Part1: About OpenRTM-aist and Outline of RT-Component Programming

National Institute of Advanced Industrial Science and Technology (AIST), Japan
Industrial CPS Research Center
Software Platform Research Team, Team Leader
Noriaki Ando, Ph.D.
Outline

• Basic concept and overview of RT-Middleware
• Comparison with ROS
• Activities of RT-Middleware community
• RTC development overview
• Conclusion
What is RT-Middleware?
What is RT?

• RT = Robot Technology   cf. IT
  – not only standalone robots, but also robotic elements (sensors, actuators, etc.…)

RT-Middleware developed by AIST

OpenRTM-aist

• RT-Middleware
  – middleware and platform for RT-element integration
• RT-Component
  – basic software unit in RT-Middleware
About Robot Middleware

• **Platform software** that provides common functions to streamline robot system construction
  - Sometimes called "robot OS"
  - Commonization and standardization of interface and protocols
  - Examples
    • Providing modular or componentized frameworks
    • Supports communication among modules
    • Provides parameter setting, deployment, startup, and module composition functions
    • Realize inter-OS and inter-language cooperation / interoperability by abstraction

• Development became active from around 2000
  - Various middleware is being developed and released all over the world
Conventional systems

Controller

Robot Controller Program

Controller software

Robot Arm

Compatible interfaces are connectable

Robot Arm1

Target Robot
Each robot has each control interfaces. If no compatibility, cannot connect each other.
By using RT-Middleware

RTM provides a common I/F for connecting separately made software modules

Improved software reusability
Easy to build RT system
Trend of Robot Software Development

Conventional Style

- Integrated design of various functions
- High run-time efficiency, but inflexible
- Development becomes difficult as the system becomes more complex

Component Oriented Development

- Division/integration of large-scale complex functions
- Improvement of development and maintenance efficiency (reuse of functions, etc.)
- Increased system flexibility
The benefits of modularization

• **Reusability**
  – A component can be reused in various systems.

• **Diversification**
  – Various type of same functional modules can be tried in systems.

• **Flexibility**
  – System structure can be changed easily.

• **Reliability**
  – Easy to test a module and well tested modules are reliable.

• **Durability**
  – Well divided and independent module error does not affect too much to whole systems.
The benefits of RT-Component model

• Provides rich component lifecycle to enforce state coherency among components
• Defines data structures for describing components and other elements
• Supports fundamental design patterns
  – Collaboration of fine-grained components tightly coupled in time (e.g. Simulink)
  – Stimulus response with finite state machines
  – Dynamic composition of components collaborating synchronously or asynchronously
Main features of RT-Component

Activity, Execution context

- Common state machine
  - Inactive
  - Active
  - Error
- Composite execution
- Real-time execution
- Life cycle management, core logic execution

Data Port

- Data centric communication
- Continuous data transfer
- Dynamic connection/disconnection

Service Port

- User defined interface
- Access to detailed functionality of RTC
  - Getting/setting parameters
  - Changing modes
  - etc...
- Ex. Servo control
- Ex. Stereo vision

Configuration

- Function for internal parameter
- Multiple parameter sets
- They can be changed from remote in run-time
- RTC can have several configuration sets.
  Runtime reconfiguration and dynamic switching are supported

Data-centric communication

Service oriented interaction

- Stereo vision interface
  - set_mode()
  - set_coordination()
  - do_calib()
  - etc...
- Image data
- Stereo Vision Component
- 3D depth data
- Data port
- Service port

- Encoder
- Controller
- Actuator
- Reference pos
- Torque
- Set name
- Name
- Value
- Set name
- Name
- Value
Advantages of RTM based development

- Self development using existing libraries
- Develop from scratch
- Reusable RTCs

Bin-picking robot example
* This diagram is simplified version

By using RTM, existing module can be reused
Reduce development from scratch
RTM can manage distributed RTCs implemented by various languages or executed on various OSs on the network.

Connections Between RTCs Can be established dynamically.
The aim of RT-Middleware

Problem Solving by Modularization

Cost
Realize low-cost robots

Technical Issue
Utilize the state of the art

Needs
Satisfy various needs

Robot System Integration Innovation
Practical/commercialization examples

HRP series: KAWADA and AIST
HIRO, NEXTAGE open: Kawada Robotics
VSTONE’s education robots

S-ONE : SCHAFT
THK: SIGNAS system
OROCHI (RT corp.)

DAQ-Middleware: KEK/J-PARC
KEK: High Energy Accelerator Research Organization
J-PARC: Japan Proton Accelerator Research Complex

TOYOTA L&F : Air-T
Robot operation simulator: NEDO
RTM as an International Standard

OMG Standard

Standardized by OMG process
→ It can not be modified by just one company
→ Various compatible implementation
→ It promoted competition and interoperability

Ten or more RT-Middleware implementation exist

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Vendor</th>
<th>Features</th>
<th>Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenRTM-aist</td>
<td>AIST</td>
<td>Reference implementation by AIST</td>
<td></td>
</tr>
<tr>
<td>HRTM</td>
<td>Honda R&amp;D</td>
<td>ASIMO is now moving to HRTM</td>
<td>☑</td>
</tr>
<tr>
<td>OpenRTM.NET</td>
<td>SEC</td>
<td>.NET(C#, VB, C++, CLI, F#, etc..)</td>
<td>☑</td>
</tr>
<tr>
<td>RTM on Android</td>
<td>SEC</td>
<td>RTM implementation for Android</td>
<td>☑</td>
</tr>
<tr>
<td>RTC-Lite</td>
<td>AIST</td>
<td>Tiny implementation on PIC and dsPIC</td>
<td></td>
</tr>
<tr>
<td>Mini/MicroRTC</td>
<td>SEC</td>
<td>RTM/RTC for CAN and Zigbee</td>
<td>☑</td>
</tr>
<tr>
<td>RTMSafety</td>
<td>SEC/AIST</td>
<td>Functional safety standard capable RTM</td>
<td></td>
</tr>
<tr>
<td>RTC CANOpen</td>
<td>SIT, CIA</td>
<td>RTM for CANOpen standard</td>
<td>☑</td>
</tr>
<tr>
<td>PALRO</td>
<td>Fujisoft</td>
<td>Yet another C++ PSM implementation</td>
<td>×</td>
</tr>
<tr>
<td>OPRoS</td>
<td>ETRI</td>
<td>Implementation of Korean national project</td>
<td>×</td>
</tr>
<tr>
<td>GostaiRTC</td>
<td>GOSTAI, THALES</td>
<td>C++ PSM implementation on a robot language</td>
<td>×</td>
</tr>
</tbody>
</table>

Users can chose and continue to use one of the RTM implementations

Robotic Technology Component (RTC)

Version 1.1

History
• September, 2005 Request for Proposal issued (starting standardization)
• September, 2006 Specification approved by OMG
• April, 2008 OMG RTC ver.1.0 released
• September, 2012 Updated to ver. 1.1
• September, 2015 FSM4RTC (FSM based RTC standard) adopted

Normative reference:  http://www.omg.org/spec/RTC/1.1
Non-normative:  http://www.omg.org/spec/RTC/20111206/rtc.rtc

Users can chose and continue to use one of the RTM implementations.

Standardized by OMG process
→ It can not be modified by just one company
→ Various compatible implementation
→ It promoted competition and interoperability

Ten or more RT-Middleware implementation exist

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Vendor</th>
<th>Features</th>
<th>Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenRTM-aist</td>
<td>AIST</td>
<td>Reference implementation by AIST</td>
<td></td>
</tr>
<tr>
<td>HRTM</td>
<td>Honda R&amp;D</td>
<td>ASIMO is now moving to HRTM</td>
<td>☑</td>
</tr>
<tr>
<td>OpenRTM.NET</td>
<td>SEC</td>
<td>.NET(C#, VB, C++, CLI, F#, etc..)</td>
<td>☑</td>
</tr>
<tr>
<td>RTM on Android</td>
<td>SEC</td>
<td>RTM implementation for Android</td>
<td>☑</td>
</tr>
<tr>
<td>RTC-Lite</td>
<td>AIST</td>
<td>Tiny implementation on PIC and dsPIC</td>
<td></td>
</tr>
<tr>
<td>Mini/MicroRTC</td>
<td>SEC</td>
<td>RTM/RTC for CAN and Zigbee</td>
<td>☑</td>
</tr>
<tr>
<td>RTMSafety</td>
<td>SEC/AIST</td>
<td>Functional safety standard capable RTM</td>
<td></td>
</tr>
<tr>
<td>RTC CANOpen</td>
<td>SIT, CIA</td>
<td>RTM for CANOpen standard</td>
<td>☑</td>
</tr>
<tr>
<td>PALRO</td>
<td>Fujisoft</td>
<td>Yet another C++ PSM implementation</td>
<td>×</td>
</tr>
<tr>
<td>OPRoS</td>
<td>ETRI</td>
<td>Implementation of Korean national project</td>
<td>×</td>
</tr>
<tr>
<td>GostaiRTC</td>
<td>GOSTAI, THALES</td>
<td>C++ PSM implementation on a robot language</td>
<td>×</td>
</tr>
</tbody>
</table>
Comparison with ROS, and trend
ROS and RTM

Pros of ROS

- Design policy based on UNIX culture
  - Officially supports only Linux
  - ROS2 supports Windows and MacOS
- Original package management systems
- Abundant quality and quantity of nodes (components)
  - Many nodes whose quality is directly controlled by OSRF
- Large number of users
- Discussions such as forum and ML are open and active
  - Sometimes core lib’s specification moves by users opinion
- Large number of English documents

Pros of OpenRTM

- Many OS/Language support
  - Windows, Linux (and other UNIX), MacOS, Real-time OS (VxWorks, QNX)
  - Windows native support
- Mainly spread in Japan
  - Japanese document, ML, tutorials
  - Less numbers of users
- GUI tools for beginners available
  - Eclipse based tools officially provided
  - CUI tools also available (rtshell)
- Standardized specification
  - Open revision procedure at OMG
  - Third-party implementation is welcome
  - Some compatible implementations exist
- Well-defined component model
  - High affinity with object-oriented, UML and SysML design
  - Model-based development
- IEC61508 functional-safety certification ready
  - RTMSafety by SEC corp.
From ROS1 to ROS2

• When they undertook NASA’s work, ROS original messaging protocol was not allowed to use. So prototyping was implemented in ROS, but they are re-implemented from scratch in the final product.

• DDS (Data distribution service, which is one of the OMG standardized messaging protocol) was used, because the only standardized protocol is allowed in NASA.

• There are inconvenient constraints such as no component model exists, only one-node-one-process model is allowed, ROS-master’s SPOF (single point of failure) problem.

• Therefore, ROS2 is completely new implementation to overcome these problems, so no compatibility between ROS1 and ROS2.
ROS2

- Current release: Foxy Fitzroy, June 5th, 2020
- Use of standardized middleware
  - Stop reinventing the wheel
  - Communication middleware is DDS standardized in OMG
- Component model is introduced
  - Suitable for embedded use and performance effective architecture like RTC
- Expansion of supported OS
  - ROS1 only supports Linux (only Ubuntu Linux supported)
  - Windows and MacOS support are added in ROS2
  - But no support for commercial real-time OS such as VxWorks, QNX

http://design.ros2.org/articles/ros_middleware_interface.html
RT-Middleware community
Project web pages

- Users can upload their own RTCs on the openrtm.org
- Users can search and download other users RTCs

<table>
<thead>
<tr>
<th>Project type</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT-Components</td>
<td>405</td>
</tr>
<tr>
<td>RT-Middleware</td>
<td>14</td>
</tr>
<tr>
<td>Tools</td>
<td>19</td>
</tr>
<tr>
<td>Documents</td>
<td>4</td>
</tr>
<tr>
<td>Hardware related RTCs</td>
<td>28</td>
</tr>
</tbody>
</table>
RT-Middleware Summer Camp

• 1 week camp every summer
• This year: August 24-28
  – The first online camp
• Number of participants: 11
• Venue: AIST Tsukuba center (Tsukuba city, Ibaraki pref.)
  – Online (Zoom)
• Lectures, practical work and presentation by five teams.
• Staying in the AIST’s accommodation and coding endlessly every night :-P.
RT-Middleware Contest

• Held as an organized session in SICE SI conference
  – Various prizes
  – Entry deadline: Sep. 23rd
  – Paper submission due: Oct. 26th
  – Online examination: from end of Nov.
  – Presentation and award ceremony: Dec.

• Record of year 2019
  – Number of applications: 11
  – SICE RT-Middleware award x1
  – Product supporting award x2
  – Company supporting award x9
  – Personal supporting award x10

• See more details: openrtm.org
  – Menu: community -> events
RTC development overview
Framework and core-logic

RTC standard interfaces

empty

RT-Component framework

+  

Stereo vision algorithm

Core-logic

Left image  Right image

Depth map

RTC standard interfaces

Left image  Right image

Depth map

Stereo vision RT-Component

RTC framework + Core logic = RT-Component
Code generation by model

RTC’s specification

- name: MyComp
- category: temp.sensor device
- description: temp. sensor RTC
- comp_type: STATIC
- act_type: PERIODIC
- InPorts: mode: TimedBool
- OutPorts: temp: TimedDouble

RTCBuilder (Template code generator)

C++ backend

RTC source for C++
class MyComp : public DataFlowComponent {
    public:
        virtual onExecute(ec_id);
    private:
        TimedBool m_mode;
        TimedDouble m_temp;
};

Java backend

RTC source for Java
import RTC.DataFlowComponent;
public class MyCompImpl extends DataFlowComponent {
    public ConsoleInImpl(mgr);
};

Python backend

RTC source for Python
#!/usr/bin/env python
import RTC
class MyComp(DataFlowComponent):
    def __init__(self, manager):
        ...
    def onExecute(self):
        ...

Core-logic is embedded to the template code
Implement the procedure in a specific function of the generated class
Same (model) component will be generated by same specification, even if implementation languages is different
RTC development flow

Windows

Input spec of RTC

Generating VC project or Makefile

Almost all steps are the same, except compiler

Linux

Implement logic and compile

make + gcc (g++)

Windows

RTBUilder

CMake

Visual C++

RTBUilder

CMake

gcc (g++)
CMake

- Open source software for compiler-independent build automation
- Can generate build files for different development environments on different operating systems
- Generate Makefile on Linux
- Generate VC (Visual C++) project file on Windows
- Most of the recent open source software is built with CMake.
State machine and lifecycle of RTC

- Created
- Alive
  - entry/RTC::onInitialize
  - exit/RTC::onFinalize
  - Stopped
  - exit/RTC::onStartup
  - ExecutionContext::start
  - Running
  - exit/RTC::onShutdown
  - ExecutionContext::stop
- Inactive
  - ExecutionContext::activate_component
  - Active
  - ExecutionContext::deactivate_component
  - entry/RTC::onActivate
  - exit/RTC::onDeactivate
  - [ReturnCode_t! = OK]
  - RTC::onError
  - Error
  - do/RTC::onReset
  - [ReturnCode_t! = OK]
  - Steady Error
  - ExecutionContext::reset_component
  - Resetting
  - entry/RTC::onReset
  - [ReturnCode_t = OK]

You don’t need to aware here

Be aware state here and its callback functions.

ActiveDo/RTC::onExecute() callback function is here
## Activity (Callback functions)

<table>
<thead>
<tr>
<th>Callback functions</th>
<th>Meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>onInitialize</td>
<td>Initialization</td>
</tr>
<tr>
<td>onActivated</td>
<td>Called once when RTC is activated</td>
</tr>
<tr>
<td>onExecute</td>
<td>Called periodically when RTC is in the active state</td>
</tr>
<tr>
<td>onDeactivated</td>
<td>Called once when RTC is deactivated</td>
</tr>
<tr>
<td>onAborting</td>
<td>Called once when entering ERROR state</td>
</tr>
<tr>
<td>onReset</td>
<td>Called once when resetting</td>
</tr>
<tr>
<td>onError</td>
<td>Called periodically when RTC is in the error state</td>
</tr>
<tr>
<td>onFinalize</td>
<td>Called once when finalizing RTC</td>
</tr>
<tr>
<td>onStateUpdate</td>
<td>Called after onExecute everytime</td>
</tr>
<tr>
<td>onRateChanged</td>
<td>Called when ExecutionContext’s rate is changed</td>
</tr>
<tr>
<td>onStartup</td>
<td>Called once when ExecutionContext starting</td>
</tr>
<tr>
<td>onShutdown</td>
<td>Called once when ExecutionContext stopping</td>
</tr>
</tbody>
</table>
InPort

- **InPort**
  - Input port for data flow type communication
- **Methods of InPort class**
  - `isNew()`: check if new data arriving
  - `read()`: retrieve data from InPort buffer to the variable bound to the InPort
  - `>>`: same as above

Basicallly paired with OutPort

Data ports (InPort/OutPort) must have the same type

Example

Sensor Data

Robot Component
OutPort

- **OutPort**
  - Output port for data flow type communication
- **Methods of OutPort class**
  - `write()`: push data from OutPort’s variable into OutPort’s buffer to be published to the remote InPort
  - `<<`: same as above

Basically paired with InPort

Data ports (InPort/OutPort) must have the same type
Recommendation

• Let’s stop reinventing the wheel!!
  - Code that has been executed thousands of times by different people works better than your code from scratch!!
  - Let's write the code you really need to write and borrow the other non-essential part for you.
  - A program released by someone is a program that has worked once!!
  - Other persons code is hard to read, but you shouldn't throw it away for that reason!!

• Commit to open source projects!!
  - Don't hesitate to ask questions on ML and forums!!
  - No matter how rudimentary a question is, it is valuable information for others.
  - Let’s complain to the project!! (good feedback grow the project)
  - Debug and send patches if you can!!
Conclusion

• Basic concept and overview of RT-Middleware
• Comparison with ROS
• Activities of RT-Middleware community
• RTC development overview
• Part2: Introduction to creating RT-Component
  – Lecturer: Nobuhiko Miyamoto

• Please check your development environment
  – Did you install OpenRTM-aist and other dependent software?
  – During lunch time, staffs support installation, if you do not install them yet.
  – If you have any question, please call us via Zoom’s chat or Slack.

We will create a robot controller for mobile robot and connect it to Raspberry Pi Mouse mobile robot component in the simulator.