Bridging Automation and Robotics
An Interprocess Communication between IEC 61131-3 and ROS

Tiago Pinto
tiago.f.pinto@inesctec.pt

ROS-Industrial EU Spring'18 Workshop
Stuttgart, 28 May 2018
**Concept**

**Problem:**
- Robot re-programming by automation technitians who do not have the required skills to delve into complicated robot systems.
- Implementing **multi-vendor and multi-protocol communication** between ROS and industrial equipment.
- Time consumed programming drivers for actuators.

**Solution:**
- Embedded System Development with ROS and CODESYS softPLC Integration
ROS – CODESYS Communication

Shared Memory

PROFIBUS
EtherNet/IP
CANopen
EtherCAT
...

Concept - Architecture - Shared Memory - Implementation (ROS - CODESYS) - Practical Applications - Conclusions
Architecture of the system
POSIX Shared Memory

• CODESYS Provides implementation function for the POSIX Named Shared Memory.

• It is known as the fastest way of passing data between two processes on the same host system.

• This method needs a synchronization mechanism.
Access Synchronization

- CODESYS has a library with Semaphore functions.
- This library implements the POSIX Named Semaphores.
- Can be implemented both on ROS and CODESYS
Architecture of the system

ROS Messages to IEC 61131-3 Data Types

<table>
<thead>
<tr>
<th>Description</th>
<th>ROS Messages Primitive Type</th>
<th>C++</th>
<th>IEC 61131-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsigned 8-bit Integer</td>
<td>bool</td>
<td>uint8_t</td>
<td>USINT</td>
</tr>
<tr>
<td>Signed 8-bit Integer</td>
<td>int8</td>
<td>int8_t</td>
<td>SINT</td>
</tr>
<tr>
<td>Unsigned 8-bit Integer</td>
<td>uint8</td>
<td>uint8_t</td>
<td>USINT</td>
</tr>
<tr>
<td>Signed 16-bit Integer</td>
<td>int16</td>
<td>int16_t</td>
<td>INT</td>
</tr>
<tr>
<td>Unsigned 16-bit Integer</td>
<td>uint16</td>
<td>uint16_t</td>
<td>UDINT</td>
</tr>
<tr>
<td>Signed 32-bit Integer</td>
<td>int32</td>
<td>int32_t</td>
<td>DINT</td>
</tr>
<tr>
<td>Unsigned 32-bit Integer</td>
<td>uint32</td>
<td>uint32_t</td>
<td>UDINT</td>
</tr>
<tr>
<td>Signed 64-bit Integer</td>
<td>int64</td>
<td>int64_t</td>
<td>LINT</td>
</tr>
<tr>
<td>Unsigned 64-bit Integer</td>
<td>uint64</td>
<td>uint64_t</td>
<td>ULINT</td>
</tr>
<tr>
<td>32-bit IEEE Float</td>
<td>float32</td>
<td>float</td>
<td>REAL</td>
</tr>
<tr>
<td>64-bit IEEE Float</td>
<td>float64</td>
<td>double</td>
<td>LREAL</td>
</tr>
<tr>
<td>ASCII String</td>
<td>string</td>
<td>std::string</td>
<td>STRING</td>
</tr>
<tr>
<td>Time (secs/nsecs)</td>
<td>time</td>
<td>ros::Time</td>
<td>TIME</td>
</tr>
<tr>
<td>Time (secs/nsecs)</td>
<td>duration</td>
<td>ros::Duration</td>
<td>TIME</td>
</tr>
</tbody>
</table>

Concept - Architecture - Shared Memory - Implementation (ROS - CODESYS) - Practical Applications - Conclusions
Mapping data structures

- **ROS Messages** are organized into structures
- **IEC61131-3** also supports structures

Implemented ROS Messages

std_msgs:
- Already implemented

common_msgs:
- Almost all implemented
- Problem with variable length arrays on CODESYS

Custom messages:
- Needs to be implemented by the user.
- Will be automated in the future.

<table>
<thead>
<tr>
<th>ROS Standard Messages</th>
<th>ROS Common Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bool</td>
<td>actionlib_msgs</td>
</tr>
<tr>
<td>Byte</td>
<td>GoalID</td>
</tr>
<tr>
<td>ByteMultiArray</td>
<td>GoalStatus</td>
</tr>
<tr>
<td>Char</td>
<td>GoalStatusArray</td>
</tr>
<tr>
<td>ColorRGBA</td>
<td></td>
</tr>
<tr>
<td>Duration</td>
<td></td>
</tr>
<tr>
<td>Empty</td>
<td></td>
</tr>
<tr>
<td>Float32</td>
<td>DiagnosticArray</td>
</tr>
<tr>
<td>Float64</td>
<td>DiagnosticStatus</td>
</tr>
<tr>
<td>Header</td>
<td>KeyValue</td>
</tr>
<tr>
<td>Int16</td>
<td>nav_msgs</td>
</tr>
<tr>
<td>Int32</td>
<td></td>
</tr>
<tr>
<td>Int64</td>
<td></td>
</tr>
<tr>
<td>Int8</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>UInt16</td>
<td></td>
</tr>
<tr>
<td>UInt32</td>
<td></td>
</tr>
<tr>
<td>UInt64</td>
<td></td>
</tr>
<tr>
<td>UInt8</td>
<td></td>
</tr>
</tbody>
</table>
Implementation on ROS

- **Topic based** implementation.
- **Automatic shared memory writing** on subscribing callback.
- **Automatic publishing** of the data read from the shared memory.
- Shared Memory access synchronization by semaphores.
Usage on ROS – Writing on the Shared Memory

```cpp
PoseStamped_SharedMemory poseStampedSMW("poseStampedWrite_example", WRITE);
UInt16_SharedMemory uint16SMW("uint16Write_example", WRITE, PRINT);
String_SharedMemory stringSMW("stringWrite_example", WRITE, NOPRINT);

while (ros::ok())
{
    poseStampedSMW.memoryWrite();
    uint16SMW.memoryWrite();
    stringSMW.memoryWrite();
    ros::spin();
}
```
Usage on ROS – Reading from the Shared Memory

```cpp
PoseStamped_SharedMemory poseStampedSMR("poseStampedRead_example", READ);
Float64_SharedMemory float64SMR("float64Read_example", READ, NOPRINT);
Int32_SharedMemory int32SMR("int32Read_example", READ, PRINT);

while (ros::ok())
{
    poseStampedSMR.memoryRead();
    float64SMR.memoryRead();
    int32SMR.memoryRead();
    ros::spinOnce();
    loop_rate.sleep();
}
```
Implementation on CODESYS

- **All variables accessible by GVL.** GVL can be accessed by multiple programs.

- **Automatic update** of the variables.

- Function Blocks for reading and writing can be used with graphical languages.

- Shared Memory access synchronization by semaphores.
Implementation example on CODESYS

CODESYS Structures

- Quaternion
  - x: LREAL
  - y: LREAL
  - z: LREAL

- Point
  - x: LREAL
  - y: LREAL
  - z: LREAL

- Time
  - sec: DWORD
  - nsec: DWORD

- Header
  - frameld: STRING[30]
  - seq: UDINT
  - stamp: TIME

- Pose

- PoseStamped

Function Blocks

- PoseStamped_Read (FB)
  - FB_Init
  - FB_Init

- PoseStamped_Write (FB)
  - FB_Init
  - MEM_Write

Concept - Architecture - Shared Memory - Implementation (ROS - CODESYS) - Practical Applications - Conclusions
Usage on CODESYS

- Device (CODESYS Control for BeagleboneBlack SL)
- Plc Logic
  - Application
    - ROS
- Library Manager
- PLC_PRG (PRG)
- ROS_Read (PRG)
- ROS_Write (PRG)
- Task Configuration
  - MainTask
    - PLC_PRG
    - ROS_Read
    - ROS_Write
- SoftMotion General Axes Pool
- <Empty> (<Empty>)

```plaintext
3  VAR_GLOBAL
4  poseStampedSMR : PoseStamped_Read('poseStampedWrite_example');
5  uint16SMR : UInt16_Read('uint16Write_example');
6  stringSMR : String_Read('stringWrite_example');
7  
8  poseStampedSMW : PoseStamped_Write('poseStampedRead_example');
9  float64SMW : Float64_Write('float64Read_example');
10  int32SMW : Int32_Write('int32Read_example');
11  END_VAR
```
Usage on CODESYS

1. Device (CODESYS Control for BeagleboneBlack SL)
2. Plc Logic
3. Application
   - ROS
   - Library Manager
   - PLC_PRG (PRG)
     - ROS_Read (PRG)
     - ROS_Write (PRG)
4. Task Configuration
   - MainTask
     - PLC_PRG
     - ROS_Read
     - ROS_Write
5. SoftMotion General Axis Pool
   - <Empty> (<Empty>)

Code examples:

```c
2 poseStampedSMR.MEM_Read();
3 uint16SMR.MEM_Read();
4 stringSMR.MEM_Read();
```

```c
2 poseStampedSMW.MEM_Write();
3 float64SMW.MEM_Write();
4 int32SMW.MEM_Write();
```
Usage on CODESYS

Device (CODESYS Control for BeagleboneBlack SL)

- PLC Logic
  - Application
    - ROS
      - Library Manager
      - PLC_PRG (PRG)
      - ROS_Read (PRG)
      - ROS_Write (PRG)
    - Task Configuration
      - MainTask
      - PLC_PRG
        - ROS_Read
        - ROS_Write
    - SoftMotion General Axis Pool
      - <Empty> (<Empty>)

Configuration

- Priority (0..31):
- Type: Cyclic
- Interval (e.g. t#200ms):
- Watchdog: Enable
- Time (e.g. t#200ms):
- Sensitivity: 1

Concept - Architecture - Shared Memory - Implementation (ROS - CODESYS) - Practical Applications - Conclusions
Practical Applications
INESC Omnidirectional mobile manipulator

• Using CODESYS to manage the CANopen drivers:
Practical Applications
Scalable Project – PSA use case
Overview and Future Work

Overview

- A communication system between ROS and CODESYS was implemented using **shared memory**.

- Three potential practical applications:
  1. Using CODESYS as Communication Bridge
  2. Horizontal Integration between ROS and CODESYS
  3. Programming robotic tasks in IEC 61131-3 programming languages

Future Work

- **Simplify the introduction of ROS Custom Messages to be used by this bridge:**
  - A tool will be developed to handle custom messages.
  - On CODESYS, auto generation of code and easy code import will be investigated.
  - Implement a solution to overcome the variable length arrays issue on CODESYS.

- Include **ROS Services and ROS Actions**.
Live Demo

Beaglebone Black

CODESYS
Web Visualization

ROS

CODESYS

Joystick

Motor commands

Motor velocity

CODESYS

Web Visualization

ROS

CODESYS

Motion Simulation
Demo development on ROS
Demo development on ROS

```cpp
#include "ros/ros.h"
#include "codesys_shared_memory/class_headers/Std_Msgs_SharedMemory.h"

int main(int argc, char **argv){
  ros::init(argc, argv, "shared_memory_read");
  Float64_SharedMemory velocity("motor_velocity", READ);
  Float64_SharedMemory position("motor_position", READ);

  while (ros::ok()){  
    velocity.memoryRead();
    position.memoryRead();
    ros::spin();
  }
  return 0;
}
```

```cpp
#include "ros/ros.h"
#include "codesys_shared_memory/class_headers/Custom_Msgs_SharedMemory.h"

int main(int argc, char **argv){
  ros::init(argc, argv, "shared_memory_write");
  JoyInterface_SharedMemory joy_interface_write("joy_interface", WRITE);

  while (ros::ok()){
    joy_interface_write.memoryWrite();
    ros::spin();
  }
  return 0;
}
```
Demo development on CODESYS

GVL Definition

Shared memory operation program

Portion of Motor program FBD

Data can be read or written directly on graphical languages

Can be assigned to a real motor
Demo