Open-Source SCA Implementation-Embedded and Software Communications Architecture

OSSIE and SCA Waveform Development

Edoardo Paone
The thesis project

Develop a OSSIE waveform based on IEEE 802.15.4:

- Understand SCA and the OSSIE waveform development
- Analyze the IEEE 802.15.4 standard protocol
- Import GNU Radio libraries into OSSIE components
- Implement a RX/TX waveform prototype using the USRP
- Evaluate the OSSIE waveform development tools and process
SDR – Software Defined Radio

Diagram:
- Receiver RF Front End
- ADC
- GPP

Diagram:
- Transmitter RF Front End
- DAC
- GPP
SoftModem

• modem with minimal hardware

• it uses host computer’s resources instead of dedicated hardware

• easier upgrades to new modem standards

• reduction in production costs, component size and weight
SCA Waveform Development

• Abstraction of underlying hardware and software
  – platform independence, portability of applications
  – reuse of waveform design modules

• CORBA for intercomponent communication
  – component implementation in different languages
  – distributed waveform (TCP transport rule)
SCA – Abstraction layers
Universal Software Radio Peripheral (USRP)

Materials for the thesis project:

- A personal computer (dualcore processor, 2 GB RAM)
- Ubuntu Linux 8.04
- GNU Radio 3.2.2
- OSSIE 0.7.4
- 2 USRPs
- Eclipse
OSSIE project

• **Open-Source SCA-based Implementation – Embedded (OSSIE)**

• Developed at Wireless@Virginia Tech (Blacksburg, VA)

• Professor-in-charge: Dr. Jeffrey H. Reed

• Source code, documentation, tutorials available at:

  http://ossie.wireless.vt.edu
OSSIE distribution

• Latest version: OSSIE 0.8.0 (released in January 2010)

• It can be installed on PCs running Linux (or in a virtual machine on Windows)

• The OSSIE package includes:
  – The Core Framework (CF)
  – The OSSIE Eclipse Feature (OEF)
  – A set of tools for waveform development
  – A component library
OSSIE Core Framework

• written in C++ using the omniORB CORBA ORB (open source)
• implements key elements of the SCA specification
• CORBA transport rules define inter-component communication:
  – TCP/IP, for distributed waveforms
  – Unix domain sockets


OSSIE Component
OSSIE Component

- **C++ or Python implementation**

- **Standard Interfaces:**
  basic real and complex data representations in 8, 16, and 32 bit sizes passed in the form of CORBA sequences

- **Custom Interfaces:**
  can be defined using Interface Description Language (IDL) and compiled with the IDL compiler
OSSIE Component

• **Data buffering:**
effectively decouples the CORBA call sequence from signal processing code
OSSIE Waveform
OSSIE Waveform

- Drag-and-drop of component library, graphical tools for interconnecting components
- Waveform XML representation generated by the OEF:
  - List of components
  - Configuration of components
  - Component interconnections
  - Target deployment node
OSSIE Waveform

<connections>
  <connectinterface id="DCE:e1aa5d66-824a-11dc-adea-00123f63025f">
    <providesport>
      <providesidentifier>samples_in</providesidentifier>
      <findby>
        <namingservice name="ChannelDemo1"/>
      </findby>
    </providesport>
    <usesport>
      <usesidentifier>symbols_out</usesidentifier>
      <findby>
        <namingservice name="TxDemo1"/>
      </findby>
    </usesport>
  </connectinterface>
  <connectinterface id="DCE:e1ad7154-824a-11dc-85b0-00123f63025f">
    <providesport>
      <providesidentifier>symbols_in</providesidentifier>
      <findby>
        <namingservice name="RxDemo1"/>
      </findby>
    </providesport>
    <usesport>
      <usesidentifier>samples_out</usesidentifier>
      <findby>
        <namingservice name="ChannelDemo1"/>
      </findby>
    </usesport>
  </connectinterface>
</connections>
IEEE 802.15.4

- MCPS = MAC common part sublayer
- MCPS-SAP = MCPS service access point
- MLME = MAC layer management entity
- MLME-SAP = MLME service access point
- PD = PHY data
- PLME = PHY layer management entity
- SSCS = service-specific convergence sublayer
- LLC = logical link control
Custom Interfaces - IDL
Custom Interfaces - IDL

#include "ossie/PortTypes.idl"

module customInterfaces{

   /* link layer SAP for mac layer */
   interface sap_mac_to_link{

      void macpsData_confirm( in octet msduHandle,
                                in octet status,
                                in unsigned long Timestamp );

      void mcpsData_indication( in octet SrcAddrMode,
                                 in unsigned short SrcPANId,
                                 in unsigned long long SrcAddr,
                                 in octet DstAddrMode,
                                 in unsigned short DstPANId,
                                 in unsigned long long DstAddr,
                                 in octet msduLength,
                                 in PortTypes::CharSequence msdu,
                                 in octet msduLinkQuality,
                                 in octet DSN,
                                 in unsigned long Timestamp,
                                 in octet SecurityLevel,
                                 in octet KeyIdMode,
                                 in unsigned long long KeySource,
                                 in octet KeyIndex );

   };

};
Component Structure
PHY layer
MAC - Simulation

MAC 1

s
a
p
m
a
t
o
p
h
y
s
a
p
h
y
t
m
a

MAC 2

TEST INPUT GENERATOR

TEST COMPONENT

TEST OUTPUT OBSERVER

TEST WAVEFORM
PHY - Simulation

![PHY Simulation Diagram]

- **MAC 1**
- **PHY 1**
- **PHY 2**
- **MAC 2**

**TEST COMPONENT**

**TEST WAVEFORM**

**TEST INPUT GENERATOR**

**TEST OUTPUT OBSERVER**
RX/TX path – GNU Radio solution

MAC → PHY → MODEM_USRP

USRP
UCLA Zigbee PHY

GNU Radio implementation of the IEEE 802.15.4 physical layer.
MODEM_USRP WorkModule
MODEM_USRP component

frameIn
(OSSIE input port)

send_packet(frameIn)

MODEM_USRP
(OSSIE component)

phy
(GNU Radio block)

callback(frameOut)

frameOut
(OSSIE output port)
RX/TX path – GNU Radio solution

Waveform Development

MAC | PHY | MODEM_USRP

GPP device | USRP device

SCA platform

OS / HW

Memory | TCP/IP

GPP | USRP
RX/TX path – OSSIE solution

Waveform Development

MAC → PHY → MODEM

SCA platform

GPP device → Memory

TCP/IP

USRP device → USRP

OS / HW

GPP

USRP
RX/TX path – OSSIE solution
MODULATOR component

- Python implementation
- mylib.symbol_sink

• ieee802_15_4_mod
• input/output buffers
DEMODULATOR component

- C++ implementation
- `fm_demodulation.h`
- `symbol_source.h`
- `input/output buffers`

- `ucla_ieee802_15_4_packet_sink.h`
- `output buffer`
- `receiver_path` (C++ class)
- `frameOut` (OSSIE port, realChar)
- `signalln` (OSSIE port, complexShort)
Waveform Simulation

- all components deployed on the GPP (no USRP)
- radio channel simulated by the channel component
- test case implemented in the TEST component
The target waveform

Two configurations:

- a chat application

  CHAT ➔ MAC ➔ PHY ➔ MODEM USRP

- a TUN/TAP interface

  TUN/TAP ➔ MAC ➔ PHY ➔ MODEM USRP
## GNU Radio vs. OSSIE

<table>
<thead>
<tr>
<th>GNU Radio</th>
<th>OSSIE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Waveform structure:</strong></td>
<td></td>
</tr>
<tr>
<td>• the <code>top_module</code> class represents a container for the flowgraph</td>
<td>• defined in the XML descriptor files and instanciated at runtime</td>
</tr>
<tr>
<td>• the flowgraph is then implemented as a single thread</td>
<td>• one or more threads for each component</td>
</tr>
<tr>
<td><strong>Component interconnections:</strong></td>
<td></td>
</tr>
<tr>
<td>• defined in the source code with the <code>connect()</code> method</td>
<td>• defined in the XML descriptor files and instanciated at runtime</td>
</tr>
<tr>
<td>• functional de-composition and hierarchical structure</td>
<td>• functional composition but flat structure</td>
</tr>
<tr>
<td><strong>Data transfer:</strong></td>
<td></td>
</tr>
<tr>
<td>• if data is available in the input buffer, the <code>top_module</code> calls the <code>work</code> method</td>
<td>• performed by CORBA calls, data is temporarily stored in input/output buffers</td>
</tr>
<tr>
<td>• synchronization and scheduling techniques</td>
<td>• no synchronization between components, independent threads</td>
</tr>
</tbody>
</table>
# Measurements

<table>
<thead>
<tr>
<th></th>
<th>GNU Radio</th>
<th>OSSIE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average data transfer size</td>
<td>12.5 KB</td>
<td>16 KB</td>
</tr>
<tr>
<td>Average data transfer delay</td>
<td>1.5 ms</td>
<td>1 ms</td>
</tr>
<tr>
<td>Average input data rate</td>
<td>8 MB/s</td>
<td>15.4 MB/s</td>
</tr>
</tbody>
</table>

**USRPs:**
- ADC sample rate: 64 Msp/s
- Decimation factor: 16
- Output sample rate: 4 Msp/s
- Theoretical limit USB 2.0: 8 Msp/s
USRP - Data transfer delay

Delay between successive read operations from the USRP
USRP - Data transfer size

Sixe of data read from the USRP

GNU Radio
OSSIE

Data transfer size (bytes)

Experiment number
Results

• GNU Radio:
  – Adaptive scheduling
  – Variable data rate
  – Data transfer synchronization

• OSSIE:
  – Independent scheduling for each component
  – Constant data rate
  – No synchronization between components
Thank you for your attention

Acknowledgements