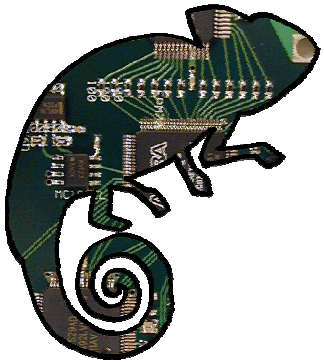

Chameleonic Radio

Technical Memo No. 9

USRP Hardware and Software Description

P. Balister

J. Reed

June 30, 2006

Bradley Dept. of Electrical & Computer Engineering
Virginia Polytechnic Institute & State University
Blacksburg, VA 24061

USRP Hardware and Software Description

Philip Balister and Jeffrey H. Reed
MPRG/Wireless@Virginia Tech
balister@vt.edu reedjh@vt.edu

June 30, 2006

1 Introduction

This technical report describes the Universal Software Radio Peripheral [1] (USRP) and the associated software that provides an interface to the Open Source SCA Implementation::Embedded (OSSIE) framework. OSSIE is an open source implementation of the software communication architecture (SCA). The OSSIE project maintains a website at [2] containing information about the project and the source code for the framework. The SCA was developed by the Department of Defense to provide a common software architecture for the Joint Tactical Radio System (JTRS). OSSIE supports the Universal Software Radio Peripheral (USRP), the USRP provides a low cost development platform for proving Software Defined Radio (SDR) concepts. An overview of the USRP is presented below.

The USRP provides an interface between four high speed analog to digital converters, four high speed digital to analog converters and a USB 2.0 high speed interface. Daughter boards available for the USRP provide a interface from the baseband signals present at the data converters to several useful frequency bands.

The USRP uses two Analog Devices AD9862 Mixed-Signal Front End (MxFE) processors for analog to digital conversion (and the reverse). In addition they provide gain control in the analog path and some signal processing in the digital path.

An Altera Cyclone series FPGA does signal processing for the transmit and receive paths. The receive path has a mixer and a decimation unit and the transmit path contains an interpolater, implemented in the FPGA. Finally a Cypress FX2 interfaces between the FPGA and a USB 2.0 port. The USRP connects to a USB port on the host computer where modulation and demodulation is performed.

The software interface device for the USRP is an SCA compatible component that provides control and data ports to the USRP. There are several key sections of the software; the radio control interface, main program, device object and port implementations.

The radio control interfaces provide a set of control functions to read and write various settings in the USRP. Furthermore, this interface should be useful to control hardware, similar to the USRP, that provide similar RF interface and digitization functions.

The component start up code creates the CORBA servant and registers it with the naming service, and provides some other housekeeping functions. The USRP device object implements the framework control interfaces and the data processing threads for the transmit and receive paths. Finally, the port implementation code contains the software to support the control and data interface ports.

The report is structured as follows; section 2 introduces the USRP and the associated software, section 2.1 describes the USRP hardware, section 2.2 details USRP control interface, section 2.3 discusses the software implementation. Finally Section 3 presents a brief summary.

2 USRP Hardware and Software

The SCA baseband processor uses a USRP to interface the RF hardware to the baseband processing hardware. The USRP was developed as part of the GNU Radio Project. The GNU Radio project is an open source software radio development environment designed operate on PC compatible hardware running (primarily) Linux. More information on the GNU Radio Project is available at [3]. Complete information for the USRP, including schematics, is available from [1]. Below is a brief summary of the USRP.

The USRP provides several functions; digitization of the input signal, digital tuning within the IF band, and sample rate reduction before sending the digitized baseband data to the computing platform via the USB interface. It provides the opposite processing functions for the transmit path. Most the processing performed by the USRP is done in an Altera Cyclone FPGA. An Analog Devices MxFE processor (AD9862) provides some signal processing in the transmit path, and conversion between analog and digital signals for both the transmit and receive paths.

The Software Communications Architecture (SCA) [4] uses the concept of a device that provides a software proxy for physical hardware. This report describes the USRP and the associated device proxy. Some knowledge of the SCA is required to fully understand this document. [5] should be useful for learning the SCA. The next portion of the report provides a high level overview of how components interface with the SCA.

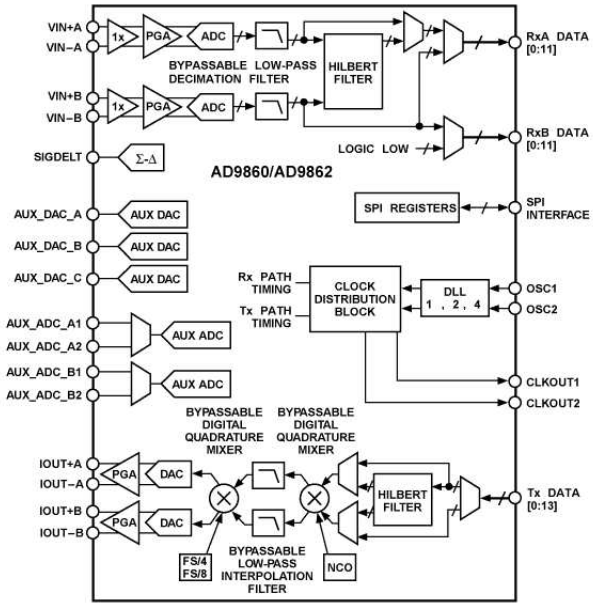

Figure 1: Analog Devices Mixed Signal Front-End Processor (AD9862)[7]

SCA connections are based on the concept of *provides ports* and *uses ports*. A *provides port* implements an interface for use by other components. A *uses port* calls methods implemented by a *provides port*. Data flow may be in either direction, however the *uses port* controls when an operation is performed by the *provides port*. Another way of thinking of this is that a *uses port* is connected to a *provides port* and the *uses port* executes operations defined by the provides port. CORBA [6] provides inter-component communication in a transparent manner between ports.

The interface descriptions are written in the CORBA interface definition language (IDL). IDL provides a language neutral interface description. The specific CORBA implementation used provides an IDL compiler to convert the IDL file into a language specific file.

2.1 Universal Software Radio Peripheral

The USRP is a data acquisition board containing several distinct sections. The analog interface portion contains four analog to digital converters (ADC) and four digital to analog convertors (DAC). The ADC's operate at 64 million samples per second (Msps) and the DAC's operate at 128 Msps. Since the USB bus operates at a maximum rate of 480 million bits per second (Mbps), the FPGA must reduce the sample rate in the receive path and

increase the sample rate in the transmit path to match the sample rates between the high speed data converter and the lower speeds supported by the USB connection.

The AD9862 provides several functions. Each receive section contains four ADC's. Before the ADC's there are programmable gain amplifiers (PGA) available to adjust the input signal level in order to maximize use of the ADC's dynamic range. The transmit path provides an interpolater and upconverter to match the output sample rate to the DAC sample rate and convert the baseband input to a low IF output. There are PGA's after the DAC's. Figure 1 shows the block diagram for the AD9862.

Most of the receive signal processing is performed in the FPGA. First the signal is coupled into the AD9862. This chip contains two channels of ADC's and two channels of DAC's. The clock provided by the USRP drives the ADC's at 64 Msps. If needed the AD9862 may divide this clock by two to reduce the sample rate. This only effects the clock rate of the ADC's, most of the sample rate conversion is done in the FPGA.

After the signal is digitized, the data is sent to the FPGA. The standard FPGA firmware provides two Digital Downconverters (DDC). The FPGA uses a multiplexer to connect the input streams from each of the ADC's to the inputs of the DDC's. This multiplexer allows the USRP to support both real and complex input signals. The DDC's operate as real downconverters using the data from one ADC fed into the real channel or as complex DDC's where the data from one ADC is fed to the real channel and the data from another ADC is fed to the complex channel via the multiplexer. There are some examples showing how to use the multiplexer at [8].

The DDC consists of a numerically controlled oscillator, a digital mixer, and a cascade integrate comb (CIC) filter. These components downconvert the desired channel to baseband (or low IF), reduce the sample rate and provide low pass filtering. For this project the maximum decimation rate available of 128 is used. The signal delivered from the USRP to the signal processing platform has a sample rate of 250 Ksps.

The transmit path for the USRP is similar to the receive path, however there are differences. Since the sample rate the DAC's operate at 128 Msps, an interpolater running on the FPGA increases the sample rate. The AD9862 also provides a further sample rate increase by a factor of four. The transmit portion of the AD9862 provides the mixer and NCO required to set the IF frequency of the transmitted signal, the FPGA performs this function in the receive path.

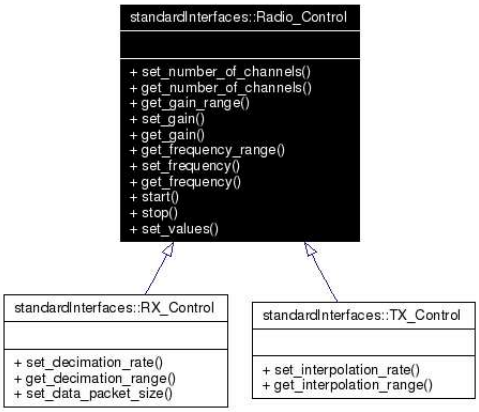

Figure 2: UML diagram for radio control interfaces

2.2 USRP interface device

The SCA uses concept of software proxies called devices to interface the SCA software based system with the hardware. The software proxy supports the start up of the hardware, configuration of various functions and parameters, and supports the data transfer to and from the hardware.

The actual interfaces are designed to support a function, in this case a digital radio front-end, rather than focus on the specific hardware. By using this approach, the hardware may be changed to another piece of hardware, that provides a similar function, without requiring application code changes. Definition of flexible, reusable interfaces is a key element for successful use of the SCA.

The USRP interface device provides ports that allow data transfer to and from the USRP and provide interfaces for setup and control of the transmit and receive sections of the USRP. The control interfaces are implemented as *provides ports*. The ports that receive data for the transmit path of the USRP are also implemented as *provides ports*. Finally, the ports that send received data are implemented as *uses ports*.

Figure 2 shows a Unified Modeling Language (UML) diagram of the transmit and receive control interfaces. Note the majority of the operations are common to transmit and receive, with only a few operations specific to transmit or receive.

Here is the CORBA Interface Definition Language (IDL) for the basic radio control interface.

```
1   interface Radio_Control {
2     /// Set the number of active channels
3     void set_number_of_channels(in unsigned long num);
4     void get_number_of_channels(out unsigned long num);
```

```
5
6       /// Gain functions
7       void get_gain_range(in unsigned long channel, out float gmin,
8           out float gmax, out float gstep);
9       void set_gain(in unsigned long channel, in float gain);
10      void get_gain(in unsigned long channel, out float gain);
11
12      /// Frequency tuning functions
13      void get_frequency_range(in unsigned long channel, out float
            fmin,
14          out float fmax, out float fstep);
15      void set_frequency(in unsigned long channel, in float f);
16      void get_frequency(in unsigned long channel, out float f);
17
18      // Start/Stop Control
19      void start(in unsigned long channel);
20      void stop(in unsigned long channel);
21
22      // Set properties
23      void set_values(in CF::Properties values);
24
25  };
```

The interface methods are grouped into five basic groups, channel control, gain control, frequency control, start/stop control, and a miscellaneous control. These methods apply to the transmit and receive sections of the USRP. They are intended to also apply to the RF front end being developed for the NIJ radio RF front end developed for this project. In the future, this interface should also support any other radio hardware that provides similar functions, such as the ICE-PIC4X [9].

The channel control methods provide an interface for controlling the number of active channels. For the USRP, the maximum number of channels supported depends on the firmware image loaded into the FPGA.

The image currently supported by the USRP device proxy supports up to two channels in the transmit and receive paths. The GNU radio [3] software also provides an alternate image that supports 4 receive channels and no transmit channels. The GNU Radio FPGA bitstreams for the USRP are also available with an open source license as part of the usrp software support library available at [10].

The gain control methods provide an interface to adjust gain settings available in the transmit and receive paths. These gains are dependent on specific hardware implementations. Although specific hardware may contain gain settings in several different places, such as Programmable Gain Amplifiers (PGA), variable attenuators, and other circuits, the radio control interface only provides one control setting. It is up to the software implementing the interface device to determine how to distribute the desired gain throughout the system.

For the USRP, there are adjustable gain settings available in the AD9862 and on some daughterboards. The USRP interface device is responsible for detecting the attached daughterboards and controlling the available gain resources based on inputs from the radio control interface.

The PGA in the AD9862 (both transmit and receive paths) provides a gain range of 0 dB to 20 dB.

The USRP's RFX-400 daughter board has a PGA in the receive path. When the interface device needs to set the receive path gain, the PGA on the daughter board is adjusted first. When it reaches the maximum setting, then PGA on the AD9862 is adjusted.

The PGA gain for the transmit path on the RFX-400 daughter board is set to the maximum available in the AD9862 for proper biasing of the circuitry in the transmit path. Output power control is performed by adjusting the amplitude of the data sent to the USRP.

The frequency control interfaces allow the user to determine the available frequency range, get the current setting and set the operating frequency. How this is done depends on the underlying hardware. The USRP provides some digital down converter implemented in the FPGA. The daughter boards for the USRP may also provide synthesizers and mixers to downconvert the received signal so that it may be digitized by the USRP data converters.

Since the FPGA in the USRP supports a digital mixer and the daughterboards may also provide a mixer, the USRP interface device provides the tunable range of both the daughterboard and the FPGA.

The RFX-400 daughter board provides a local oscillator that tunes from 400MHz to 500 MHz in 6 MHz steps, the USRP interface device sets this oscillator to a frequency below the desired carrier frequency and uses the NCO located in the USRP FPGA to provide fine tuning.

The start and stop methods control the data flow from the radio channel. For the USRP device start and stop control the data being sent from the receiver channels. For the transmit path, the USRP interface device will still accept data on the transmit data ports, however, the data is not sent to the USRP until after the start command is receiver.

The USRP interface device only provides limited data buffering. When the USRP is stopped, or the waveform is sending data to the USRP faster than it is transmitted, the

provides port will not return control to the calling component until there is room for more data. Care should be taken to ensure the upstream components stop generating data when this occurs.

2.3 Software implementation details

The software implementing the USRP interface device is divided into three parts; a main routine that creates the USRP control object and performs the CORBA initialization, a USRP control object that implements to core framework device interface, and a set of classes that implement to communication ports.

The current development version of the USRP interface device software is available on line [11]. To view the software you must create an account, and to email ossie@vt.edu to request your account be added to the access list for subversion source code repository. General OSSIE installation instructions are available at *http://ossie.mprg.org/OSSIE_Installation*. Subversion is version control software [12].

2.3.1 Main Program

The main program creates the CORBA servant that processes the messages received from other components. The program is started with three arguments: the unique identifier for the component, the usage name and the software profile XML file name. These values are read from XML files describing the waveform. Some XML files are described in Appendices A and B. Listing 1 shows the code that performs this and the following functions. The CORBA servant is created in line 12.

Next, on line 13, a CORBA object reference is created from the servant. Lines 12 though 17 show how the object reference is registered with the CORBA naming service. The CORBA naming service allows the object reference to be located by other components in the system.

On line 20, the program starts to handle CORBA messages.

```
1       ossieSupport::ORB *orb = new ossieSupport::ORB;
2
3       char *id = argv[1];
4       char *label = argv[2];
5       char *profile = argv[3];
6
7       USRP_i* usrp_servant;
8       CF::Device_var usrp_var;
9
```

```
10      // Create the USRP device servant and object reference
11
12      usrp_servant = new USRP_i(id, label, profile);
13      usrp_var = usrp_servant->_this();
14
15      string objName = "DomainName1/";
16      objName += label;
17      orb->bind_object_to_name((CORBA::Object_ptr) usrp_var, objName
            .c_str());
18
19      // Start the orb
20      orb->orb->run();
```

Listing 1: main.cpp

2.3.2 USRP device object

The USRP device object is the central data structure for the component. The USRP_i class implements the device interface from the Core Framework. This report will not detail the behavior of the Core Framework interfaces, these will be covered in detail in the component manual and the general behavior is covered in [4].

The software implementing this functions is currently available on the OSSIE project subversion server described above. Due to the length of the software implementing this function, it is not included here.

Currently, the OSSIE team is reviewing design structures for components, such as the component developed for this report. With the current structure, a component developer must be knowledgable in C++, CORBA, multi-threaded programming, signal processing, and object oriented development. The component structuring work focuses on standardizing port implementation to provide an efficient and simple to use design that hides much of the non-signal processing from the component developer. This work will be documented in a future technical report.

The class constructor creates the CORBA servants for the ports and creates object references for each port. Since these ports must be accessible to waveforms that have no knowledge of the specific hardware present in the system, these object references are added to the CORBA naming service.

Beyond implementing the required Core Framework interfaces, the USRP_i class provides two additional functions. The class provides global storage for the component in the private

area of the class. A SCA component consists of several CORBA object representing the Core Framework and port interfaces. The classes must communicate information between them. By declaring the port classes as friend classes to the USRP_i class, the port classes can read and write data stored in the private area of the USRP_i class. Maintaining this data in the private area helps automatic documentation generation by not exposing data that is basically private to the component's public interface.

The USRP_i class also defines to threads to handle data transfer to the USRP for transmission and handle data received by the USRP.

2.3.3 Port Implementation

The port implementation section of the code contains all the code to support the ports for the USRP interface device. The USRP interface device has five ports; three provide an interface and two use an interface. The *provides ports* are the transmit control, receive control, and transmit data ports. The *provides ports* inherit from specific port implementations. The control ports are described above and the transmit data port implements an interface that receives two sequences of numbers, each pair of numbers represents a complex value. The exact format of the sequences depends on the number of channels transmitted.

The *uses ports* are connected to *provides ports* that belong to components that perform signal processing. A *uses port* implements methods defined in the Core Framework port class. These methods support the port connection operations. The connection operations supply the port with a CORBA object reference of the provides port that will receive data from the USRP.

3 Summary

The USRP, developed as part of the GNU Radio [3] project, is an interface board that converts streaming baseband data to an RF output with the help of band-specific daughter boards. Likewise the USRP and daughter boards create a streaming data stream from the RF inputs. RF Daughter boards convert the base band signals to and from the appropriate frequency bands. The FPGA on the USRP converts the data rate to match data rates supported by a USB 2.0 high speed controller. In addition the USB interface combined with the FPGA provide digital IO and low speed analog IO signals for use by daughter boards. The low speed analog IO points are useful for controlling variable gain amplifiers.

The USRP interface device program provides an interface between the USRP hardware and SCA waveform components. The high level control interfaces are structure so that they may be used to control hardware similar to the USRP, without changing any of the

waveform control software that connects to the USRP interface device. This allows waveform development on a standard piece of hardware to proceed, while special purpose hardware is developed. Since the control and data interfaces remain the same, the new hardware is easily integrated into the system.

The current USRP interface device has been extensively tested at lower data rates. The current testing is done with a narrowband FM waveform from a Family Band Service (FRS) radio that requires the maximum decimation rates available on the USRP. This is a rate of 250 thousand samples per second (Ksps). The transmit path has been tested at similar data rates. Good voice quality from the receiver has been demonstrated. The theoretical limit for data transfer over a USB 2.0 connection is 8 Msps. Over the next few months, we will perform testing to see if the combination of the USRP device and the OSSIE framework can do useful processing at higher data rates.

References

[1] "GnuRadio:UniversalSoftwareRadioPeripheral." http://comsec.com/wiki?UniversalSoftwareRadioPeripheral.

[2] "Front page - ossie." http://ossie.mprg.org/.

[3] "GNU Radio - GNU FSF Project." http://www.gnu.org/software/gnuradio/gnuradio.html.

[4] "Software Communications Architecture Specification," Final/15 May 2006 Version 2.2.2, Joint Program Executive Office (JPEO) of the Joint Tactical Radio System (JTRS), May 2006. Also available at http://jtrs.spawar.navy.mil/sca/.

[5] J. Bard and V. Kovarik, *Software Defined Radio: The Software Communications Architecture*. Wiley, 2007.

[6] "Welcome To The OMG's CORBA Website." http://www.corba.org/.

[7] "Analog Devices AD9862 - 12/14-Bit Mixed Signal Front-End (MxFE®) Processor for Broadband COmmunications." http://www.analog.com/en/prod/0%2C2877%2CAD9862%2C00.html.

[8] "USRP Diagrams - mux usage." http://webpages.charter.net/cswiger/usrp_diagrams/.

[9] "Untitled Document." http://www.ice-online.com/ICE-PIC4X.htm.

[10] "GnuRadio: GnuRadio2.X." http://comsec.com/wiki?GnuRadio2.X.

[11] "/platform/USRP/trunk/USRP - OSSIE - Trac." http://ossie-dev.mprg.org:8080/browser/platform/USRP/trunk/USRP.

[12] "subversion.tigris.org." http://subversion.tigris.org/.

A USRP.spd.xml

The software profile descriptor for the USRP device.

```
1  <?xml version="1.0" encoding="us-ascii"?>
2  <!DOCTYPE softpkg SYSTEM "../dtd/softpkg.dtd">
3  <!--
4      Generated by Zeligsoft Component Enabler 2.0.2 (Build:
           200510131317)
5      http://www.zeligsoft.com
6  -->
7  <softpkg id="DCE:1d9783ae-677d-4150-887e-3d56761d4c37" name="USRP"
       type="sca_compliant" version="">
8      <title />
9      <author>
10         <name>Ossie Team</name>
11         <company>Va Tech/MPRG</company>
12         <webpage>http://ossie.mprg.org/</webpage>
13     </author>
14     <descriptor name="">
15         <localfile name="USRP.scd.xml" />
16     </descriptor>
17     <!--[Implementation USRPImplementation_linux]-->
18     <implementation id="DCE:005a01c3-e469-4332-a4bd-a83b0159853e"
           aepcompliance="aep_compliant">
19         <code type="Executable">
20             <localfile name="../../bin/USRP" />
21         </code>
22         <os name="Linux" version="5.5.1" />
23         <processor name="x86" />
24     </implementation>
25 </softpkg>
```

Listing 2: USRP.spd.xml

B USRP.scd.xml

The software component descriptor for the USRP device.

```
1  <?xml version="1.0" encoding="us-ascii"?>
2  <!DOCTYPE softwarecomponent SYSTEM "../dtd/softwarecomponent.dtd">
3  <!--
4      Generated by Zeligsoft Component Enabler 2.0.2 (Build:
           200510131317)
5      http://www.zeligsoft.com
6  -->
7  <softwarecomponent>
8      <corbaversion>2.2</corbaversion>
9      <componentrepid repid="IDL:CF/Device:1.0" />
10     <componenttype>device</componenttype>
11     <componentfeatures>
12         <supportsinterface repid="IDL:CF/Device:1.0" supportsname=
               "Device" />
13         <supportsinterface repid="IDL:CF/PropertySet:1.0"
               supportsname="PropertySet" />
14         <supportsinterface repid="IDL:CF/Resource:1.0"
               supportsname="Resource" />
15         <supportsinterface repid="IDL:CF/PortSupplier:1.0"
               supportsname="PortSupplier" />
16         <supportsinterface repid="IDL:CF/LifeCycle:1.0"
               supportsname="LifeCycle" />
17         <supportsinterface repid="IDL:CF/TestableObject:1.0"
               supportsname="TestableObject" />
18         <ports>
19             <provides repid="IDL:standardInterfaces/TX_Control:1.0
                   " providesname="TX_Control">
20                 <porttype type="control" />
21             </provides>
22             <provides repid="IDL:standardInterfaces/RX_Control:1.0
                   " providesname="RX_Control">
23                 <porttype type="control" />
24             </provides>
```

```
25              <provides repid="IDL:StdInterfaces/ComplexPort:1.0"
                    providesname="TX_Data">
26                  <porttype type="data" />
27              </provides>
28              <uses repid="IDL:StdInterfaces/ComplexPort:1.0"
                    usesname="RX_Data_1">
29                  <porttype type="data" />
30              </uses>
31              <uses repid="IDL:StdInterfaces/ComplexPort:1.0"
                    usesname="RX_Data_2">
32                  <porttype type="data" />
33              </uses>
34          </ports>
35      </componentfeatures>
36      <interfaces>
37          <interface repid="IDL:StdInterfaces/ComplexPort:1.0" name=
                "ComplexPort"/>
38          <interface repid="IDL:CF/Device:1.0" name="Device">
39              <!--[Inherited interface IDL:CF/PropertySet:1.0]-->
40              <inheritsinterface repid="IDL:CF/PropertySet:1.0" />
41              <!--[Inherited interface IDL:CF/Resource:1.0]-->
42              <inheritsinterface repid="IDL:CF/Resource:1.0" />
43              <!--[Inherited interface IDL:CF/PortSupplier:1.0]-->
44              <inheritsinterface repid="IDL:CF/PortSupplier:1.0" />
45              <!--[Inherited interface IDL:CF/LifeCycle:1.0]-->
46              <inheritsinterface repid="IDL:CF/LifeCycle:1.0" />
47              <!--[Inherited interface IDL:CF/TestableObject:1.0]-->
48              <inheritsinterface repid="IDL:CF/TestableObject:1.0"
                    />
49          </interface>
50          <interface repid="IDL:standardInterfaces/TX_Control:1.0"
                name="TX_Control" />
51          <interface repid="IDL:standardInterfaces/RX_Control:1.0"
                name="RX_Control" />
52      </interfaces>
53  </softwarecomponent>
```