TUBITAK REFERENCE WAVEFORM (TRWF)

Adem Zumbul (TUBITAK-UEKAE, Kocaeli, Turkey, ademz@uekae.tubitak.gov.tr)

ABSTRACT

TUBITAK Reference Waveform (TRWF) is an SCA [1] based open source waveform implementation. It has been developed as part of a Master of Science thesis in Bogazici University in conjunction with TUBITAK-UEKAE. TRWF project aims supporting SDR developers who are involved in SCA waveforms, and it can be downloaded from SDRForum web site [2] with no charge. TRWF has been tested to work with SCARI-Open core framework [3] on Pardus [4] platform which is an open source Linux distribution of TUBITAK. TRWF can run on a single PC or between two connected PCs. In single PC mode, it allows looping voice or data on the waveform components and in double PC mode it allows voice or data communication between them. It also demonstrates how to apply modulation through configurable parameters.

1. DESIGN

TRWF has been designed to demonstrate the applicability of the SCA. Figure 2 shows the model of our waveform. TRWF is composed of eight components. The Zeligsoft [5] model shows these components with large boxes. The lines connecting the boxes represent the CORBA [6] connections among components. For example, ReadData component is connected to ProcessTx component over CORBA which means ReadData will send some data to ProcessTx component by using a predefined CORBA interface. The line representing CORBA connection has two small boxes at the border of components. These represent the port objects. A port object is the CORBA object implementing Port interface. It must be noted that one of the port object is black and the other is white.

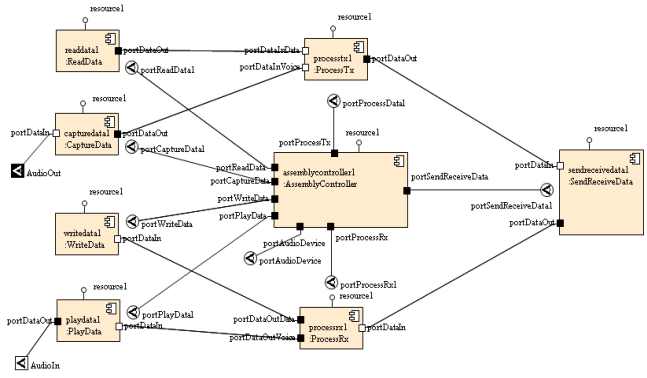

Figure 1: Zeligsoft Model of TRWF.

The color represents the client and the server. White means that this port is the server port and it implements the operation and black means that this is the client object and it wants to call the operation on the server object. Except from the connections between waveform components, there are some floating small rings or boxes that the components have connections with. They represent a component of the core framework, or an object that can be found by using CORBA naming service. For example, PlayData has a connection to the AudioIn floating port. It means that PlayData waveform component, connects to the AudioIn port which is located in AudioDevice of the core framework. PlayData uses that port to play the incoming data by using the device of the core framework.

Each waveform component has a corresponding executable file and some configuration XML files. XML files mainly describe the following:

- **Connections:** Connections are defined in XML files and they describe "Which component talks to which component?" Modifying XML files can change the connections among different components. As far as two components have compatible interfaces, they can be connected to each other. Therefore, it becomes easy to create new applications by connecting different components. The connections are established by the core framework when installing the waveform.
- **Dependencies:** It defines the requirements to run the executable file of the component such as the operating system that the executable file is compiled for, processor type to run this component, memory requirements and so on.
- **Configurations:** Configurations defines reconfigurable properties of each component. Each waveform component is initially configured with the value written in configuration XML file. The configurations can be changed by using PropertySet interface over the CORBA at run time.

The functionalities of each TRWF component are summarized below:

- **ReadData:** ReadData components is responsible to read the input data file as a sequence of bytes and send them to the ProcessTx component for further processing. The number of bytes to be read is defined as a configuration parameter and it is currently set to 8192 bytes at each step.
- **CaptureData:** CaptureData component is the input source component of the waveform when running in voice mode. It is connected to the AudioDevice component of the core framework. Once it is started, the AudioDevice of the core framework captures data from physical audio device and sends the voice as a sequence of integers to the CaptureData component. It must be noted that CaptureData does not deal with the physical hardware and the operating system to capture voice. The layered architectural design, lets the application to be independent of the rest of the system as far as the CORBA connections are established. CaptureData component sends the captured voice to the ProcessTx component for next operations.
- **WriteData:** As the name refers, this component writes the given data array to a file. This component extracts the ID of the sender from incoming packets and creates a file with the name of sender. It supports both text and binary files.
- **PlayData:** This component is responsible to play the given voice data. As in the case of CaptureData component, PlayData does not directly connect to the physical audio device of the radio, but instead uses AudioDevice component of core framework to play the given sound. AudioDevice is responsible from the capacity values of physical audio device and if the capacity is full, it does not allow more connections. In addition, it provides a central initialization and configuration mechanism for sound card, therefore multiple initializations are prevented and once the AudioDevice configuration is changed all the radio is affected.
- **ProcessTx:** ProcessTx component prepares the packet to be sent over network. It appends a header to the message. Figure 3 shows the content of a packet between the waveforms. After packetizing it sends the packets to the SendReceiveData component.
- **ProcessRx:** ProcessRx component unpacks the packets and extracts the header and data from them. It also reads the header and analyzes the incoming data. If it is a voice packet, this component forwards the data part of the packet to PlayData component, and if the incoming packet contains binary or text file data than the incoming packet is sent to WriteData component to be written to the disk.
- **SendReceiveData:** This component has two responsibilities: send packets coming from ProcessTx component to the RFDevice and receive packets from RFDevice and send them to ProcessRx component.
- **AssemblyController:** AssemblyController is the manager component of the waveform and it controls the assembly of the waveform component. It is responsible to talk to the core framework and forward the incoming commands or configurations to other components. Each waveform should have a predefined AssemblyController component which is specified in the XML configuration files of the waveform. Once the core framework reads the configuration files, then it talks to AssemblyController component for any managing operations. For example, in order to start or stop the waveform, core framework calls start or stop method of the AssemblyController and AssemblyController calls start and stop on the rest of the components. The main idea behind this architecture is to make any waveform application transparent to the underlying core framework. Since core frameworks know that there is an AssemblyController component in the waveform and it can use that component to manage that waveform regardless of the internal complexity of it.

These components build up the waveform and once the waveform is started they begin to transfer packets to each other along CORBA connections. Figure 2 shows an example scenario where two SDRs communicate in our proposed design.

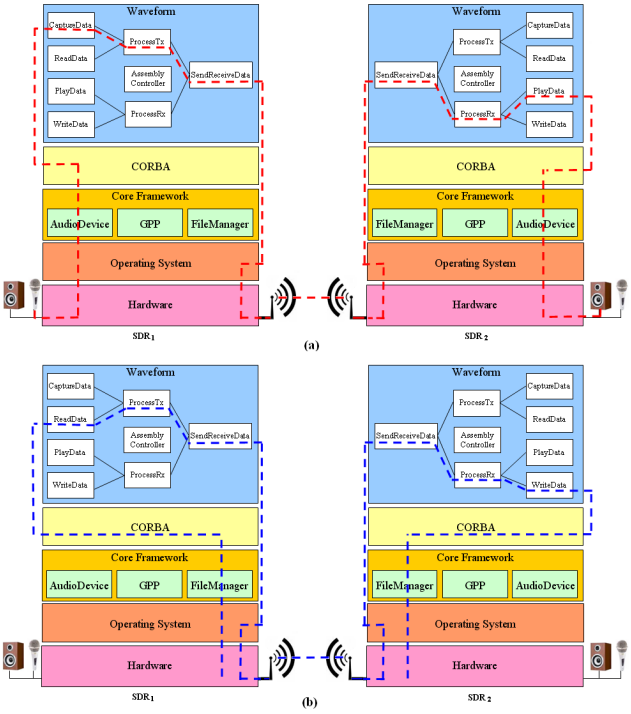

Figure 2: Example scenario where two TRWF installed SDRs communicate (a) Path of the packets in voice mode (b) Path of the packets in data mode.

It also shows the layers of our design and their interaction with each other. Red dashes represent the path of the packets in voice mode, and similarly blue dashes show the path in data mode. SDRs in this layout are connected each other. In voice mode, SDR_1 captures audio from microphone and sends it to SDR_2 and in data mode SDR_1 reads a file from hard disk and sends it to SDR_2. Simultaneously, SDR_2 captures the incoming packets and analyze them. If the packet includes voice data than it sends it to the speaker for being played, similarly if the packet is part of a file it appends the packet to a file of which name is the ID of the sender.

In this layout, actual operating system dependent audio operations are done in AudioDevice component which is part of the core framework layer. Similarly, FileManager component implements operating system dependent file operations. The WF components use core framework components for system dependent operations. Therefore, the same waveform code can work in a different system as long as the target core framework implements the same functionality. In other words, system dependent code goes into the core framework to make waveforms more portable.

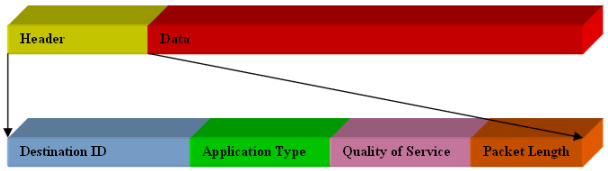

Figure 3: Packet Structure of TRWF.

The packets travelling over the network has two sections: metadata and data. The metadata is the header part and it contains some information about the content of the packet. Figure 3 shows the structure of the packets between waveforms.

The packets between waveforms contain a header part and a data part. The data part contains the actual data which can be voice or file data. The size of the data part is 8 kilo bytes. The header part is appended to data part and contains information about the packet. The header contains the following fields:

- **Destination ID:** Destination ID is the mobile identification number of the mobile terminal that is wanted to be communicated such as 00905351234566. The length of destination ID is 14 bytes which is the length of mobile phone numbers with international area codes.
- **Application Type:** It is a 1 byte value that can take values of 0 or 1 where 0 means the packet contains voice data and 1 means that it contains file data.
- **Quality of Service (QoS):** Quality of Service parameter can take values between 1 and 5. 1 means the data is not critical so that the mobile terminal can send it over slower base stations and it also means that the errors when transferring the data are not too important. On the other side, 5 means the data is too critical and it should be send over the fastest networks with no error even if the communication cost is higher. There is a tradeoff between the QoS parameter and the communication costs. 3 is the optimum value for QoS parameter.
- **Packet Length:** PacketLenght is a 4 byte value that is containing the length of data part of the packet. In our implementation it can take values at most 8 kilo bytes which make 8192 bytes. For files smaller than 8 kB, this parameter shows the size of the file, and also since the waveform sends data packet by packet, the last packet of a file may have a size smaller than 8 Kb, in that case PacketLength parameter shows the size of last packet. For voice packets, the PacketLength is 8 Kb by default.

The size of the fields of the header is larger than it requires. This is because of debugging purposes. For example, ApplicationType field can be 0 or 1 which can be represented by 1 bit, however since we develop a proof-of-concept application and the value of 1 bit is more difficult to debug, we have chosen to represent it with one byte. Since we focus on the design of the architecture of the application, performance issues are not too critical.

2. IMPLEMENTATION

In this section, we present the implementation details by summarizing development stages. We provide information about the tools we have used to design and implement our waveform and mention about the middleware and configuration management technologies in our implementation. The development strategies we present in this section do not only reflect our methodology, but also provides a guideline for the developers who intend to develop waveform applications.

2.1. Development Stages

When we refer waveform we mean a set files. A waveform application basically includes two types of file:

- **Binary files:** These are the actual executable files doing the job of the waveform. A waveform consists of at least one binary file, but usually it includes more than one. Each executable file has a target platform that it can run such as Linux and x86 platform.

- **XML files:** These files are the descriptor files of the binary files. Binary files cannot be used by core framework without its XML descriptors since a binary file does not provide any information such as the platform the executable is build for, external configuration parameters, CORBA interfaces it supports and so on.

We can divide waveform development into three main stages. They are:

- **Component based modeling of the waveform:** Component based modeling allows a very flexible architectural design. Flexible nature of component allows creation of even the waveform of waveforms. There are several design tools for component based modeling of waveforms. Most of them are commercial products. We use Zeligsoft Component Enabler design tool in this project. It is also a proprietary software. It allows visual designing of components. It also allows defining connections, dependencies and configuration parameters of waveform components.
- **Implementing the components:** Implementation means the realization of the design in a programming language according to the target platform. In most of the software development life cycles, the developer notices the weaknesses of the initial design when implementing the waveform. Therefore, it is usually impossible to design a system from scratch at first try. It is an evolving and repetitive process. Implementing waveform components is not an exception to this. Once a design is implemented, it usually requires redesign to maturate. If all the requirements are captured and modeled in the initial design, the following design modifications become smaller, however in worst case the developer may have to throw away the initial design and restart from the beginning if the initial design is too poor.
- **Generating the XML configuration files:** At the last step, the XML configuration files for waveform components are created. Each waveform component has at most three types of XML files. These files can be explained as follows:
  - **Software Packet Descriptor(SPD):** Describes implementation details of the components.
  - **Software Component Descriptor(SCD):** Describes CORBA interfaces of the components.
  - **Property Descriptor(PRF):** Describes the configuration parameters of the components.

There is also a Software Assembly Descriptor (SAD) file for each waveform to describe deployment characteristics and connectivity of application components. Core framework initially reads SAD file when installing the waveform. SAD file includes paths to other configuration files for each component so that the core framework can locate them.

Preparing these XML files manually is too difficult and error-prone process. We have used Zeligsoft Component Enabler to create these files according to our waveform model, however manual modification of the created files is sometimes required in case of errors when running the waveform. In addition, it may be necessary to change initial configuration parameters defined in XML files with a text editor.

3. EVALUATIONS

In this section, the design of TRWF and its layers is evaluated. We summarize performance and static code analysis of our waveform.

3.1. Static Code Analysis

Table 1 shows the files of the TRWF and their properties.

Table 1: TRWF File Analysis.

Component	Number of Files	Size (kB)
Source file (cpp)	37	461
Header file (h)	53	225
Binary file	8	17844
XML	22	41
IDL	3	55
DTD	8	15
Zeligsoft model	1	310
Waveform (All files)	38	17485
Waveform (All files in jar)	1	3584

As shown in the table, TRWF is implemented in C++ programming language. It is composed of 38 files. These files contain executables and configuration files for the waveform components. SCARI-Open core framework installs the waveforms in form of a jar file. The jar extension, should not be mixed with java executables. It is nothing more than a zip file with a jar extension. It is generated by compressing 38 waveform components to a zip file and changing the extension to jar. It is advantageous to follow this methodology because of the following reasons:

- The total size of the waveform is reduced which is useful to distribute it.
- Jar file acts as a container which keeps every thing together.
- Number of waveforms that can be installed to limited memory devices is increased.

Table 2: TRWF Component Analysis.

Component	FnC	WrC	ToC	FoT	WoT	FLOC	WLOC	TLOC
ReadData	27985	51745	79730	35.10	64.90	933	1725	2658
CaptureData	28044	52554	80598	34.79	65.21	935	1752	2687
WriteData	11983	50552	62535	19.16	80.84	399	1685	2085
PlayData	9853	50826	60679	16.24	83.76	328	1694	2023
ProcessTx	12483	55319	67802	18.41	81.59	416	1844	2260
ProcessRx	14308	55247	69555	20.57	79.43	477	1842	2319
SendReceiveData	18241	51638	69879	26.10	73.90	608	1721	2329
AssemblyController	15876	65300	81176	19.56	80.44	529	2177	2706

Table 2 presents some comparison of the component source codes in our implementation. The field of the table can be explained as follows:

- **FnC:** Size of functional code in bytes.
- **WrC:** Size of wrapper code in bytes.
- **ToC:** Size of total code in bytes.
- **FoT:** Percentage of functional code.
- **WoT:** Percentage of wrapper code.
- **FLOC:** Functional lines of code.
- **WLOC:** Wrapper lines of code.
- **TLOC:** Total lines of code.

3.2. Performance Analysis

In this section we present performance analysis of our implementation. The TRWF has been tested on a laptop with the following configuration:

Table 3: Test Configuration.

Component	Configuration
CPU	Intel based Centrino Duo 2 GHz
RAM	2 GB
Hard disk	20 GB Linux partition on a 160 GB and 5400 RPM hard disk
Soundcard	Full duplex sound card
Ethernet	Intel PRO 100 ethernet card
Operating System	Pardus 2008
WF ORB	ACETAO
Core framework	SCARI-Open
CF ORB	Java ORB

Each scenario has been tested several times and the averages are taken. In the first scenario, we have tested timings for each life cycle operation of our PC based SDR platform. Figure 4 shows the life cycle operations and their precedence.

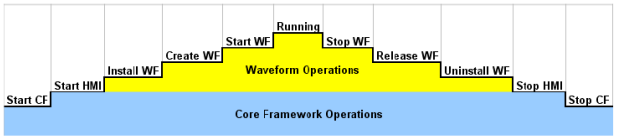

Figure 4: Lifecycle operations.

Life cycle of an SCA compliant SDR platform can be investigated in two categories:

- **Core framework operations:** It includes operations related to core framework and human machine interface (HMI) which is the user interface of the core framework. Separating HMI and CF can be useful to manage SDR by using a different machine over network.
- **Waveform operations:** These are waveform management operations and they are executed by the operator over HMI and these operations cannot be started before core framework takes start.

We have measured the time elapsed at each SDR life cycle operation. The following tables show the performance analysis of TRWF on SCARI-Open core framework.

Table 4: Timings for Lifecycle Operations
(a) Core framework operations (b) Waveform operations.

Component	Time (sec)
Start CF	1.6
Start HMI	2
Stop HMI	0.5
Stop CF	4.9

(a)

Component	Time (sec)
Install WF	3.5
Create WF	12.5
Start WF	0.1
Stop WF	0.1
Release WF	10
Uninstall WF	6.5

(b)

Creating the waveform includes parsing the XML files and deploying, configuring and connecting the waveform components. Therefore it takes the longest time when compared to other operations. On the other hand, starting and stopping the waveform takes very negligible time, it is because it only calls start or stop methods on the components over CORBA.

In the next scenario, we have measured the running performance of our waveform. Implemented waveform has four modes of operation:

- VoiceTx: Transmit voice.
- VoiceRx: Receive voice.
- DataTx: Transmit text or binary data file.
- DataRx: Receive text or binary data file.

In this scenario, we have measured the total time of DataTx and DataRx operations with and without network operations. We have tried to see the effect of core framework and middleware overhead for different file sizes. We have also compared results with the time that is required to copy and paste the same files.

Table 5 shows the results of our experiments in this scenario.

Table 5: Data Transfer Times.

File Size (kB)	Time 1 (sec)	Time 2 (sec)	Time 3 (sec)
10	0.4	0.3	0.02
100	3.5	3.2	0.05
1000	32	30	0.1
2000	63	60	0.2
5000	159	155	0.4
10000	322	318	0.9
20000	640	635	1.7
50000	1590	1580	3.4

In this table we compare 3 cases:

- **Time 1:** Data packets travel through ReadData, ProcessTx, SendReceiveData, RFDevice, SendReceiveData, ProcessRx and WriteData.
- **Time 2:** Data packets travel through ReadData, ProcessTx, SendReceiveData, ProcessRx and WriteData. Only difference with case 1 is that packets do not pass over RFDevice, but instead they loop between waveform components.
- **Time 3:** This is the time to copy and paste the same file.

It can be noticed that, the copy and paste method is significantly faster then other two cases, because processing and sending packets between components over CORBA takes too much time. We also observe a linear ratio between file size and the elapsed time which means that overhead of middleware and core framework is constant for each unit of file.

4. CONCLUSION

In this paper, we summarize design and implementation details of TRWF which is an open source SCA waveform. It demonstrates basic operations that a WF can have, and demonstrates how to apply modulation and streaming of data through configurable parameters. It works with the SCARI-Open core framework for letting everyone to be able to try it freely. All source codes, configuration files, sample inputs and related documentation can be downloaded from SDRForum web site [2].

5. REFERENCES

[1] Joint Tactical Radio System, “Software communications architecture specification-Final”, Version 2.2.2, Space and Naval Warfare System Center, San Diego CA, 15 May 2006.

[2] TRWF Download Page, “http://www.sdrforum.org/pages/documentLibrary/documentLibrary.asp?DocumentSubType=Document%20Library%20-%20Current%20Input%20Documents”, 2009.

[3] SCARI-Open, “http://www.crc.ca/en/html/crc/home/research/satcom/ rars/sdr/products/scari_open/scari_open”, 2009.

[4] Pardus , “http://www.pardus.org.tr/eng/index.html”, 2009.

[5] Zeligsof t, http://www.zeligsoft.com/, 2009.

[6] CORBA home page, “http://www.corba.org/”, 2009.