Insights into waveform portability issues of FM3TR waveform

F. Le Roy[a], L. Rakotondrainibe[b], J.- P. Delahaye[c], A. Mansour[a]
[a]LabSTICC, Ensta Bretagne, Brest, France; frederic.le_roy@ensta-bretagne.fr; mancour@ieee.org;
[b]Thales Belgium S.A., Tubize, Belgium ; lahatra.rakotondrainibe@be.thalesgroup.com;
[c]DGA-MI, Bruz, France, jean-philippe.delahaye@intradef.gouv.fr;

ABSTRACT

This work discusses some issues related to the implementation of the Future Multiband Multiwaveform Modular Tactical Radio (FM3TR) waveform on two different SCA platforms with similar hardware but different SCA development and deployment environments. Our experimental results showed that a SCA standardization based on technologies such as CORBA, XML, IDL, is not enough to ensure the portability of the waveform. Indeed, the files generated by certified SCA 2.2.2 environments may often use specific non-standard IDL interface to generate software components. To corroborate our conclusion, some specific examples of SCA components are discussed. Finally, a non-optimal solution called "device" or "black box" software component platform is presented and discussed.

1. INTRODUCTION

In numerous SDR projects the waveform (WF) portability has been investigated [1], as it is considered that portable code can reduce time, efforts and save budget investments. Since the last decade, researchers from all around the world have been involved in the concept of portable codes. In "Wireless Innovation Forum Top 10 Most Wanted Wireless Innovations" [2], porting activity was on the top of the list.

The porting concept was mainly introduced to reinforce the links between a single source of code for WF to target multiple platforms to finally reach the interoperability between various radio systems. In this work, the analysis of executive settings is investigated and several areas related to WF design are considered (such as glue code generation, IDL, CORBA messaging and model of computation (MoC) of "pipelined components").

In this manuscript, two types of SCA [3] component generation are considered. Two generation examples are also presented. We present also the architectures of WF and platforms used in this porting work. Finally, we analyze the porting limitations which were observed in our experiments.

2. OVERVIEW OF PORTABILITY CONCERNS IN SDR

In this part, the different aspects of SDR WF design that impacts its portability are presented. In the context of Software Defined Radio (SDR), the WF design is done on real time embedded systems, so software portability can be considered as a multi-aspect problem. The first aspect is related to the variety of resources used in SDR to execute digital signal processing. In [1], the authors present a survey of various hardware platforms proposed in US military SDR projects with different technical approaches used during the last two decades. In these projects, different Processing Elements (PE) is used such as General Purpose Processor (GPP), Digital Signal Processor (DSP), Field Programmable Gate Array (FPGA), System on Chip, (SoC), etc. By combining different PE technologies in heterogeneous reconfigurable hardware in SDR platforms, recent SDR architectures can make a trade off among the overall performance, the power consumption or the flexibility. The variety of these heterogeneous and distributed architectures implies different repartition of WF functions and code between platforms nodes which limits the WF portability. New technologies such as MPSoCs (Multi Processor System on Chip), multicore, manycore processors, or NoCs (Network on Chip) are coming rapidly so the WF portability can be reduced or even decrease during its long life cycle. Another aspect of SDR platform that impacts software portability is the use of a middleware over the SDR platform hardware. A middleware should help application programming and software portability by providing high level of abstraction and a uniform access over distributed hardware [4]. The support standardized platform services as given by the SCA [1] and the ESSOR Architecture [5], to answer the needs of a large variety of WFs is one of the most important aspect for WF portability. The abstraction and standardization should be done over the entire SDR Platform hardware like proposed by the ESSOR architecture extensions on OE (Operating Environment) Services for DSP and FPGA and additional APIs defining Radio Devices and Radio Services to solve the WF portability challenges.

Some important aspects of portability are coming during the WF development. The SCA Domain Specific Modelling tools that allow the generation of SCA compliant source codes is one of these important portability enabler. In fact, these tools enable WF development methodologies, design guidelines associated with tooling representing the WF software development process.

According to [3], the ESSOR methodology is introduced to define the WF portability. Taking into account the diversity of platform architectures, this methodology allows to develop and to share among several actors a common

waveform. Therefore, the ESSOR methodology relies on BaseWF/TargetWF design approach, where the BaseWF is the portable object. This two-step approach can generate, at the BaseWF level, a software code independent from any target platform supported by a WF PIM modeling language profile. According to [5], the ESSOR methodology for portability is a generic methodology elaborated to design and validate the BaseWF, and relying on the ESSOR Architecture.

The different kind of PE implies to deal with the different programming approaches and different programing languages such as C/C++ for GPP and DSP, VHDL for FPGA. This aspect that limits waveform portability is also discussed in [1].

The rest of the paper presents detailed insights of waveform portability, especially discussing the SCA component design with related design tools and some platform aspects in regard with a porting experience of a waveform.

3. SCA COMPONENT GENERATION

The main objective of the SCA specification is to define the Operating Environment (OE) in a software radio terminal. This OE defines a set of software interfaces that forms the SCA v2.2.2 Core Framework (CF) and other software architectures elements such as the Application Environment Profile (AEP). The SCA v2.2.2 also relies on technological choices such as XML Language for the Domain Profile, Object Oriented technologies, Design Patterns and UML Language.

The CF of the SCA specification is mainly defined by its interfaces (API). The CF is responsible to control, to manage, and to deploy the waveform on a SDR platform. In the context of JTR System "a waveform is used to describe the entire set of functions that occurs from the user to the RF output and vice versa". An implementation of a waveform is a list of interconnected SCA components producing services.

The component design is based on meta-model defined within each code generation tool. These meta-models can be very different from on tool to another despite the facts that tools are compliant with SCA coding rules, component definitions, interfaces and XML files of the "DomainProfile".

3.1. SCA component definition

In SCA the concept of component is mainly defined with the IDL used to define the SCA interfaces and the XML used to "*create the SCA Domain Profile elements which identify the capabilities, properties, inter-dependencies, and location of the hardware devices and software components that make up an SCA-compliant system*" [3]. API standards explicitly defines port concept which is required to deploy software components in SDR platforms. The authors [6] showed that SCA components inherit a set of interfaces defined in the Core Framework (CF). To exchange data, the software components communicate using ports of processing services, such as: port Provide or port use. These ports inherit the resource interface of SCA standard and they must implement service package allowing the CF to manage interconnection, configuration, testing and lifecycle of software components.

Each component has a well-defined set of ports specified by two properties:

1. The type of port or the type of produced service: An "input port" or "provide" port can receive requests from component "output port" or use port. An "input port" should wait remote calls. On the other hand, an output port represents the client side that triggers requests to the server side. In the context of waveform datapath, output ports send data, while input ports receive requests.
2. The type of data carried by ports: Each port has a well-defined data type. Relating a port type to a data type is equivalent to the definition of port's interfaces. These interfaces can be standard APIs or custom interfaces. Creating a custom interface in IDL allows the designer to choose for instance the interface name, the associated methods, data types.

3.2. Implementation possibilities

In an SCA development tool chain, the implementation containers of SCA components are generated by a code generator. In our experiment, three concepts of implementation had been studied:

1. At first, a scheme in which SCA component's class specializes the interface of the class "Resource" of the CF.
2. A second scheme makes separation between the functionalities of a SCA component and its ports.
3. The last one consists in distributing the component works on its possible ports.

The three concepts are developed from the study of SCA Domain Specific Modeling tools like "OSSIE", "SCA Architect" or an older one "Zeligsoft CE" v2.4 (ZCE), etc. The codes generated by these three concepts are conforming to SCA specification; however the code portability depends on the implementation choices. The drawback of the first concept is that the waveform functional code or business code is mixed with the platform non-functional code or glue code (SCA code). From the portability point of view, the second concept is better because it separates between functional code and the glue code also called SCA container. However, this separation affects the size of the generated code. The last concept doesn't provide the separation of concerns, it does not respect the concept of encapsulation of software components and it maximizes porting complexity of a waveform.

The choice of software components implementing model is strongly linked to the choice of software component generation tool.
The second choice can promote exchange and understanding within a team of developers using the same chain of tools. However, when the chain is changed, the compatibility of codes is no longer satisfied and the functional code must be manually integrated in the generated component container.

3.3. Example of code generation.

In this section, two generation examples based on OSSIE [7] and ZCE [8] are presented. They use the second concept presented above. Nevertheless, even if they are based on the same concept, implementations can be different.

3.3.1 OSSIE example

The software component generated by the OEF (OSSIE Eclipse Feature) for the interface of the **Figure 1** produces three C++ files: one for the component class declaration, the second and the last one "main.cpp" are required to start the component in a thread of a middleware (eg. omni_thread). In addition to the source files, the tool generates configuration scripts of installation and XML files for the "DomainProfile" of the SCA CF.

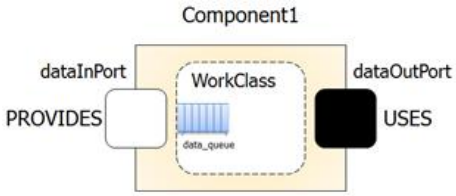

Figure 1 : A minimal SCA component

The class "Component1" generated by OEF inherited the "Resource_impl" class which includes all classes necessary for SCA support, such as for example "getPort", "start", and "stop". In addition to these SCA methods, the component body, file "Component1.cpp" contains several methods such as "Run", "releaseObject", "boot", "query", "configure" and "ProcessData". "ProcessData" should implement the functionality of the component. Component interfaces are also instantiated in this class as illustrated in **Figure 2.**

```
Component1_i::Component1_i(const char *uuid, omni_condition *condition) :
    Resource_impl(uuid), component_running(condition)
{
    dataIn_0 = new standardInterfaces_i::complexShort_p("dataInPort");
    dataOut_0 = new standardInterfaces_i::complexShort_u("dataOutPort");
    start();
}
```

Figure 2 : SCA component generated by OEF

In the OSSIE example, "dataIn_0" (resp. "dataOut_0") ports inherit classes from classes "complexShort_p" (resp. "complexShort_u"). To achieve this task, these two ports use well defined methods "getData" and "pushpacket". These methods interface the component with finite size buffer of type "complexShort". The function code defined in the "ProcessData" method is well isolated from its environment. But the model of computation works as a bounded KPN[9], [14]. In this model, network queues ensure the exchange of messages in asynchronous mode.

3.3.2 ZCE example

The description of ZCE SCA component which fulfills the concept of component container (glue code) that encapsulates the functional code is illustrated in **Figure 3.**

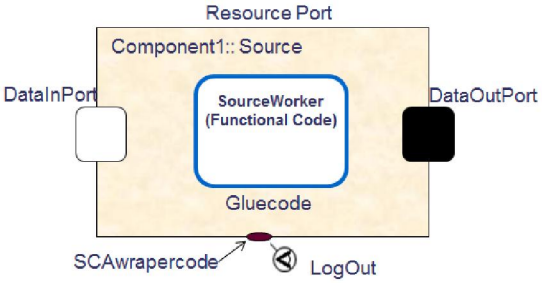

Figure 3 : ZCE software component

ZCE adds proprietary infrastructure and scripts (written by developers) to make possible Component Based Software Design (CBSD). The CBSD offered by Zeligsoft tool addresses the limitations of IDL2.0 based design by establishing architectural choices for component implementations.
ZCE can integrate ORB from different providers and various OS and CF. According to [10], SCA component generated by ZCE satisfies the concept of components in the SCA specification [6]. It is worth mentioning that the architecture of a ZCE component is divided into three parts: the functional one, the SCA connector and the linking code.
When ZCE generates an SCA component three classes "SourceMain", "SourceServant" and "SourceWorker" are produced: "SourceMain" creates an object of class "SourceServant" connected to the CF, "SourceServant" instantiates the "SourceWorker" and classes associated with port components.
The functional code describing functionalities of an SCA component must be completed in the class "SourceWorker" according to coding rules of the waveform designer.
The component of **Figure 1** can't be generated exactly in the same way by two environments. As illustrated in **Figure 7**, ports implement specific classes "SimpleOctePacketSink" and "SimpleOctePacketUses" that are specific to ZCE. Moreover, the functional code of the worker class is executed in a lightweight process (eg. dmtkThread) which is different from the middleware thread.

With these two examples, we showed that the model of computation control can be defined by communication

meta-model used by SCA development tools. Finally, functional code can depend on MOC used by tools generators to connect SCA software components.

4. FM3TR TARGET WF ON TWO HETEROGENEOU FM3TR PLATFORMS

4.1. Porting specification

The objective of this porting work was to evaluate porting effort in porting processes of SCA waveforms. For this task, we chose the reference SCA model FM3TR waveform developed by Calit2 [11]. This waveform implements frequency hopping over both very high frequencies (VHF) and ultra-high frequency (UHF) of the military bands (30-400 MHz). The FM3TR waveform can transport voice and data. It had been deployed by Calit2 in an SDR-4000. Our objective is trying to port it on an SDR-3002 platform that came from the same "Spectrum Signal Processing" branch of "Vecima" society. "Spectrum Signal Processing" becomes one of the leading developers of high-performance, software-reconfigurable SDR platforms.

4.2. Software architectures

The software architecture of the FM3TR waveform developed by Calit2 is illustrated in [11]. The Calit2 demonstrator is composed of two platforms SDR-4000 associated with two computers supporting a GUI which encapsulates sound or Instant Text Messaging (ITM) over TCP/IP. As [11] shows waveform components can be decomposed into software components using a network point of view.

The Calit2 implementation of FM3TR is organized around two kinds of source files: the "SCA components" and the "Devices". The software components are generated using the "SCA Architect" of Nordiasoft tool chain [12].

4.2.1. Devices

- The Net device (data/voice) handles the platform specific transport of voice and data packets between the SDR platform and the TCP/IP Ethernet interface.
- The Modem device is compliant to MHAL modem API. It encapsulates (or extracts) voice and data to MHAL frames. These frames are exchanged with non-CORBA components.

4.2.2. SCA components

- The Continuously Variable Slope Delta modulation **(**CVSD) codec is a voice variable step coding and decoding component.
- The Data Link Control (DLC) segments and reassembles voice and data messages. It implements the classical Automatic Repeat reQuest **(**ARQ**)** network protocols.
- The RS is an SCA resource that encodes outgoing data packets into a R-S block code and decodes received R-S encoded blocks.
- The data Media Access Control (MAC) converts the format between MHAL frames to match the RS encoding format.
- The voice MAC converts the format between voice samples and MHAL frames.

4.3. Platform architecture and mapping

4.3.1. SDR-4000 architecture

The Calit2 demonstrator platform combines the SDR-4000 with a "National Instrument" PXI system for the frequency transposition. This PXI system consists of:

- A card "PXI-5610 Up-converter",
- A card "PXI-5600 Down-converter".

Application or platform components are implemented in the GPP processor card PRO-4600 subsystem SDR-4000. Non CORBA processing base band signal component is implemented in the TMS320C6416 processor PRO-4600 card while the frequency translation component is done using the Virtex-4 of the XMC-3321 card.

4.3.2. SDR-3002 architecture

The architecture of the platform (SDR-3002) used in our project is illustrated in **Figure 4**. The entire system consists of a combination of a SCA subsystem and a transceiver subsystem. The transceiver subsystem is a part of the radio chain that converts the baseband into a radio signal for transmission and converts the radio signal into a baseband reception.

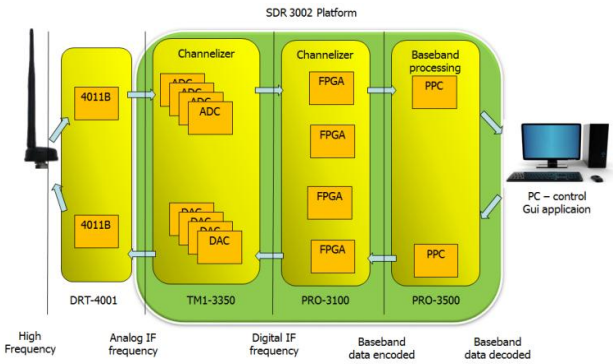

Figure 4 : SDR-3002 platform

The SDR-3002 platform consists of two integrated subsystems in the same cPCI chassis.

The DRT-4001 consists of an amplifier subsystem and a transceiver (transceiver) radio frequency that transposes an intermediate frequency signal up to 3 GHz. The RF signal to be transposed into the DRT-4001 should be centered on an intermediate frequency (IF) of 70 MHz. The RF signal received by the DRT-4001 is transposed to 17.5 MHz

The sub SDR-3002 system consists of:

- The TM1-3350 grabber radio signal (both channels ADC and two DAC channels).

- SBC board: Single Processor Board, x86/Win (Host PC)
- The PRO-3100 card that has four Xilinx Virtex-II, a power PC 405 and an Ethernet interface.
- Ethernet with a GPS receiver.

4.3.3. Mapping and result

The chalenge of SCA FM3TR waveform portability, based on the CALIT 2 SCA waveform is illustrated in [11]. The ZCE model obtained is illustrated in **Figure 8**.

The transfer methodology of application code used in the ZCE model consists of:

1. Searching equivalences between the port types available in ZCE and port types used in the target code.
2. Generating each SCA component of the waveform.
3. Adding Manually the functional code in ZCE components.
4. Creating the SCA model.

We have made the following mapping of the waveform on the SDR-3002 platform.

Table 1 : Waveform mapping on the SDR-3002

Component	*card*	*Target Circuit*	*OS*
cvsd	PRO3500, EPMC8310	P0, PPC7410	VxWork
datamac	PRO3500, EPMC8310	P0, PPC7410	VxWork
fm3trcontroller	SBC	Pentium	Windows
mac	PRO3500, EPMC8310	P0, PPC7410	VxWork
nspr842_duc	PRO3100	XC2V3000 Virtex-II, SAND 0	VxWork
nspr842_ddc	PRO3100	XC2V3000 Virtex-II, SANN 3	VxWork
rs	PRO3500, EPMC8310	P0, PPC7410	VxWork
net	SBC	Pentium	Windows
voiceNet	SBC	Pentium	Windows
modem_device	PRO3500, EPMC8310	P0, PPC7410	VxWork

In this mapping phase, the use of "ZCE" instead of "SCA architect" initially used by Calit2 team made the porting process difficult to be manually managed.

5. OBSERVED PORTING LIMITS

Hereinafter, the limitations observed on development tool, middleware and platforms are discussed.

5.1. Development tool limitations

The observed limitations come from component interfaces and architecture.

5.1.1. Component interfaces

SCA compliant platform comes with its BSP (Board Support Package), its devices and its software development kit (SDK). As indicated by SCA specification, devices and component interfaces may be abstracted by additional specific interfaces that warranty independence of software waveform to platform services. However, BSP and SDK libraries called by SCA tools in generation process of software component can use specific IDL which is not defined in SCA CF interface. In next example; three IDL interfaces generated by tree different SCA development tools are provided.

```
interface IoPacket {
  Oneway void pushPacket
 (in CF:OctetSequence Payload);};
-------------------------------------------------
interface SimpleOctetPacketSink {
 void pushPacket
 (in NullControl unusedControl,
 in CF:OctetSequence Payload)
 raises (PushPacketFailure);};
-------------------------------------------------
interface OctetStream : PayloadStatus {
  void pushPacket
 (in StreamControlType control,
 in JTRS::OctetSequence payload)
 raises( UnableToComplete );};
```

Figure 5 : IDL definition for different Packet interfaces

Figure 5 shows that for similar service of data exchange different interface definitions with some differences in behavior are used and are supported by Platforms. It represents an additional porting effort to adapt from one to another and sometimes difficult to be realized. However this porting can be achieved by importing specific libraries from the first tool/platform to the second one or by redesigning the waveform according to fit this specific interface. This experience shows that the use of different IDL interface definitions between different SCA platforms limits the portability event if tool chains help to perform the required transformation.

5.1.2. Component architecture

The SCA specifies that components inherit the "Resource" class from the SCA CF. A component must implement "uses" and "Provides" ports (see **Figure 1**). However, SCA specification doesn't specify details about implementation. Therefore, the designer can freely implement components.

In the case of "ZCE" tool, code of an instance of a "worker" class runs functional code i.e. a part of a component waveform. This approach separates the structural from the functional parts of a software component. Indeed, the "servant" class implements "Provide" ports that realize interfaces of the processing task (CF::Resource). This separation of concerns is at the expense of code expansion.

Another software design approach uses interface by encoding method. According to our third concept find in subsection 3.2, functionalities are embedded in implementations of "class Port". The major drawback of this approach is the loss of functional code visibility.

In our study, we distinguished between two types of SCA component implementations. The first follows the methodology CBSD (Component-Based Software Development) while the second uses the customer separation / server provided by CORBA 2.x component. The first approach improves the portability; but the designer is free to define the implementation because the SCA standard does not impose any constraint on the implementation else than the use of CORBA.

5.2. CORBA and MOC limitations

SCA waveforms are made from a blend of software components (application components, devices API and controllers). This combination of software components executes usually on target in pipelined manner. SCA 2.2.2 relies on CORBA; data transported by the CORBA bus provides two types of messages: "One-way messaging" and the "two-way messaging". The authors of [13] describe the problem of "pipeline" vacuum related to the use of "two-way messaging". They also describe how "one-way messaging" can be used to limit the impact of empty pipeline on throughput and processing latency. "One-way messaging" is usually considered as a better approach to increase processing's rate. Finally, solutions such as flow control mechanism for "one-way messaging" and "threads" using "two-way messaging" are proposed to address drawbacks.

According to the middleware used by SCA CF, (e.g. TAO or omniORB, etc.) ORB settings act on the waveform portability. Indeed this action changes the model of computation (MOC) [14] of component message exchanged in waveform applications.

5.3. Platform limitations

Processing boards inside SDR platform usually have fast specific link that can be used to bypass the CORBA bus. For example, "FlexFabric" of **Figure 6** are used by Spectrum Signal systems for high-speed communication between two processing resources of the SDR platform. As illustrated in this figure, connection between ports of the two components is associated with the use of an abstract port called "DeviceThatLoadedThisComponentRef". For the SDR-3002 platform, the use of this port in a ZCE model refers explicitly "FlexFabric" link in the model. In this configuration, model and after the source code becomes platform dependent and it is not portable. In addition, this type of connection modifies the computation model of waveform. Indeed, they can be configured in point to point blocking and non-blocking channel. Using this type of connection limits waveform portability, because it is specific to the platform and it modifies the scheduling of the execution or the computation model.

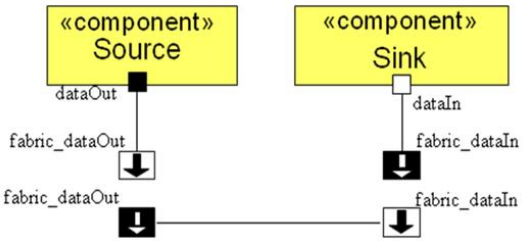

Figure 6 : Fast communication use in ZCE

6. CONCLUSION

In this manuscript, we investigate the portability of FM3TR waveform on SDR-3002. Initially, Calit2 implemented FM3TR an SDR-4000 which is similar to our target platform SDR-3002. Even though the two platforms are similar, the portability of the code is not an easy task. In fact, our experimental works showed that generated source codes depend on software development kit (SDK), CORBA ORB, and OS for the implementation. We have to say again that the execution model is not defined in the SCA specification. This model can be affected by the implementation of software components and the setting of the ORB. Accordingly, this affects the waveform code portability by creating dependencies on the platform.

Our experimental works demonstrate that the SCA development tool chain (such as "Zeligsoft CE", "Spectra CX" or "SCA Architect") improves the development of a SCA waveform. However, the software configurations are difficult (e.g.: dependency management, settings ...) code portability is then partial between tool chain elements and portability at the model level is also poor.

Even if SCA specification enforces the uses of a large sets of Interfaces (mainly related to CF), the use of additional APIs (Radio Devices and Radio Services) that should be standardized to cover all the waveform needs on the overall SDR platform. Therefore, there are still difficulties to port a waveform on COTS platform because these products don't provide a support for additional standardized APIs.

Finally, we can notice the facts that selecting an appropriate meta-model can help us to adapt the code to any specific platform. This approach can be carrying on by using MDE approach which can better manage the development and validation of SCA models.

6. REFERENCES

[1] L. Goeller and D. Tate, "A Technical Review of Software Defined Radios: Vision, Reality, and Current Status", *proceeding of the IEEE Military Communications Conf. 2014.*

[2] "Wireless Innovation Forum Top 10 Most Wanted Wireless Innovations", *Document WINNF-11-P-0014*, Version V3.0.0 24 September 2013.

[3] Software Communications Architecture Specification, JTRS Standards, v2.2.2, may 2006

[4] I. Gomez Miguelez, *et al.* "ALOE: an open-source SDR execution environment with cognitive computing resource management capabilities", *IEEE communications magazine*, Vol. 49, num. 9, September 2011, p. 76-83

[5] C. Serra *et al*., "ESSOR Architecture – Motivation and Overview", *Proceeding of the SDR'10 Conf. 2010.*

[6] Joint Program Executive Office, "Software Communications Architecture Specification", version 2.2.2, 2006.

[7] E. Paone, "Open-Source SCA Implementation-Embedded and Software Architecture - OSSIE SCA Waveform Development", *Master Report*, KTH, 2010.

[8] M. Hermeling, J. Hogg and F. Bordeleau, "Developing SCA Compliant Systems", *Zeligsoft White paper*, 2005.

[9] G. Kahn, "The Semantics of a Simple Language for Parallel Programming*", Proceedings of IFIP Congress*, 1974.

[10] M. Hermeling, "Code Generation for SCA Components", Zeligsoft White Paper, 2005.

[11] P. Johansson, Z. Cao and W. Hodgkiss "Rapid Porting of an SCA-Compliant FM3TR Waveform", *Proceeding of the SDR Forum Tech. Conf. 2009.*

[12] "SCA Architect", Nordiasoft, http://www.nordiasoft.com/#!sca-architect/cxxs

[13] S. Bernier, H. Latour and J. P. Zamora Zapata, "How different messaging semantics can affect SCA applications performances", *International Journal in Analog Integrated Circuits and Signal Processing*, Vol 69, Issue 2-3, December 2011, Pages 227-243.

[14] E. A. Lee and A. Sangiovanni-Vincentelli, "A framework for comparing models of computation*", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol 17 , Issue 12, December 1998,Pages 1217 – 1229.

```
zceComponent1Servant::zceComponent1Servant( const CORBA::ORB_ptr& orb, const sink_params& execParams ZCE_EXC_ENV_ARG)
{

    ZCE_ASSERT_EXCEPTION_VOID;
    orb_ = CORBA::ORB::_duplicate(orb);

    params_ = execParams;

    state_ = UNINITIALIZED;


    worker_ = new zcesComponent1Worker(execParams ZCE_EXC_ENV_PARAM );
    in_dataInPort = new zceSimpleOctetPacketSinkProvidesPort("dataInPort", SINK_DATAIN, worker_ ZCE_EXC_ENV_PARAM );
    out_dataOut = new zceSimpleOctetPacketSinkUsesPort("dataOutPort" ZCE_EXC_ENV_PARAM );
}
```

Figure 7 : SCA component generated by ZCE

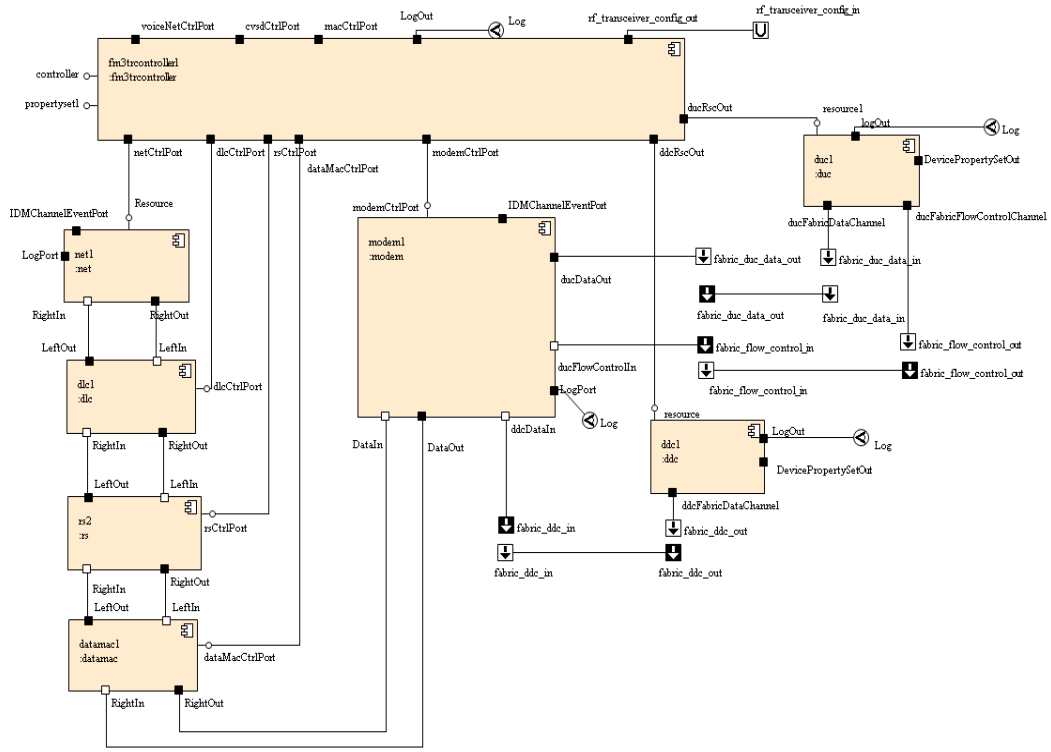

Figure 8 : FM3TR waveform deployment in a SDR-3002