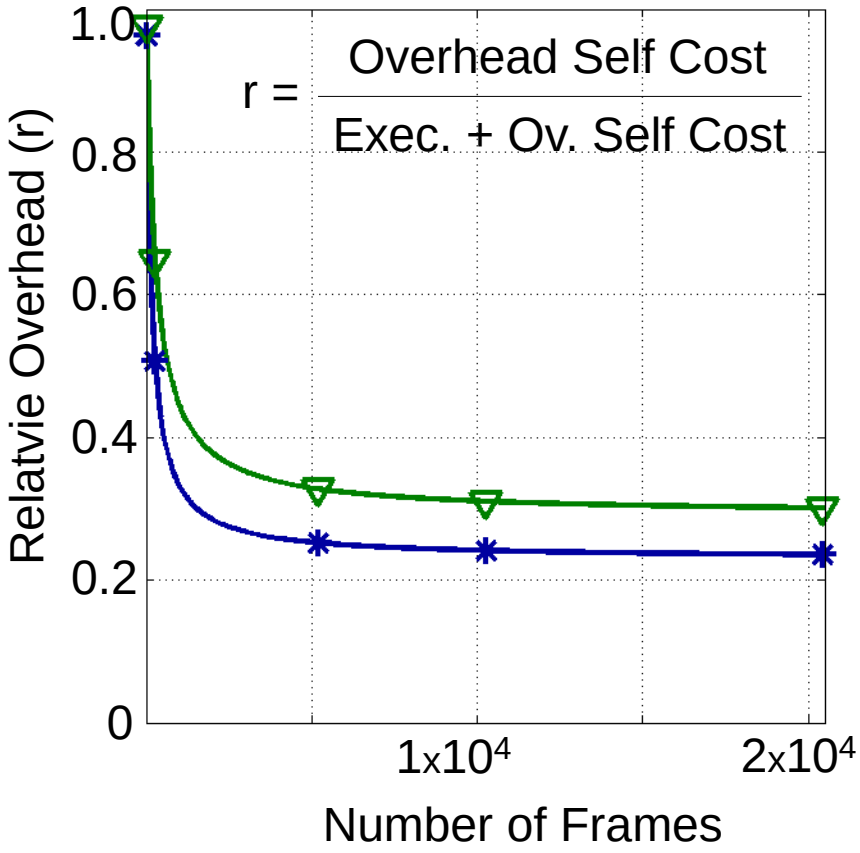




Note, Self Costs indicates the number of instructions/data accesses

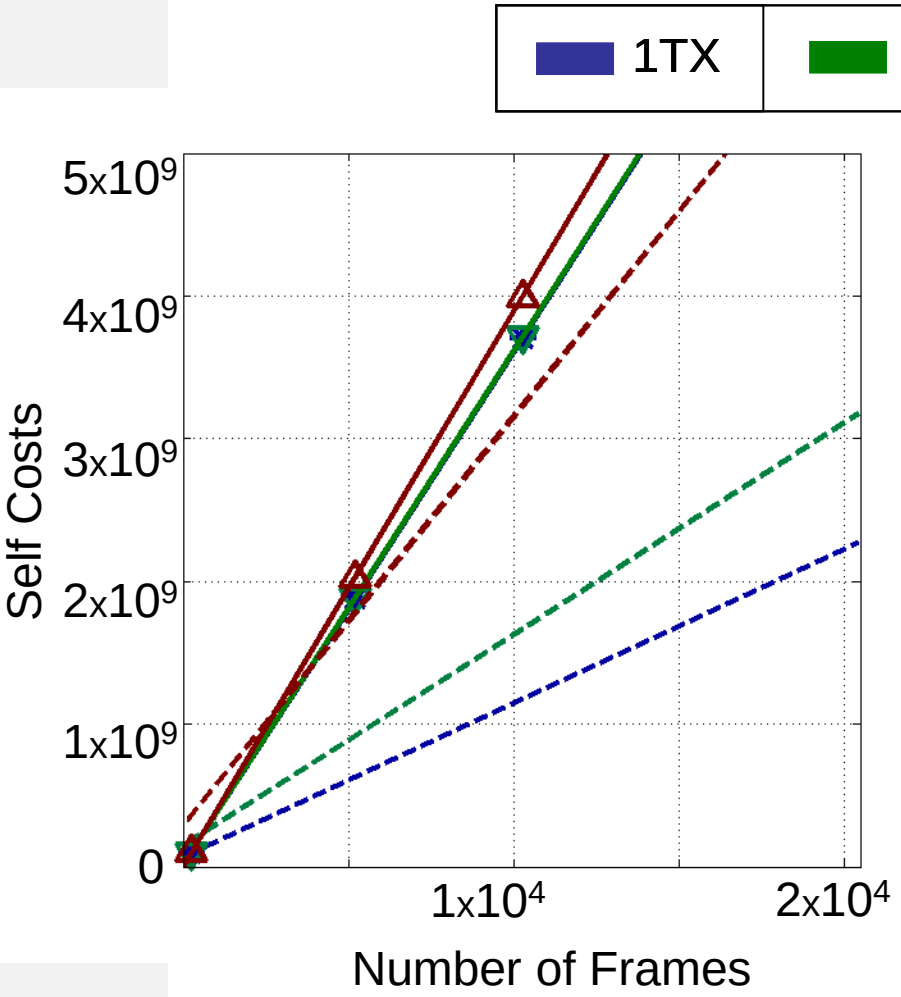
Profiling Results Comparison

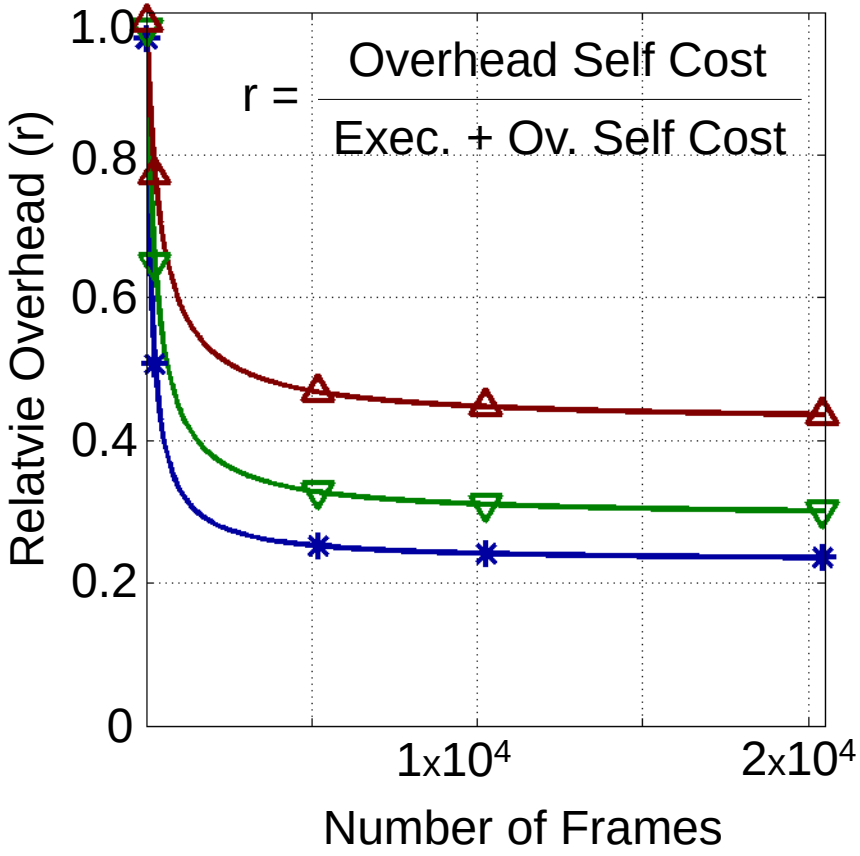




Note, Self Costs indicates the number of instructions/data accesses

Profiling Results Comparison



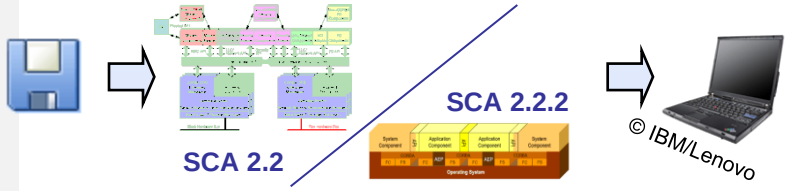


Note, Self Costs indicates the number of instructions/data accesses

Motivation

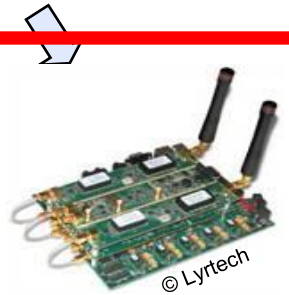
- Work flow

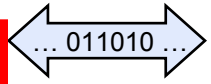
1. SCA-based implementation of STANAG 4285 waveform



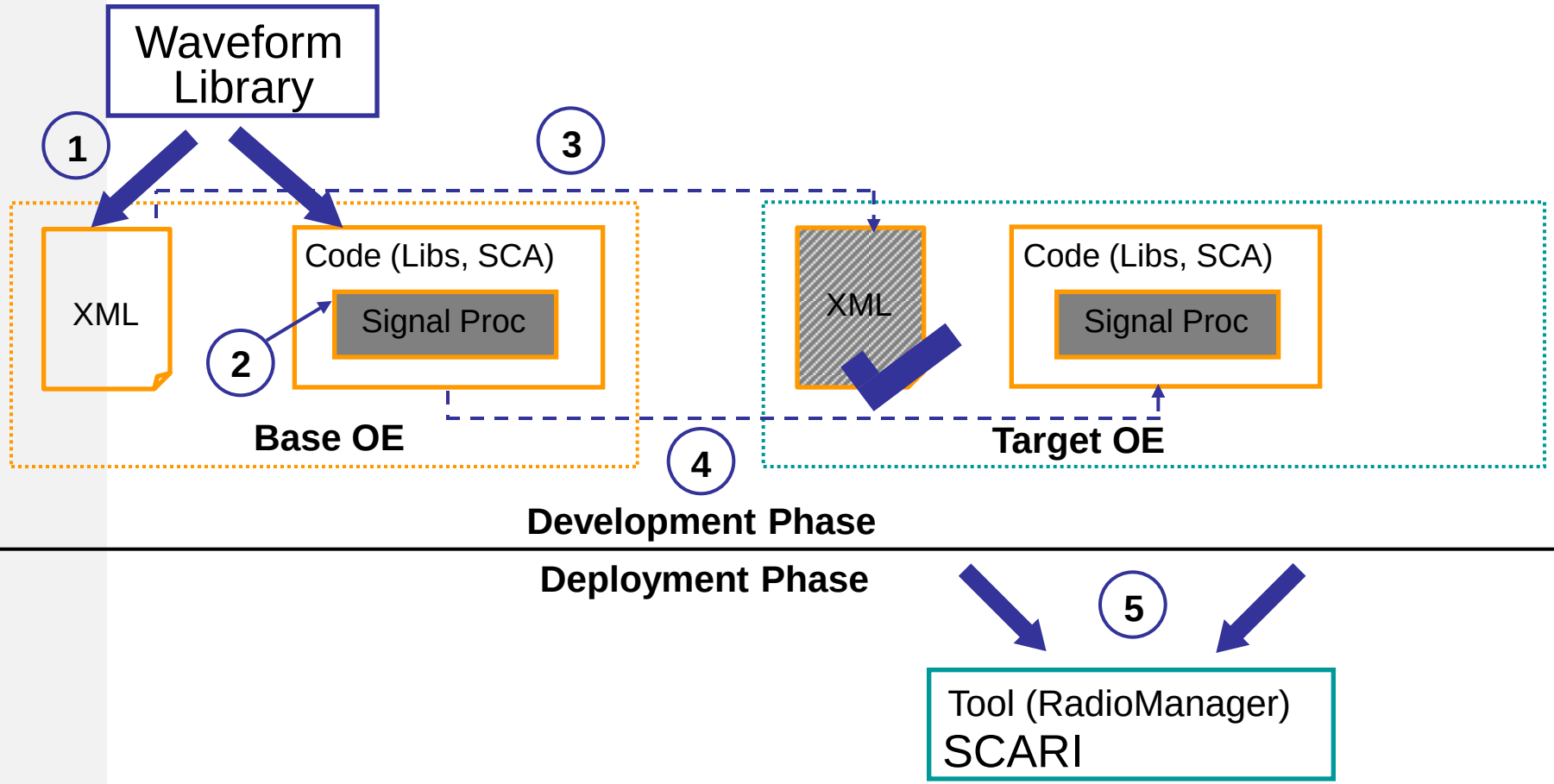
3. demonstrate interoperability between the different implementations

2. demonstrate portability onto national SDR platforms

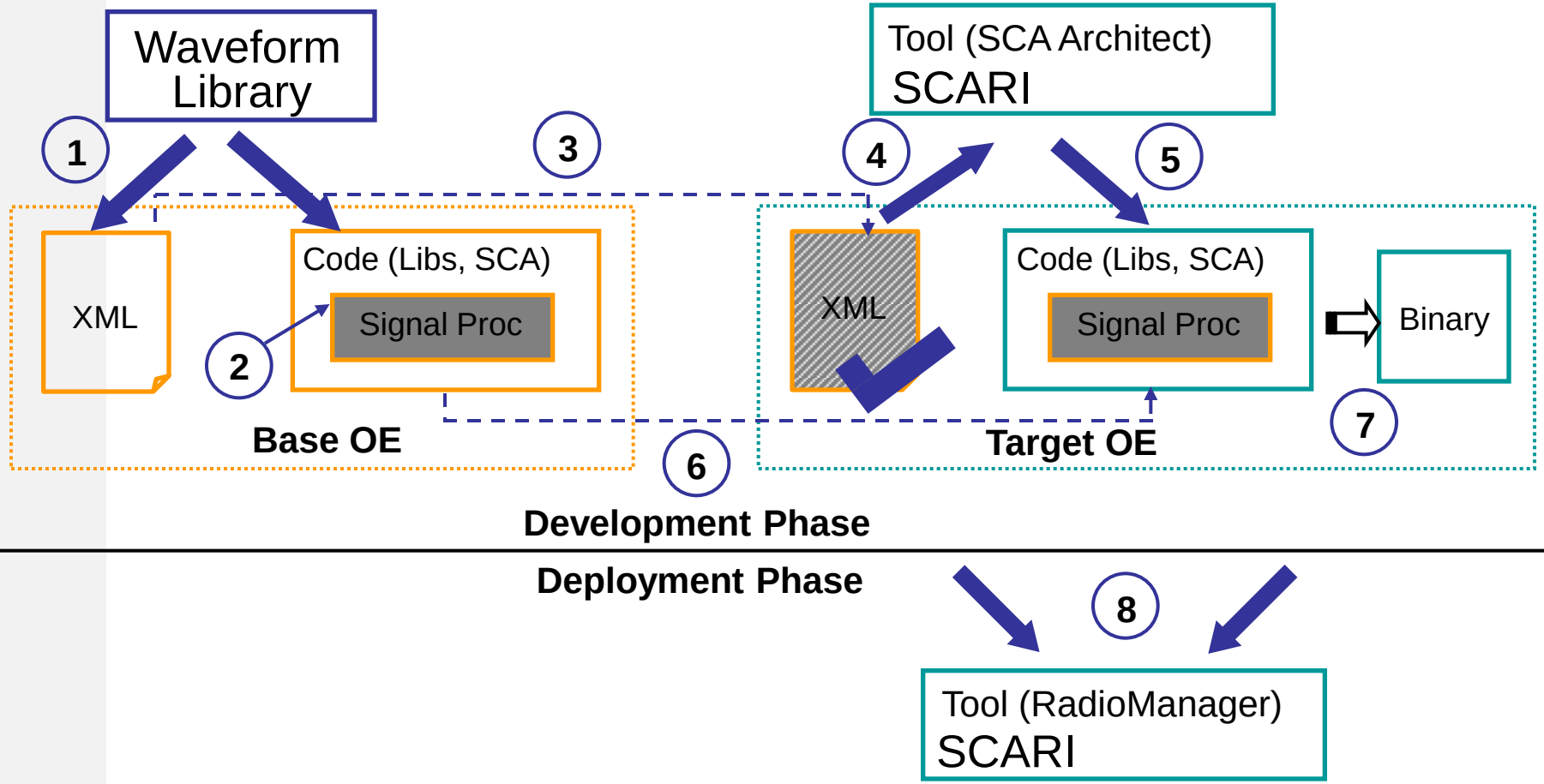




Portability Approach (ideal case)

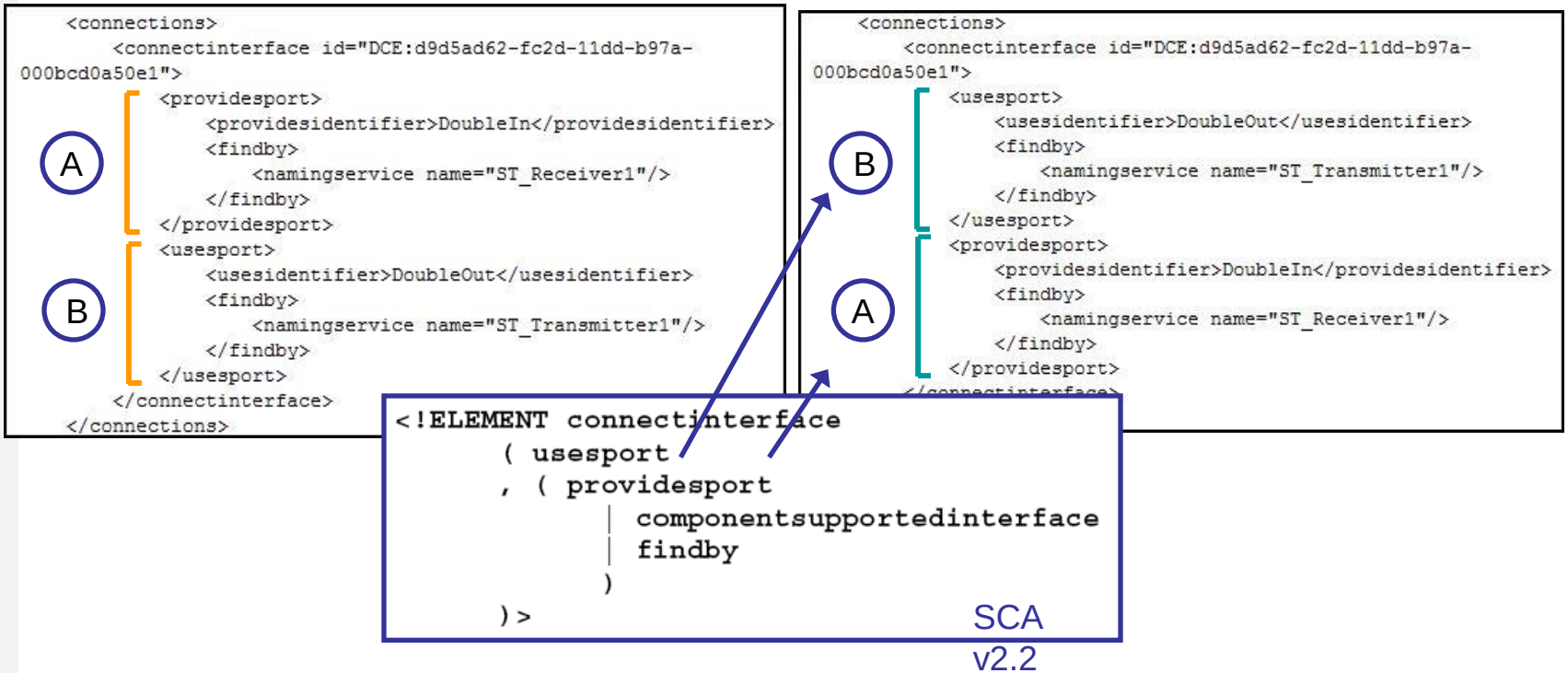


Portability Approach (practical case)



Manual Changes in Generated Code

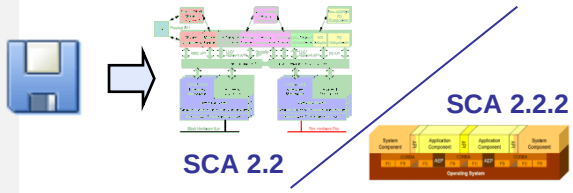
- Example: Using Code generated by OSSIE in SCARI++
  - Domain Profiles: Software Assembly Descriptor (SAD)

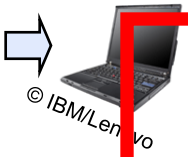


Motivation

- Work flow

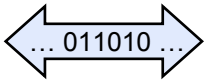
1. SCA-based implementation of STANAG 4285 waveform





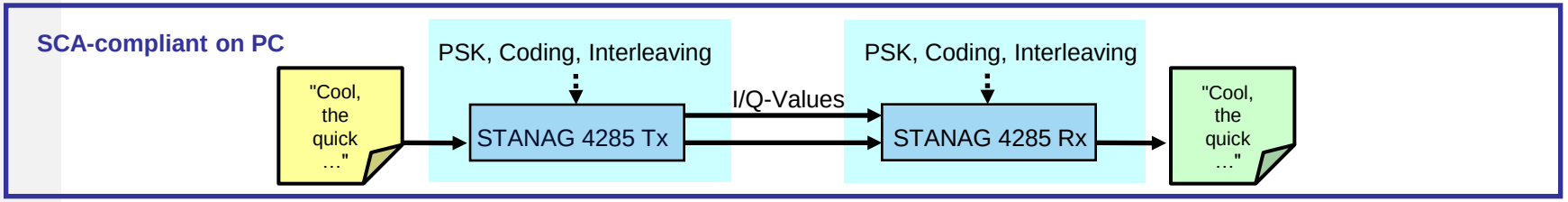
2. demonstrate portability onto national SDR platforms

3. demonstrate interoperability between the different implementations



Demonstrating Interoperability

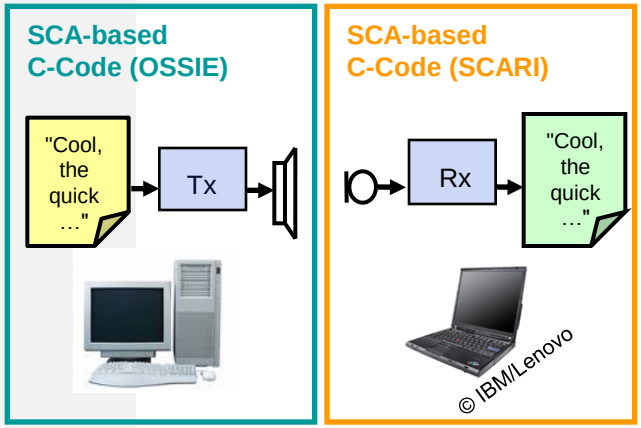
- Levels of Interoperability
  - Baseband I/Q-values (inphase / quadrature)
    e.g., recorded in data files
  - IF-signal (intermediate frequency) → *here: in the audio spectrum*
    e.g., recorded in data files, wired, or wireless
  - *RF-signal (radio frequency)* → *here: out of scope*
    e.g., wired or wireless

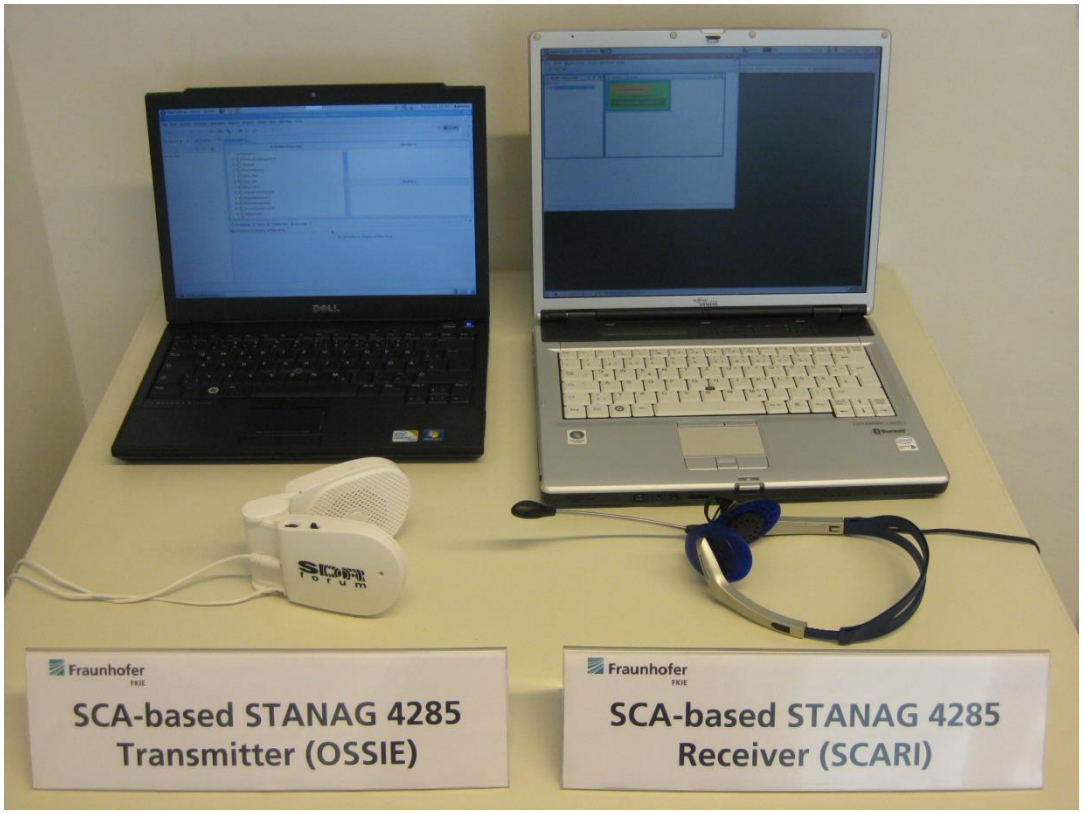


Portability & Interoperability

	Hardware	Tools	GPP/DSP/ FPGA	SCACore Framew.	(RT)OS	CORBA
GE	PC	SCARI++	1 / 0 / 0	SCARI	Linux	TAO ORB
GE	PC	OSSIE	1 / 0 / 0	OSSIE	Linux	omniORB
NO	PC + USRP	OSSIE	1 / 0 / 0	OSSIE	Linux	omniORB
NO	SDR-2000	Zeligsoft	1 / 2 / 1	Harris	WinXP	TAO ORB
TU	PC	Zeligsoft	1 / 0 / 0	TURCOS	Linux	TAO ORB
TU	PPC	Zeligsoft	1 / 0 / 0	TURCOS	Linux	TAO ORB
TU	PPC	Zeligsoft	1 / 0 / 0	TURCOS	VxWorks	ORBExpress
TU	PPC	SCARI	1 / 0 / 0	SCARI GT	Integrity	ORBExpress

Demonstration





Example: „412“ – Q-PSK, rate ½ FEC, long Interleaver (1.2 kbit/s)

Some Lessons Learned & Conclusions

The focus on technical expertise and the possibility of sharing experiences in an international government, university and industry cooperation is unique and of value to its members

- Implementation
  - implementations of different granularity level have been realized
  - → efficient distribution of signal processing functionality
  - → avoid running highly granular application on same processing element
  - → trade-off between computational efficiency and reusability

Some Lessons Learned & Conclusions

- Portability
  - experiments with different implementations of SCA Core Framework, Operating Systems, and Processing elements

  → difference in the code generation of available SCA development tools

  → separation of signal processing code (e.g. Library approach)

- Interoperability
  - over-the-air communication via audio device (intermediate frequency)

  → little impact of SCA implementation on waveform applications

Questions?

Many Thanks for

Your Attention ...

Backup Slides (Properties Descriptor)

- Replacing elements based on SCA Schema

```
<!ELEMENT simple
        ( description?
        , value?
        , units?
        , range?
        , enumerations?
        , kind*
        , action?
        )>
```

SCA v2.2

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "dtd/properties.2.2.dtd">
<!--Created with OSSIE WaveDev-->
<!--Powered by Python-->
<properties>
    <simple id="DCE:dd57f964-fc2c-11dd-a97d-000bcd0a50e1"
mode="readonly" name="Psk" type="short">
        <value>2</value>
        <description>
        </description>
        <kind kindtype="configure"/>
    </simple>
</properties>
```

```
<?xml version="1.0" en
<!DOCTYPE properties S
<!--Created with OSSIE
<!--Powered by Python-
<properties>
    <simple id="DCE:dd57f964-fc2c-11dd-a97d-000bcd0a50e1"
mode="readonly" name="Psk" type="short">
        <description>
        </description>
        <value>2</value>
        <kind kindtype="configure"/>
    </simple>
</properties>
```

Backup Slides (Software Component Descriptor)

- Additional Interfaces

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE softwarecomponent SYSTEM "dtd/softwarecomponent.2.2.dtd">
<!--Created with OSSIE WaveDev-->
<!--Powered by Python-->
<softwarecomponent>
    <corbaversion>2.2</corbaversion>
    <componentrepid repid="IDL:CF/Resource:1.0"/>
    <componenttype>resource</componenttype>
    <componentfeatures>
        <supportsinterface repid="IDL:CF/Resource:1.0"
supportsname="Resource"/>
        <supportsinterface repid="IDL:CF/LifeCycle:1.0"
supportsname="LifeCycle"/>
        <supportsinterface repid="IDL:CF/PortSupplier:1.0"
supportsname="PortSupplier"/>
        <supportsinterface repid="IDL:CF/PropertySet:1.0"
supportsname="PropertySet"/>
        <supportsinterface repid="IDL:CF/TestableObject:1.0"
supportsname="TestableObject"/>
        <ports>
          <uses repid="IDL:PushPorts/ShortSeqConsumer:1.0"
usesname="ShortOut">
                <porttype type="data"/>
            </uses>
        </ports>
    </componentfeatures>
    <interfaces>
        <interface name="Resource" repid="IDL:CF/Resource:1.0">
            <inheritsinterface repid="IDL:CF/LifeCycle:1.0"/>
            <inheritsinterface repid="IDL:CF/PortSupplier:1.0"/>
            <inheritsinterface repid="IDL:CF/PropertySet:1.0"/>
            <inheritsinterface repid="IDL:CF/TestableObject:1.0"/>
        </interface>
        <interface name="ShortSeqConsumer"
repid="IDL:PushPorts/ShortSeqConsumer:1.0"/>
    </interfaces>
</softwarecomponent>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE softwarecomponent SYSTEM "dtd/softwarecomponent.2.2.dtd">
<!--Created with OSSIE WaveDev-->
<!--Powered by Python-->
<softwarecomponent>
    <corbaversion>2.2</corbaversion>
    <componentrepid repid="IDL:CF/Resource:1.0"/>
    <componenttype>resource</componenttype>
    <componentfeatures>
        <supportsinterface repid="IDL:CF/Resource:1.0"
supportsname="Resource"/>
        <supportsinterface repid="IDL:CF/LifeCycle:1.0"
supportsname="LifeCycle"/>
        <supportsinterface repid="IDL:CF/PortSupplier:1.0"
supportsname="PortSupplier"/>
        <supportsinterface repid="IDL:CF/PropertySet:1.0"
supportsname="PropertySet"/>
        <supportsinterface repid="IDL:CF/TestableObject:1.0"
supportsname="TestableObject"/>
        <ports>
          <uses repid="IDL:PushPorts/Sh
usesname="ShortOut">
                <porttype type="data"/>
            </uses>
        </ports>
    </componentfeatures>
    <interfaces>
        <interface name="Resource" repid="IDL:CF/Resource:1.0">
            <inheritsinterface repid="IDL:CF/LifeCycle:1.0"/>
            <inheritsinterface repid="IDL:CF/PortSupplier:1.0"/>
            <inheritsinterface repid="IDL:CF/PropertySet:1.0"/>
            <inheritsinterface repid="IDL:CF/TestableObject:1.0"/>
        </interface>
        <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"/>
        <interface name="ShortSeqConsumer"
repid="IDL:PushPorts/ShortSeqConsumer:1.0"/>
    </interfaces>
</softwarecomponent>
```

```
<!ELEMENT interfaces
        ( interface+
        )>
```

SCA v2.2

Backup Slides (Software Package Descriptor)

- Replacing elements based on SCA Schema
- Addition of necessary elements

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE softpkg SYSTEM "dtd/softpkg.2.2.dtd">
<!--Created with OSSIE WaveDev-->
<!--Powered by Python-->
<softpkg id="DCE:eda4420a-fc2c-11dd-ac86-000bcd0a50e1"
name="ST_Transmitter" type="sca_compliant">
    <title/>
    <description>
    </description>
    <author>
        <name>OSSIE Project</name>
        <company>Mobile and Portable Radio Research Group</company>
        <webpage>http://www.mprg.org</webpage>
    </author>
    <propertyfile type="PRF">
        <localfile name="ST_Transmitter.prf.xml"/>
    </propertyfile>
    <descriptor>
        <localfile name="ST_Transmitter.scd.xml"/>
    </descriptor>
    <implementation aepcompliance="aep_compliant" id="DCE:edaf01ae-
fc2c-11dd-a37c-000bcd0a50e1">
        <description>Description</description>
        <code type="Executable">
            <localfile name="ST_Transmitter"/>
        </code>
        <processor name="x86"/>
    </implementation>
</softpkg>
```

```
<?xml version="1.0" encoding="
<!DOCTYPE softpkg SYSTEM "dtd/
<!--Created with OSSIE WaveDev
<!--Powered by Python-->
<softpkg id="DCE:eda4420a-fc2c
name="ST_Transmitter" type="sc
    <title/>
    <author>
        <name>OSSIE Project</
        <company>Mobile and Po
        <webpage>http://www.mp
    </author>
    <description>
    </description>
    <propertyfile type="PRF">
        <localfile name="ST_Tr
    </propertyfile>
    <descriptor>
        <localfile name="ST_Transmitter.scd.xml"/>
    </descriptor>
    <implementation aepcompliance="aep_compliant" id="DCE:edaf01ae-
fc2c-11dd-a37c-000bcd0a50e1">
        <description>Description</description>
        <code type="Executable">
            <localfile name="ST_Transmitter"/>
        </code>
        <os name="Linux"/>
        <processor name="x86"/>
    </implementation>
</softpkg>
```

```
<!ELEMENT  implementation
        ( description?
        , propertyfile?
        , code
        , compiler?
        , programminglanguage?
        , humanlanguage?
        , runtime?
        , ( os
          | processor
          | dependency
          )+
        , usesdevice*
        )>
```

SCA v2.2