## Robotics Domain Task Force Final Agenda

**OMG Technical Meeting - Cambridge, MA, USA -- June 18-22, 2012**

[http://robotics.omg.org/](http://robotics.omg.org/)

### Sunday: WG activities(pm)

<table>
<thead>
<tr>
<th>Time</th>
<th>Host</th>
<th>Joint (invited)</th>
<th>Agenda Item</th>
<th>Purpose</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>13:00</td>
<td>Robotics</td>
<td>Joint (invited)</td>
<td>Robotics submitters meeting</td>
<td>Arrangement</td>
<td>Cambridge B, 2nd FL</td>
</tr>
</tbody>
</table>

### Monday: WG activity and Robotics-DTF Plenary(am)

<table>
<thead>
<tr>
<th>Time</th>
<th>Host</th>
<th>Joint (invited)</th>
<th>Agenda Item</th>
<th>Purpose</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:00</td>
<td>Robotics</td>
<td>Joint (invited)</td>
<td>RoIS FTF final report meeting</td>
<td>Arrangement</td>
<td>William Dawes A, Lobby Lvl</td>
</tr>
<tr>
<td>09:45</td>
<td>Robotics</td>
<td>Joint (invited)</td>
<td>Revised Submission for DDC4RTC RFP Review</td>
<td>Joint with MARS</td>
<td>Molly pitcher, Lobby Lvl</td>
</tr>
<tr>
<td>12:00</td>
<td></td>
<td>Joint (invited)</td>
<td>LUNCH</td>
<td></td>
<td>Riverview Pavilion, Lobby Lvl</td>
</tr>
<tr>
<td>13:00</td>
<td></td>
<td>Joint (invited)</td>
<td>Architecture Board Plenary</td>
<td></td>
<td>Haym Saloman, Lobby Lvl</td>
</tr>
<tr>
<td>13:00</td>
<td></td>
<td>Joint (invited)</td>
<td>DDC4RTC (Robotic Infrastructure) WG(8h) - Noriaki Ando (AIST) and Seung-Woog Jung (ETRI)</td>
<td>discussion</td>
<td>William Dawes B, Lobby Lvl</td>
</tr>
<tr>
<td>14:00</td>
<td></td>
<td>Joint (invited)</td>
<td>RoboI (Robotic Functional Services) WG(8h) - Su-Young Chi (ETRI), Koji Kamei (JARA/ATR) and Toshio Hori (AIST)</td>
<td>discussion</td>
<td>William Dawes A, Lobby Lvl</td>
</tr>
</tbody>
</table>

### Tuesday: WG activity(am) and Robotics-DTF Plenary(pm)

<table>
<thead>
<tr>
<th>Time</th>
<th>Host</th>
<th>Joint (invited)</th>
<th>Agenda Item</th>
<th>Purpose</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:00</td>
<td>Robotics</td>
<td>Joint (invited)</td>
<td>DDC4RTC (Robotic Infrastructure) WG(8h) - Noriaki Ando (AIST) and Seung-Woog Jung (ETRI)</td>
<td>discussion</td>
<td>William Dawes B, Lobby Lvl</td>
</tr>
<tr>
<td>12:00</td>
<td></td>
<td>Joint (invited)</td>
<td>LUNCH</td>
<td></td>
<td>Riverview Pavilion, Lobby Lvl</td>
</tr>
<tr>
<td>13:00</td>
<td></td>
<td>Joint (invited)</td>
<td>Contact Reports: Makoto Mizukawa(Shibaura-IT), and Young-Jo Cho(ETRI)</td>
<td>Information Exchange</td>
<td></td>
</tr>
<tr>
<td>13:30</td>
<td></td>
<td>Joint (invited)</td>
<td>WG Reports and Discussion (Service WG, Infrastructure WG, Models in Robots ICS WG)</td>
<td>Presentation and discussion</td>
<td></td>
</tr>
<tr>
<td>14:00</td>
<td></td>
<td>Joint (invited)</td>
<td>Robotics-DTF Plenary Wrap-up Session</td>
<td></td>
<td>William Dawes B, Lobby Lvl</td>
</tr>
<tr>
<td>14:15</td>
<td></td>
<td>Joint (invited)</td>
<td>Adjourn joint plenary meeting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14:15</td>
<td></td>
<td>Joint (invited)</td>
<td>Robotics WG Co-chairs Planning Session (Preliminary Agenda for next TM, Draft report for Friday)</td>
<td>planning for next meeting</td>
<td></td>
</tr>
</tbody>
</table>

### Wednesday: WG activity

<table>
<thead>
<tr>
<th>Time</th>
<th>Host</th>
<th>Joint (invited)</th>
<th>Agenda Item</th>
<th>Purpose</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:00</td>
<td>Robotics</td>
<td>Joint (invited)</td>
<td>Robotics WG activity follow-up</td>
<td>Discussion</td>
<td>William Dawes B, Lobby Lvl</td>
</tr>
<tr>
<td>12:00</td>
<td></td>
<td>Joint (invited)</td>
<td>LUNCH</td>
<td></td>
<td>Charles View, 16th FL</td>
</tr>
<tr>
<td>14:00</td>
<td></td>
<td>Joint (invited)</td>
<td>Robotics WG activity follow-up</td>
<td>Discussion</td>
<td>William Dawes B, Lobby Lvl</td>
</tr>
<tr>
<td>18:00</td>
<td></td>
<td>Joint (invited)</td>
<td>OMG Reception</td>
<td></td>
<td>Empress, 14th FL</td>
</tr>
</tbody>
</table>

### Thursday: WG activity

<table>
<thead>
<tr>
<th>Time</th>
<th>Host</th>
<th>Joint (invited)</th>
<th>Agenda Item</th>
<th>Purpose</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:30</td>
<td>MARS</td>
<td>Joint (invited)</td>
<td>Joint Plenary with MARS (tentative) reserved for DCC4RTC RFP Re-Review and Voting</td>
<td>Joint with MARS</td>
<td>Molly pitcher, Lobby Lvl</td>
</tr>
<tr>
<td>09:00</td>
<td></td>
<td>Joint (invited)</td>
<td>Robotics WG activity follow-up (tentative)</td>
<td>Discussion</td>
<td></td>
</tr>
<tr>
<td>12:00</td>
<td></td>
<td>Joint (invited)</td>
<td>LUNCH</td>
<td></td>
<td>Riverview Pavilion, Lobby Lvl</td>
</tr>
<tr>
<td>13:00</td>
<td></td>
<td>Joint (invited)</td>
<td>Architecture Board Plenary</td>
<td></td>
<td>Haym Saloman, Lobby Lvl</td>
</tr>
</tbody>
</table>

### Friday

<table>
<thead>
<tr>
<th>Time</th>
<th>Host</th>
<th>Joint (invited)</th>
<th>Agenda Item</th>
<th>Purpose</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:30</td>
<td>AB, DTC, PTC</td>
<td>Joint (invited)</td>
<td></td>
<td></td>
<td>Pres Ballroom A, Lobby Lvl</td>
</tr>
<tr>
<td>12:00</td>
<td></td>
<td>Joint (invited)</td>
<td>LUNCH</td>
<td></td>
<td>Riverview Pavilion, Lobby Lvl</td>
</tr>
</tbody>
</table>

### Other Meetings of Interest

<table>
<thead>
<tr>
<th>Time</th>
<th>Host</th>
<th>Joint (invited)</th>
<th>Agenda Item</th>
<th>Purpose</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:00</td>
<td>OMG</td>
<td>Joint (invited)</td>
<td>New Attendee Orientation</td>
<td></td>
<td>Paul Revere B, Lobby Lvl</td>
</tr>
<tr>
<td>09:00</td>
<td>OMG</td>
<td>Joint (invited)</td>
<td>Introduction to OMG Specifications Tutorial</td>
<td></td>
<td>Paul Revere B, Lobby Lvl</td>
</tr>
<tr>
<td>09:00</td>
<td>Liaison ABSC</td>
<td>Joint (invited)</td>
<td>Liaison ABSC</td>
<td></td>
<td>Paul Revere B, Lobby Lvl</td>
</tr>
<tr>
<td>17:00</td>
<td>RTF-FTF Chair’s Workshop</td>
<td>Joint (invited)</td>
<td></td>
<td></td>
<td>Paul Revere B, Lobby Lvl</td>
</tr>
<tr>
<td>09:00</td>
<td>OMG</td>
<td>Joint (invited)</td>
<td>Healthcare Interoperability Workshop and Information Day</td>
<td></td>
<td>Pres Ballroom A, Lobby Lvl</td>
</tr>
<tr>
<td>09:00</td>
<td>OMG</td>
<td>Joint (invited)</td>
<td>Smart Energy Information Day</td>
<td></td>
<td>Pres Ballroom B, Lobby Lvl</td>
</tr>
<tr>
<td>09:00</td>
<td>SysA</td>
<td>Joint (invited)</td>
<td>System Assurance PTF</td>
<td></td>
<td>Haym Saloman, Lobby Lvl</td>
</tr>
</tbody>
</table>

Please get the up-to-date version from [http://staff.aist.go.jp/t.kotoku/omg/RoboticsAgenda.pdf](http://staff.aist.go.jp/t.kotoku/omg/RoboticsAgenda.pdf)
Minutes of the Robotics-DTF Meeting  
March 18-22, 2012  
Reston, VA, USA  
(robotics/2012-06-02)

Meeting Highlights
- The DDC4RTC submission was reviewed and adopted in MARS-PTF, but rejected in AB review.
- The deadline of DDC4RTC revised submission extended to the upcoming Cambridge Meeting.
- One presentation;
  - “A Trial Approach for Automation in Open Cut Mine”, Takashi Tsubouchi (Univ. of Tsukuba)

List of Generated Documents

mars/2012-02-15 Dynamic Deployment and Configuration for RTC (DDC4RTC) submission
mars/2012-02-16 DDC4RTC XMI
mars/2012-02-18 DDC4RTC Inventory
mars/2012-03-25 DDC4RTC Specification Presentation (Noriaki Ando)
mars/2012-03-26 DDC4RTC Convenient document without change bar
mars/2012-03-27 DDC4RTC Convenient document with change bar
mars/2012-03-28 DDC4RTC Errata
mars/2012-03-29 DDC4RTC XMI (revised)

robotics/2012-03-01 Final Agenda (Tetsuo Kotoku)
robotics/2012-03-02 Santa Clara Meeting Minutes [approved] (Seung-woog Jung and Koji Kamei)
robotics/2012-03-03 Opening Presentation (Tetsuo Kotoku)
robotics/2012-03-04 Roadmap for Robotics Activities (Tetsuo Kotoku)
robotics/2012-03-05 A Trial Approach for Automation in Open Cut Mine (Takashi Tsubouchi)
robotics/2012-03-06 Infrastructure WG Report (Seung-Woog Jung)
robotics/2012-03-07 Robotic Functional Services WG Report (Toshio Hori)
robotics/2012-03-08 RoIS-FTF Issues and Resolutions (Toshio Hori)
robotics/2012-03-09 Call for Paper: 3rd International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR2012)
robotics/2012-03-10 Wrap-up Presentation (Tetsuo Kotoku)
robotics/2012-03-11 DDC4RTC Specification Presentation [mars/2012-03-25] (Noriaki Ando)
robotics/2012-03-12 Next Meeting Preliminary Agenda · DRAFT (Tetsuo Kotoku)
robotics/2012-03-13 DTC Report Presentation (Noriaki Ando)
robotics/2012-03-14 Reston Meeting Minutes · DRAFT (Geoffrey Biggs and Seung-woog Jung)
Minutes

Tuesday, 21 March, 2012, Lake Fairfax B, 2nd floor.
Robotics DTF Plenary Meeting

AIST, ETRI, JARA, Univ. of Tsukuba (Quorum: 3)
6 attendees

15:00 - 17:00 Robotics DTF Opening Session, Chair: Dr. Kotoku
- Minutes takers: Seung-Woog Jung (ETRI) and Geoffrey Biggs (AIST)
- Santa Clara Meeting minutes approved
  - AIST (motion), ETRI (second), JARA (white ballot)

15:00 - 15:10 Brief summary of Santa Clara meeting
- 18 participants
- 3 talks
- 2 WG reports
- Deadline of DDC4RTC revised submission was extended to the Reston meeting.
- Final report of RTC RTF was accepted.

15:10 - 15:55 Talk: A Trial Approach for Automation in Open Cut Mine
Takashi Tsubouchi (University of Tsukuba)
- Limestone mining in Japan.
- Rocks from blasting must be broken up using hydraulic breakers so they will fit into the crusher.
  - Breaking up the rocks in the bulldozing chamber is done by remote operation.
  - The operator needs to move the rocks using the chisel into a good position for breaking. This is grasp-less manipulation, which is an interesting challenge for robotics.
- Studies:
  - Manipulation of rocks at the work face using a 1/12 scale electric model.
  - Detection of plugging of the grizzly bars using video cameras.

15:55 - 16:20 Infrastructure working group report (DDC4RTC)
Seung-Woog Jung, ETRI
- Submit merged document after last meeting.
- Received comments from AB members (Steve and Elisa).
- Reviewed comments on Monday and assigned tasks to make changes based on comments.
- On Tuesday, the changes were merged.
- If MARS or the AB rejects DDC4RTC, the specification will be terminated.
  - If the AB rejects with positive comments, then it will be re-submitted.

16:20 - 16:45 Service working group report (RoIS)
Toshio Hori (JARA)
- A private meeting was held in Seoul in February, 2012.
  - 26 issues were raised, 15 were resolved.
- One new issue was received by the comment deadline (2012/02/20).
- Only two members attended the OMG meeting, limiting discussions.
- Schedule from now is to present the final draft and FTF report by the next meeting deadline (21 May 2012).
- Another private meeting is planned in Seoul in May, one week before the deadline.
Deadlines:
  - Report due date: May 21, 2012
  - Report deadline: June 29, 2012

16:45 - 17:00 Contact reports
  - SIMPAR 2012, Noriaki Ando (AIST)
  - Nov 5-8, 2012, Tsukuba, Japan

16:00 - 16:30 Robotics DTF Wrap-up Session, Chair: Dr. Kotoku
  - Robotics-DTF Co-Chair: postpone voting one more meeting
  - Schedule for next meeting

ATTENDEE (6 attendees):
  ● Noriaki Ando (AIST)
  ● Geoffrey Biggs (AIST)
  ● Toshio Hori (JARA/AIST)
  ● Seung-Woog Jung (ETRI)
  ● Tetsuo Kotoku (AIST)
  ● Takashi Tsubouchi (Univ. of Tsukuba)

Prepared and submitted by Geoffrey Biggs (AIST) and Seung-Woog Jung (ETRI)
Robotics-DTF Plenary Meeting
Opening Session

June 19, 2012
Cambridge, MA, USA
Hyatt Regency Cambridge

Approval of Minutes

Meeting Quorum : 3
AIST, ETRI, JARA, Univ. of Tsukuba, Sibaura-IT

Minutes taker(s):
• Geoffrey Biggs
• Seung-Woog Jung

Minutes review
Reston Meeting Summary

Robotics Plenary: (6 participants)

- The DDC4RTC submission was reviewed and adopted in MARS-PTF, but rejected in AB review.
- The deadline of DDC4RTC revised submission extended to the upcoming Cambridge Meeting.
- One presentation;
  “A Trial Approach for Automation in Open Cut Mine”, Takashi Tsubouchi (Univ. of Tsukuba)
# Roadmap for Robotics Activities

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Flyer of Robotics-DTF [Publicity Sub-Committee]</td>
<td>Suspended</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic Deployment and Configuration for RTC (DDC4RTC) RFP [Robotic Infrastructure WG] in MARS</td>
<td>In Process</td>
<td></td>
<td>In Process</td>
<td>Revised Submission &amp; Voting</td>
<td></td>
<td></td>
<td>Sponsor: MARS</td>
<td></td>
</tr>
<tr>
<td>Robotic Map Services RFP [Robotic Functional Services WG]</td>
<td>Planned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IEEE R&amp;A?</td>
<td></td>
</tr>
<tr>
<td>etc…</td>
<td>Future</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robotics Exhibition</td>
<td>Planned</td>
<td></td>
<td></td>
<td>Exhibition</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RoIS Finalization Task Force</td>
<td>In Process</td>
<td></td>
<td></td>
<td>FTF Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RoIS 1.1 Revision Task Force</td>
<td>Planned</td>
<td></td>
<td></td>
<td>Charter</td>
<td></td>
<td></td>
<td>FTF Report</td>
<td></td>
</tr>
<tr>
<td>DDC4RTC Finalization Task Force</td>
<td>Planned</td>
<td></td>
<td></td>
<td>Charter</td>
<td></td>
<td></td>
<td>FTF Report</td>
<td></td>
</tr>
<tr>
<td>RTC 1.2 or 2.0 Revision Task Force</td>
<td>Planned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLS 1.2 or 2.0 Revision Task Force</td>
<td>Planned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>will go to ISO/TC211</td>
<td></td>
</tr>
</tbody>
</table>

Related Events

- Chu-suk (Special Holidays in Korea)
Contact Report
Standardization of RTC-CANopen in CiA

2012/6/19
Makoto Mizukawa
Dean, College of Engineering
Professor, Human-Robot-Interaction Lab.
Department of Electrical Engineering,
College of Engineering
Shibaura Institute of Technology

3-7-5, Toyosu, Koto-ku, Tokyo 135-8548

Acknowledgement
This work is supported by NEDO
(New Energy and Industrial Technology Development Organization, Japan) project
“Intelligent RT Software Project”.

Robotic Technology Component Specification

- Standardized in Object Management Group (OMG).
  - Ver.1.0 was opened to the public in April, 2008.
  - Specify a basic part of RT-Component.
    - RT-Component Interfaces
    - Basic state transition
CANopen

- Standardized upper protocol based on Controller Area Network (CAN) that suits embedded network for machine control.
- CANopen decreases message overhead.

7. Application
   - CANopen application standards

6. Presentation
   - Partly implemented by upper protocol like CANopen

5. Session

4. Transport

3. Network

2. Data Link
   - Classical CAN implementation

1. Physical

Bypass without upper protocol

---

CANopen

- Application and communication are independent of each other by Object Dictionary that is a kind of data table.
- Some data defined in the profile specialized device and application.

SDO: Service Data Object=2 data frames+ID
PDO: Process Data Object
RTC-CANopen

RT-Middleware
- RT-Middleware can improve software reusability.
- It’s difficult to use RT-Component on embedded MPU, because RT-Component needs CORBA environment.

CANopen
- CAN-based upper protocol for embedded network
- CANopen can improve hardware reusability, because this protocol is standardized.

Development RT-Middleware for embedded system

RTC-CANopen

---

State Machine Mapping:

Power On Hardware Reset

Initializing
Reset Application
Reset Communication
Pre-Operational
Operational
Stopped

CANopen:
- Initializing
- Pre-Operational
- Operational
- Stopped
- Reset Application
- Reset Communication

RT-Component:
- onInitialize
- onActivated
- onExecute
- onDeactivated
- onAborting
- onReset
- onError
- onFinalize

Process:
- Initialization Invoked only once when the RTC starts.
- Invoked only once when the INACTIVE RTC is activated.
- Invoked periodically while in the ACTIVE state.
- Invoked only once when the ACTIVE RTC is deactivated.
- Invoked only once before the RTC goes to the ERROR state.
- Invoked when the RTC recovers from the ERROR state to the INACTIVE state by RESET.
- Invoked periodically while in the ERROR state.
- Invoked only once when the RTC finalizes.

---

System configuration for RTC-CANopen

A system constructed in RTC-CANopen consists of the following:

- **Device RTC**: RTC running on embedded MPU based on Slave Node of CANopen.
- **Proxy RTC**: RTC running on general-purpose PC, accessing to Device RTC virtually.
- **RTC-CANopen Servers**: RTC managing the whole network of state transition of Device RTC and Proxy RTC.
  (PhP Manager, Status Manager, RTC-CANopen Manager)

---

**SHIBAURA INSTITUTE OF TECHNOLOGY**

Advantage of RTC-CANopen

- Available to convert existing hardware for robots.
  - Available to use existing CANopen compliant devices in robot systems.
  - Reduction of a hardware cost.

- Available to run in many embedded MPU
  - All MPU that can implement CANopen.
    - M16C, R8C, H8S, SH series, V850, PIC18Fxx8, and so on.

- Improvement of reliability of communication between components
  - High reliability and high speed communication can be realized.
System Development using RTC-CANopen

-PnP system for RTC-CANopen-

RTC-CANopen

RTC-CANopen is the system that can manage CANopen devices on RT-Middleware.

RTC-CANopen Server manages this system.
-RTC-CANopen Manager
-PnP Manager
-Status Manager

RTC-CANopen Manager
-Detects the device boot-up.
-Registered device information is confirmed from PnP Manager.
-Invokes the ProxyRTC.
RTC-CANopen Standardization

- Existing CANopen Devices can be used as RTC
- Various application fields from Robotics to Industries
- Increase of robotics devices with lower cost

- 2008 Starts negotiation with CiA(CAN in Automation) in Brussels OMG TM
- 2010.6 Introduction of RTC-CANopen in CAN Newsletter
- 2010.7 CFE(Call for Expert) from CiA to establish Service Robot SIG
- 2010.11 CiA Kick off meeting in Tokyo
- 2011.3 Service Robot SIG starts (Chair Prof. Mizukawa(SIT))
- 2011.11 Working Draft issued
  CiA318:Implementation guideline - Mapping of RTC to CANopen
  CiA460:Service robot controller profile - NMT master application and CANopen device proxies
- 2012.2 DSP(Draft Standard Proposal) issued

CiA DSP(Draft Standard Proposal) issued on 2012/2/10
Robotic Functional Service WG
WG Report

WG Co-Chair: Toshio Hori
2012/06/19

WG activities before this meeting

• At Seoul private Meeting (2012/05)
  – Discussed issues for RoIS Framework
    • Attendants: Chi(ETRI), Cho(ETRI), Kamei(ATR), Nishio(ATR), Hori(AIST)
  – 27 issues were raised and discussed.
    • 20 issues were resolved.
    • 3 issues were deferred. 2 closed w/o change and 2 merged with others.

• We hold 3 polls in April and May
  – All the issues were passed as proposed (20 resolved, 3 deferred, 2 closed and 2 merged)
  – During the polls, Sam (Sparx) was removed from FTF member because he left Sparx and, hence, quitte
WG activities during this meeting

• Monday morning
  – Prepare for AB in the afternoon.

• AB in Monday afternoon
  – Presented FTF report
  – Resolutions to the comments from AB reviewer were presented
  – Several problems of XML PSM were pointed out by Tom.

• Tuesday morning
  – Problems of XML PSM were resolved by Kamei and emailed to Tom and Andrew.
  – Working with beta document to resolve AB comments.

Schedule hereafter

• Post a revised beta document and related materials by Thursday.

• Present our resolutions at AB in Thursday afternoon (if we can keep our schedule above mentioned)
  – If we can’t, RoIS FTF report and beta document will be voted by email at AB.

• Deadlines (indicated in the charter):
Infrastructure WG Progress Report

Infra. WG, Robotics DTF
Noriaki Ando, AIST
robotics/2012-06-07

Overview

**DDC4RTC**
- Submitted in February
  - 64 issues from AB comments/All resolved
  - Approved in MARS/Rejected in AB
    - Too much modification from 4w document
- Revised specification submitted in May
  - 15 (+4) comment from AB (and remedy IT)
  - 14 (+3) issues are resolved
  - Other issues are working state
Comments from AB and Remedy IT

- Comments are returned from
  - Manfred Koethe
  - Johnny Willemsen (Remedy IT)

- Comment X-Y
  - X: {Manfred => 1, Johnny => 2}
  - Y: Comment number

- Resolution
  - → Green text: Resolved issue
  - → Red text: Unresolved issue

Comment 1-1

- The Conformance section needs improvement. In the current form, it is too unclear. I think you could just state that everything in this specification must be implemented.

- → Revised text:
  This specification extends the Configuration and Deployment of Component-based Applications (DEPL) specification to achieve dynamic configuration and deployment. This specification provides four independent conformance points, each of which corresponds to one conformance point from the DEPL specification: RepositoryManager, TargetManager, NodeManager and ExecutionManager. Implementing a conformance point of the DDC4RTC specification requires implementing the corresponding conformance point of the DEPL specification.
Comment 1-2

- Section 4 to 7 contain several spelling and formatting mistakes. See annotated PDF for details.

- → Typos have been corrected.

Comment 1-3

- Figure 7.1, the state chart, the transition conditions should be directly on the transitions (as guards) instead in comments.

- → Transition conditions described as comments are replaced guards and actions in the diagram.
Comment 1-4

- In figure 8.1, the package diagram, the intended meaning of the dependencies is «import», right? Also, the two comments on the right need better explanations. Since this is a UML Package diagram, the mentioning of XML is a bit out-of-place.

- → «<import>> prototype has been added to relations between packages. Notes on the diagram have been removed.

Comment 1-5

- in figure 8.2, Component Data Model, the dependencies, while not wrong, are not needed as the types are used in the Property definitions of RTCImplementationDefinition

- → Arrows between enumerations and classes have been removed from Figure 8.2.
Comment 1-6

• In 8.2.3 and 8.2.4: EVENTDRIVEN vs. EVENT_DRIVEN ... which is right?

• → “EVENTDRIVEN” is wrong (This keyword came from RTC specification). All the EVENTDRIVEN have been modified to EVENT_DRIVEN

Comment 1-7

• Figure 8.3: The diagram is a bit confusing.

• → This would be improved in FTF.
Comment 1-8

- 8.4: "SupervisorFSM" is everywhere spelled with lower-case "ν", except in the model, where it is spelled "SuperVisorFSMDescription"

  → SuperVisor in Figure 8.10 and Figure 8.12 are modified to Supervisor.

Comment 1-9

- In figure 8.6, again, the dependencies are not needed. And following that diagram, the types in the descriptions are bold, should be normal font. (also in description after figure 8.7)

  → Arrows between enumerations and class were removed from Figure 8.6. The types in the descriptions was modified to normal font.
Comment 1-10

- 8.4.2 and 8.4.3 “Behavior” is constantly misspelled in text (but correct in model)
  - Corrected from “Bihavior” -> “Behavior”

Comment 1-11

- In figure 8.9, the multiplicities at the black diamonds are strange. I guess [0..1] was intended.
  - The multiplicities in Figure 8.9 have been corrected from 0 -> 0..1.
Comment 1-12

• In the description after figure 8.14, the return type for “search(query:String” is in CORBA notation, should be RTCInstanceDeploymentDescription[0..*]

• → In Figure 8.14, “DeploymentPlan” has been modified to “RTCInstanceDeploymentDescription[0..*]”
• Sequence<RTCInstanceDeploymentDescription → RTCInstanceDeploymentDescription [0..*]

Comment 1-13

• Annex A.1 and A.2 (the xml) should be reformatted for better readability

• →
Comment 1-14

• The XMI file uses Japanese encoding (SHIFT JIS) and contains Enterprise Architect-specific material.

• → XMI file has been recreated.

Comment 1-15

• In the XMLSchema in mars-12-05-06, it lacks the import of the underlying DEPL schema, so several base types are undefined. (ComponentImplementationDescription, ComponentPortDescription, SubcomponentPortEndpoint) A similar problem exists with State and StateMachine in the XMLSchema in mars-12-05-07

• → This issue will be resolved in the FTF.
Comment 2-1

- About dependency on the CORBA notification service. That is only used for the event mechanism for which for example DDS could also be used. Maybe in some way this spec can be made more flexible in terms of the service delivering the event mechanism?

- → This would be discussed in the next revision of DDC4RTC.

Comment 2-2

- Section 8.2.8 describes the RTC Component Ports, be aware that CCM has also this support and that we didn’t had to extend DnC for this.

- → We could not find pertinent section in the CCM 4.0 specification.
Comment 2-3

- Section A.0 lists "interface Table1 DirectoryManage" which is not legal IDL.
- The interface “DirectoryManager” has been modified.

Comment 2-4

- I can't give concrete feedback on the parts related to RTC but I do see that some of the limitations we see in CCM4 are solved in some way.
- We would like to discuss referring CCM4 for next revision of DDC4RTC and new DEPL (or Dynamic Deployment) specification. We hope we (robotics DTF) could make some feedback to new DEPL specification from our experience about DDC4RTC.
## Document Numbers

- Dynamic Deployment & Config for RTC  mars/12-05-02
- DDC4RTC.xmi  mars/12-05-04
- DDC4RTC.idl  mars/12-05-05
- ComponentDataModel.xsd  mars/12-05-06
- ExecutionDataModel.xsd  mars/12-05-07
- Inventory file  mars/12-05-03

- Errata  mars/12-06-09
- DDC4RTC with change-bar  mars/12-06-10
- DDC4RTC without change-bar  mars/12-06-11
- Updated XMI file  mars/12-06-12
- Updated IDL file  mars/12-06-13
- Inventory file

## Future Plan

- Voting will be done in MARS plenary on Thursday morning
- And we will go to AB Thursday afternoon
  - We hope we could pass AB and start FTF
Robotics-DTF Plenary Meeting

Wrap-up Session

June 19, 2012
Cambridge, MA, USA
Hyatt Regency Cambridge

Document Number

robotics/2012-06-01 Final Agenda (Tetsuo Kotoku)
robotics/2012-06-02 Reston Meeting Minutes [approved] (Geoffrey Biggs and Seung-woog Jung)
robotics/2012-06-03 Opening Presentation (Tetsuo Kotoku)
robotics/2012-06-04 Roadmap for Robotics Activities (Tetsuo Kotoku)
robotics/2012-06-05 Contact Report: Standardization of RTC-CANopen in CiA (Makoto Mizukawa)
robotics/2012-06-06 Robotic Functional Service WG Report (Toshio Hori)
robotics/2012-06-07 Infrastructure WG Progress Report (Noriaki Ando)
robotics/2012-06-08 Wrap-up Presentation (Tetsuo Kotoku)
robotics/2012-06-09 Next Meeting Preliminary Agenda - DRAFT (Tetsuo Kotoku)
robotics/2012-06-10 DTC Report Presentation (Tetsuo Kotoku)
robotics/2012-06-11 DDC4RTC-FTF Charter (Noriaki Ando)
robotics/2012-06-12 Cambridge Meeting Minutes - DRAFT (Geoffrey Biggs and Seung-woog Jung)
Call for volunteer

- Robotics-DTF Co-Chair

=> Postpone voting one more meeting
Next Meeting Agenda (no meeting)
September 10-14 (Jacksonville, FL, USA)

No Robotics-DTF Plenary Meeting

After the Next Meeting Agenda
December 10-14 (Burlingame, CA, USA)

Tuesday:
Robotics Information Day

Wednesday:
WG activity [Parallel WG Session] (am)
Robotics-DTF Plenary Meeting (pm)
• Guest and Member Presentation
• Contact reports
Plenary Attendee (10 participants)

- Geoffrey Biggs (AIST)
- Makoto Mizukawa (Shibaura-IT)
- Noriaki Ando (AIST)
- Seung-Woog Jung (ETRI)
- Su-Young Chi (ETRI)
- Takashi Tsubouchi (Univ. of Tsukuba)
- Tetsuo Kotoku (AIST)
- Toshio Hori (JARA/AIST)
- Tsuyoshi Kamei (JARA/ATR)
- Young-Jo Cho (ETRI)
### Robotics Domain Task Force Preliminary Agenda

**OMG Technical Meeting - Jacksonville, FL, USA -- September 10-14, 2012**

**Host**: Joint (Invited)

<table>
<thead>
<tr>
<th>Time</th>
<th>Agenda Item</th>
<th>Purpose</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:00</td>
<td><strong>No Robotics-DTF Meeting</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**OMG Technical Meeting - Burlingame, CA, USA -- December 10-14, 2012**

<table>
<thead>
<tr>
<th>Time</th>
<th>Agenda Item</th>
<th>Purpose</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:00</td>
<td><strong>No Robotics-DTF Meeting</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Please get the up-to-date version from http://staff.aist.go.jp/t.kotoku/omg/RoboticsAgenda.pdf
Highlights from this Meeting:

Dynamic Deployment and Configuration for RTC (DDC4RTC) Submission (sponsored by MARS)
- mars/12-05-02 (Specification Document)
- mars/12-06-12 (XMI)
- mars/12-06-13 (IDL)
- mars/12-05-06 (ComponentDataModel XML schema)
- mars/12-05-07 (ExecutionDataModel XML schema)
- mars/12-06-23 (Inventory)

Robotics Plenary: (10 participants)
- 2 WG Reports
  - Robotic Infrastructure WG [robotics/2012-06-07]
  - Robotic Functional Services WG [robotics/2012-06-06]

Future deliverables (In-Process):
No Plan

Next Meeting (in Jacksonville):
No Robotics-DTF meeting

After the Next Meeting (in Burlingame):
- Robotics Information day
- Several Exhibitions related to RTC, RLS, RoIS
Proposed Charter for Dynamic Deployment and Configuration for Robotic Technology Component (DDC4RTC) 1.0 FTF

- **Adopted Specification:**
  - Mars/12-05-02 (Specification Document)
  - Mars/12-06-12 (XMI)
  - Mars/12-06-13 (IDL)
  - Mars/12-05-06 (ComponentDataModel XML schema)
  - Mars/12-05-07 (ExecutionDataModel XML schema)
  - Mars/12-06-23 (Inventory)

- **Members:**
  - Seung-Woog Jung (Co-Chair), ETRI
  - Noriaki Ando (Co-Chair), AIST
  - Takashi Suehiro, UEC
  - Takeshi Sakamoto, Shibaura Institute of Technology
  - Takashi Tsubouchi, University of Tsukuba
  - Geoffrey Biggs, JARA
  - HongSeong Park, Kangwon National University
  - Chuljong Hwang, KAR
  - Manfred R. Koethe, 88solutions

- **Deadlines:**
  - Beta Specification Publication: 22nd July, 2012
  - Comments Due: 18th February, 2013
  - Report Due Date: 20th May, 2013
  - Report Deadline: 28th June, 2013

OMG FTF Charter, Version 1.5, omg/2008-01-02
Minutes of the Robotics Domain Task Force Meeting - DRAFT
June 18-22, 2012
Cambridge, MA, USA
(robotics/2012-06-12)

Meeting Highlights
- The DDC4RTC submission was reviewed and adopted in MARS-PTF and AB.
- The DDC4RTC-FTF charter was adopted.
- The RoIS-FTF Final Report was reviewed in AB, but a vote by e-mail will be held on next Monday. (There is no time to review the revised report)

List of Generated Documents

mars/2012-05-06 ComponentDataModel.xsd
mars/2012-05-07 ExecutionDataModel.xsd
mars/2012-06-08 DDC4RTC Presentation on Mon. (Noriaki Ando)
mars/2012-06-09 Errata
mars/2012-06-10 DDC4RTC with change-bar
mars/2012-06-11 DDC4RTC without change-bar
mars/2012-06-12 Updated XMI file
mars/2012-06-13 Updated IDL file
mars/2012-06-23 Updated Inventory file
mars/2012-06-26 DDC4RTC Presentation on Thu. (Noriaki Ando)

dtc/2012-05-28 RoIS FTF Report (Toshio Hori)
dtc/2012-06-25 Updated RoIS FTF Report
dtc/2012-06-26 Updated Inventory file
dtc/2012-06-27 Updated Beta 2 document without change bars
dtc/2012-06-28 Updated Beta 2 document with change bars
dtc/2012-06-29 Updated C++ PSM header file
dtc/2012-06-30 Updated CORBA PSM IDL file
dtc/2012-06-31 Updated XML PSM schema file
dtc/2012-06-32 Updated XMI file
dtc/2012-06-33 Errata

robotics/2012-06-01 Final Agenda (Tetsuo Kotoku)
robotics/2012-06-02 Reston Meeting Minutes [approved] (Geoffrey Biggs and Seung-woog Jung)
robotics/2012-06-03 Opening Presentation (Tetsuo Kotoku)
robotics/2012-06-04 Roadmap for Robotics Activities (Tetsuo Kotoku)
robotics/2012-06-05 Contact Report: Standardization of RTC-CANopen in CiA (Makoto Mizukawa)
robotics/2012-06-06 Robotic Functional Service WG Report (Toshio Hori)
robotics/2012-06-07 Infrastructure WG Progress Report (Noriaki Ando)
robotics/2012-06-08 Wrap-up Presentation (Tetsuo Kotoku)
robotics/2012-06-09 Next Meeting Preliminary Agenda - DRAFT (Tetsuo Kotoku)
robotics/2012-06-10 DTC Report Presentation (Tetsuo Kotoku)
robotics/2012-06-11 DDC4RTC-FTF Charter (Noriaki Ando)
robotics/2012-06-12 Reston Meeting Minutes - DRAFT (Geoffrey Biggs and Seung-woog Jung)
Minutes

Tuesday, 19 June, 2012, William Dawes B, Lobby Lvl
Robotics DTF Plenary Meeting
AIST, ETRI, JARA, Shibaura-IT, Univ. of Tsukuba, (Quorum: 3)
10 attendees

13:00 - 13:10 Robotics-DTF Opening Session,
Chair: Tetsuo Kotoku (AIST)
- Minutes takers: Seung-Woog Jung (ETRI) and Geoffrey Biggs (AIST)
- Brief summary of Reston meeting
  - 6 participants
  - 1 special talk
  - 2 WG reports
  - DDC4RTC passed MARS but was rejected by the AB due to too many changes to the submission.
  - Deadline was extended one meeting.
- Reston Meeting minutes (robotics/2012-06-02) approved
  - AIST (motion), Shibaura-IT (second), Univ. of Tsukuba (white ballot)

13:10 - 13:25 Contact report: Standardization of RTC-CANopen in CiA,
Makoto Mizukawa (Shibaura-IT)
- RT-Component standardized by the OMG in 2008.
- CANopen is a widely-used technology.
- RTC-CANopen is an RT-Middleware for embedded systems.
- Advantages:
  - Existing CANopen devices in robotics can be used.
  - Many MPUs support CAN.
  - CANopen is already known to be quite reliable.
- 2008: Start negotiation with CiA (CAN in Automation)
- 2011.11 Working draft issued
  - CiA 318: Implementation guideline - Mapping of RTC to CANopen
  - CiA 460: Service robot controller profile
  - NMT master application and CANopen device proxies
- 2012.02 DSP (Draft Standard Proposal) issued

13:25 - 13:35 Contact report: ISO TC 184 WG meetings, Politecnico di Milano, Italy,
Su-Young Chi (ETRI)
- ISO TC 184/WG 8 (Service robots), July 11-12, 2012
- RoIS is one of the candidate of NWIP

13:35 - 13:45 Working group report: Robotic Functional Service WG,
Toshi Hori (AIST)
- A meeting was held in Seoul in May to work on the remaining issues for the RoIS framework.
  - 5 participants
  - 27 issues raised and discussed
  - 20 resolved
  - 3 deferred
  - 2 closed without changed
  - 2 merged with others
- Three polls were held in April and May.
- Sam (Sparx) was removed from the FTF due to his leaving Sparx and OMG activities.
- WG activities during this meeting
  - Attended the AB on Monday afternoon. Several problems were pointed out in the XML PSM. These have been resolved.
  - A revised beta document will be posted by Thursday.
  - The revised document will go to a vote at the AB on Thursday afternoon, or, if the revised document is not ready in time, by an email vote.
- FTF report deadline is 29 June, 2012.

13:45 - 13:50 Working group report: Infrastructure WG, Noriaki Ando (AIST)
- Activities at the Reston meeting:
  - 64 issues from the AB comments. All were resolved.
  - Approved in MARS
  - Rejected by the AB: Too many modification from the 4-week document
  - Revised specification submitted in May
    - 15 comments from the AB review and 4 from MARS (Remedy IT)
    - 14 AB comments and 3 MARS comments are resolved. 2 issues remain.
  - Vote will be held in MARS on Thursday morning. If passed, then it will go to the vote in the AB on Thursday afternoon.

13:50 - 16:00 Robotics DTF Wrap-up Session, Chair: Tetsuo Kotoku (AIST)
- Robotics-DTF Co-Chair selection: postpone one more meeting
- Roadmap discussion.
  - An exhibition and Information day will be held at the December meeting presenting products developed based on the OMG’s robotics standards.
  - No meeting will be held in September.

ATTENDEE (10 attendees):
- Seung-Woog Jung (ETRI)
- Su-Young Chi (ETRI)
- Young-Jo Cho (ETRI)
- Tetsuo Kotoku (AIST)
- Noriaki Ando (AIST)
- Geoffrey Biggs (AIST)
- Toshio Hori (AIST)
- Koji NAMEI (ATR)
- Takashi Tsubouchi (Univ. of Tsukuba)
- Makoto Mizukawa (Shibaura-IT)

Prepared and submitted by Geoffrey Biggs (AIST) and Seung-Woog Jung (ETRI)
Robotic Interaction Service (RoIS) Framework

*FTF - Beta 2*

OMG Document Number: dtc/2012-06-27

Standard document URL:  [http://www.omg.org/spec/RoIS/1.0/](http://www.omg.org/spec/RoIS/1.0/)

Associated File(s)*:  

* original file(s): robotics/2011-05-02, 2011-05-03

This OMG document replaces the submission document (robotics/2011-05-01, Alpha). It is an OMG Adopted Beta Specification and is currently in the finalization phase. Comments on the content of this document are welcome, and should be directed to issues@omg.org by February 20, 2012.

You may view the pending issues for this specification from the OMG revision issues web page [http://www.omg.org/issues/](http://www.omg.org/issues/).

The FTF Recommendation and Report for this specification will be published on June 29, 2012. If you are reading this after that date, please download the available specification from the OMG Specifications Catalog.
USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical,
including photocopying, recording, taping, or information storage and retrieval systems—without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™, Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™, MOF™, OMG Interface Definition Language (IDL)™, and OMG SysML™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials. Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software
developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.
OMG’s Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue (http://www.omg.org/technology/agreement.)
Table of Contents

Preface ................................................................................................................................. 9
Overview ............................................................................................................................... 11
1 Scope ................................................................................................................................. 14
2 Conformance ...................................................................................................................... 14
3 References ........................................................................................................................ 14
   3.1 Normative References ............................................................................................... 14
   3.2 Non-Normative References ...................................................................................... 15
4 Terms and Definitions ....................................................................................................... 15
5 Symbols ................................................................................................................................ 16
6 Acknowledgements ......................................................................................................... 16
7 Platform Independent Model ............................................................................................ 17
   7.1 Format and Conventions ........................................................................................... 17
      7.1.1 Class and Interface ............................................................................................ 17
      7.1.2 Enumeration ...................................................................................................... 17
      7.1.3 Message .............................................................................................................. 18
      7.1.4 HRI Component and method ......................................................................... 18
   7.2 Structure of the RoIS Framework ............................................................................. 18
   7.3 Return Codes .............................................................................................................. 21
   7.4 RoIS Interface ............................................................................................................ 21
      7.4.1 Interaction .......................................................................................................... 21
      7.4.2 Interfaces .......................................................................................................... 30
      7.4.3 Message Data ................................................................................................... 34
   7.5 Profiles ....................................................................................................................... 39
      7.5.1 Overview ............................................................................................................ 39
      7.5.2 Parameter Profile ............................................................................................. 40
      7.5.3 Message Profile ................................................................................................. 41
7.5.4 HRI Component Profile ................................................................. 42
7.5.5 HRI Engine Profile ................................................................. 43
7.6 Common Messages ................................................................. 43
7.6.1 System Information .............................................................. 45
7.6.4 Person Identification ............................................................ 49
7.6.5 Face Detection ................................................................. 51
7.6.6 Face Localization ............................................................... 52
7.6.7 Sound Detection .............................................................. 53
7.6.8 Sound Localization ............................................................. 54
7.6.9 Speech Recognition ............................................................. 56
7.6.10 Gesture Recognition ........................................................ 57
7.6.11 Speech Synthesis ............................................................... 58
7.6.12 Reaction ............................................................................ 60
7.6.13 Navigation ....................................................................... 61
7.6.14 Follow .............................................................................. 63
7.6.15 Move ............................................................................... 64
7.7 Platform Specific Model ........................................................... 66
7.7.1 C++ PSM ........................................................................ 66
7.7.2 CORBA PSM ................................................................. 76
7.7.3 XML PSM ....................................................................... 86
Annex A Examples of Profile in XML (informative) ....................... 89
A.1 Parameter Profile ................................................................. 89
A.2 Message Profile ................................................................. 89
A.2.1 Command Message Profile ............................................... 89
A.2.2 Event Message Profile ....................................................... 90
A.2.3 Query Message Profile ..................................................... 90
A.3 HRI Component Profile ......................................................... 91
A.4 HRI Engine Profile ............................................................. 92
Annex B Examples of CommandUnitSequence in XML (informative) .............. 95
B.1 CommandUnitSequence ........................................................ 95
B.2 CommandMessage .............................................................. 96
Annex C  Examples of User-Defined HRI Component (informative) ..........................98
  C.1  Speech Recognition (W3C-SRGS) ........................................................................... 98
  C.2  Person Gender Identification ................................................................................... 99
  C.3  Person Age Recognition ......................................................................................... 100

Annex D  Examples of Data Type (informative) .................................................................101
  D.1  Reaction Type .......................................................................................................... 101
Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG’s specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG’s specifications include: UML® (Unified Modeling LanguageTM); CORBA® (Common Object Request Broker Architecture); CWMTM (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A Specifications Catalog is available from the OMG website at:


Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM)
Platform Specific Model and Interface Specifications

CORBA services
CORBA facilities
OMG Domain specifications
OMG Embedded Intelligence specifications
OMG Security specifications

All of OMG’s formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA

Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: pubs@omg.org

Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

Helvetica/Arial - 10 pt: Exceptions

Note – Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.
Overview

Figure 1: Example of robot scenario for robotic reception service. Events delivered from sensors, actuators or other event sources, such as an internal timer, to a service application trigger each state transition and the application controls the robot according to the scenario.

Many service-robot applications prepare robot scenarios like the one shown in Figure 1. Such a scenario describes an application that controls robot behavior after the output from a variety of sensors embedded in the robot or the environment triggers a transition in the state of the robot. Figure 1 shows an example of a robot scenario for a robotic reception service. In this scenario, events like “detect a person” and “identify the person” or obtained information like “person ID” and “position of the person” act as state-transition triggers while commands like “approach the person” and “tell the message” determine what the robot is to do next. Of importance here is that state-transition triggers and commands in the robot scenario are not described on the physical level (hardware layer) as in sensors and movement mechanisms in the robot but rather on the symbol level (symbolic layer) as in “person detection” and “person identification.”

At present, however, the service-robot developer and service application programmer is often one and the same (individual or group) and applications like the one shown in Figure 2 are optimized by directly accessing the hardware layer. As a result, any changes made to the hardware mechanism make it necessary to revise the application to accommodate those changes. It is essential that this problem be solved for the sake of improving the reusability of applications and expanding the market for service robots.

To make the above development of service-robot applications more efficient, this specification defines a new framework that abstracts and unifies the various types of components that are possibly implemented by RTC [RTC] or ROS, and the human-robot interaction service functions provided by the robot as shown in Figure 3.
Figure 2: Conventional style of service application programming. Service application programmer must write service application programs for each robot independently because functions provided by each robot are different.

Figure 3: RoIS service application programming style. The same service application program works on different robot platforms with little modification.
Proof of Concept

This specification is based on our extensive surveys on human-robot interaction function methodologies and implementations, which are currently used in robotic products and research projects in Japan and Korea. Members from 12 organizations in Japan and 3 organizations in Korea joined in composing the document. All of them have rich research and/or production experiences in the field of robotics, especially of service robots working in domestic environments or indoor environments such as shopping malls, airports and hospitals.
Part I

1 Scope

This specification defines a framework that can handle messages and data exchanged between human-robot interaction service components and service applications. It includes a platform-independent model (PIM) of the framework.

2 Conformance

Any implementation or product claiming conformance to this specification shall support the following conditions:

- Implementations shall provide interfaces described in “Section 7.4 RoIS Interface”.
- Implementations shall support the return codes described in “Section 7.3 Return Codes”.
- Implementations shall support the common messages described in “Section 7.6 Common Messages”. This does not mean that the module shall include every common messages described herein. However, every module should support the common messages when the module use the basic components listed in “Section 7.6 Common Messages”.
- Data structure of messages treated by implementations shall support the profile described in “Section 7.5 Profiles”

3 References

3.1 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.


[ISO639] International Organization for Standardization, Codes for the representation of names of languages

[ISO19111] International Organization for Standardization, Geographic information - Spatial referencing by coordinates, 2007


[ISO19143] International Organization for Standardization, Geographic information - Filter encoding, 2010

[ISO19784] International Organization for Standardization, Biometric application programming interface, 2006

[RLS] Object Management Group, Robotic Localization Service (RLS), Version 1.0, 2010

3.2 Non-Normative References


4 Terms and Definitions

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic HRI Component</td>
<td>An HRI Component which provides a basic HRI function of service robots, where “basic HRI function” means “an HRI function implemented in many (but not all) service robots.” 15 Basic HRI Components and their interfaces are defined in this document.</td>
</tr>
<tr>
<td>Detection</td>
<td>A function that finds target objects, such as persons and faces, and returns the number of the objects found. When the function can detect only existence or non-existence of the target, the number shall be provided in only two states, i.e. one and zero.</td>
</tr>
<tr>
<td>HRI</td>
<td>Abbreviated form of “Human-Robot Interaction”</td>
</tr>
<tr>
<td>HRI Component</td>
<td>An object which uses sensors or actuators to provide a specific HRI function, such as person detection, person identification or speech. An HRI Component may be implemented as a software object or an aggregate of multiple objects, while such internal structure is encapsulated.</td>
</tr>
<tr>
<td>HRI Engine</td>
<td>An object that manages HRI Components. It mediates Human-Robot Interaction functions of the HRI Components to Service Application(s).</td>
</tr>
<tr>
<td>Identification</td>
<td>A function that finds target objects and returns a list of identifiers of objects found.</td>
</tr>
<tr>
<td>Identifier (ID, in short)</td>
<td>A token, such as an integer or a text string, assigned to an object with which an HRI system deals. Any ID cannot exist alone but it must be defined in some name space of a Reference Coordinate System (RCS), so ID and its corresponding RCS shall be treated as a unit. There exist two kinds of identifiers: permanent ID and temporary ID. Permanent ID is an identifier assigned to an object permanently, such as the social security number or an employee ID in a company. Temporary ID is used when sensors find objects which should be distinguished later but whose permanent IDs are not handy.</td>
</tr>
<tr>
<td>Localization</td>
<td>A function that finds target objects and returns a list of locations of objects found. A list of identifiers assigned to each object shall also be returned to distinguish objects each other.</td>
</tr>
<tr>
<td>Service Application</td>
<td>A software which controls HRI Components (via HRI Engine) to implement a robot scenario.</td>
</tr>
<tr>
<td>User-defined HRI Component</td>
<td>An HRI Component which provides an HRI function other than those any Basic HRI Components provide.</td>
</tr>
</tbody>
</table>
5 Symbols

No symbols are defined in this document.

6 Acknowledgements

Submitted by

Electronics and Telecommunications Research Institute
Japan Robot Association

Supported by

Advance Telecommunications Research Institute International
Future Robot Co., Ltd.
Hitachi, Ltd.
Korean Robot Association
National Institute of Advanced Industrial Science and Technology
New Energy and Industrial Technology Development Organization
Shibaura Institute of Technology
Technologic Arts Incorporated
University of Tokyo
University of Tsukuba
7  Platform Independent Model

7.1 Format and Conventions

7.1.1 Class and Interface

Classes and interfaces described in this PIM are documented using tables of the following format:

\[ \text{Table x.x : <Class / Interface Name>} \]

<table>
<thead>
<tr>
<th>Description : &lt;description&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived From: &lt;parent class&gt;</td>
</tr>
</tbody>
</table>

Attributes

<table>
<thead>
<tr>
<th>&lt;attribute name&gt;</th>
<th>&lt;attribute type&gt;</th>
<th>&lt;obligation&gt;</th>
<th>&lt;occurrence&gt;</th>
<th>&lt;description&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

Operations

<table>
<thead>
<tr>
<th>&lt;operation name&gt;</th>
<th>&lt;description&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;direction&gt;</td>
<td>&lt;parameter name&gt;</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

Note that derived attributes or operations are not described explicitly. Also, as the type of return code for every operation in this specification is Returncode_t, which is defined in Section 7.3, Return Codes, this is omitted in the description table.

The ‘obligation’ and ‘occurrence’ are defined as follows.

**Obligation**

- **M (mandatory):** This attribute shall always be supplied.
- **O (optional):** This attribute may be supplied.
- **C (conditional):** This attribute shall be supplied under a condition. The condition is given as a part of the attribute description.

**Occurrence**

The occurrence column indicates the maximum number of occurrences of the attribute values that are permissible. The followings denote special meanings.

- **N:** No upper limit in the number of occurrences.
- **ord:** The appearance of the attribute values shall be ordered.
- **unq:** The appeared attribute values shall be unique.

7.1.2 Enumeration

Enumerations are documented as follows:

\[ \text{Table x.x : <enumeration name>} \]
7.1.3 Message

Messages that are exchanged via the interfaces described in this PIM are documented using tables of the following format:

<table>
<thead>
<tr>
<th>Table x.x : &lt;Message Name&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description : &lt;description&gt;</td>
</tr>
<tr>
<td>Derived From: &lt;parent class&gt;</td>
</tr>
<tr>
<td>Attributes</td>
</tr>
<tr>
<td>&lt;attribute name&gt;</td>
</tr>
<tr>
<td>…</td>
</tr>
</tbody>
</table>

7.1.4 HRI Component and method

Methods that are incorporated in an HRI Component in this PIM are documented using tables of the following format:

<table>
<thead>
<tr>
<th>Table x.x : &lt;HRI Component Name&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description : &lt;description&gt;</td>
</tr>
<tr>
<td>Command Method</td>
</tr>
<tr>
<td>&lt;method name&gt;</td>
</tr>
<tr>
<td>argument</td>
</tr>
<tr>
<td>Event Method</td>
</tr>
<tr>
<td>&lt;method name&gt;</td>
</tr>
<tr>
<td>result</td>
</tr>
<tr>
<td>Query Method</td>
</tr>
<tr>
<td>&lt;method name&gt;</td>
</tr>
<tr>
<td>result</td>
</tr>
</tbody>
</table>

Note that derived methods are related to commands, events, and query messages, which are defined in Section 7.4.

The ‘argument’ and ‘result’ indicate that the columns of the line describe element of ‘ArgumentList’ and ‘ResultList’ for each message type, which are defined in Table 7.17 and Table 7.16, respectively.

7.2 Structure of the RoIS Framework

The Robotic Interaction Service (RoIS) Framework abstracts the hardware in the service robot (sensors and actuators) and the Human-Robot Interaction (HRI) functions provided by the robot, and provides a uniform interface between the service robot
and application.

Calling each of the HRI functions provided by a robotic system such as a service robot or intelligent sensing system a “functional implementation,” a robotic system can be expressed as a set of one or more functional implementations. These functional implementations (e.g. face recognition, wheel control) are usually provided in a form that is dependent on robot hardware such as sensors and actuators.

Referring to Figure 4, this specification defines the RoIS Framework as one that manages the interface not in units of functional implementations incorporated in the robot but rather in abstract functional units applicable to a service application. Such an abstract functional unit is called an “HRI Component.” Here, HRI Components (e.g. person detection, person identification) are logical functional elements making up the description of a human-robot interaction scenario.

These HRI Components are realized through physical units such as sensors placed on the robot and/or in the environment. It is assumed that one physical unit can have more than one function, which means that there is not necessarily a one-to-one match between physical units and functional units. As a result, physical units must be defined separately from functional units. With this in mind, a physical unit equipped with HRI Components is called an “HRI Engine.”

An entire system can consist of multiple physical units, and for such a system, the interface is managed by defining individual physical units as sub HRI Engines and the total system as the (main) HRI Engine that includes these sub HRI Engines.

The HRI Component provides hardware-independent APIs. Only symbolic data is exchanged between HRI Components and Service Applications through the HRI Engine. The symbolic data is used in the Service Applications without special handling such as pattern recognition, signal processing and human judgment. For example, the symbolic data shall not include raw data such as image data and sound data collected by the sensors.

Using the RoIS Framework as a go-between, a Service Application selects and uses only necessary functions and leaves
hardware-related matters such as which sensor to use to the HRI Engine. In the case that more than one sub HRI Engine includes the same HRI Component, the HRI Engine can be entrusted with selecting the appropriate sub HRI Engine. The use of HRI Components need not be static. Switching between HRI Components belonging to different sub HRI Engines can also be considered depending on robot position, sensor status, and other conditions. In this case, the Service Application simply specifies necessary functions since the main HRI Engine will automatically perform HRI Component switching. For example, in the case of the robotic service that covers broad areas, such automatic switching relieves the Service Application programmers of the selection of the actual HRI Components.

In this way, selection and switching of appropriate sub HRI Engines and HRI Components are all performed on the HRI-Engine side, so that in the RoIS Framework, service-application requirements assume unified interaction with only one HRI Engine, that is, the main HRI Engine regardless of the number and hierarchical configuration of sub HRI Engines and HRI Components. In other words, there is no need for the Service Application to be aware of the existence of sub HRI Engines or of how the main HRI Engine and sub HRI Engines interact with each other.

![Figure 5](image)

**Figure 5:** Schematic diagram of RoIS Framework and its message flows. In the framework, Service Application communicates with HRI Engine by some messages through System, Event, Query and Command Interfaces.

The RoIS Framework provides the following four interfaces consisting of a System Interface that enables the Service Application to use the RoIS Framework and three interfaces that enable the Service Application to exchange information with the HRI Engine (Figure 5).

- **System Interface**: Manages the connection status between the Service Application and HRI Engine.
- **Command Interface**: Enables the Service Application to send commands to the HRI Engine.
- **Query Interface**: Enables the Service Application to query the HRI Engine on information it holds.
- **Event Interface**: Enables the Service Application to receive notifications on changes in HRI-Engine status.

Here, data exchanged between the Service Application and HRI Engine via any of these interfaces are called “messages.” The following sections describe these interfaces and messages in more detail.

These messages shall include only the symbolic data. By doing so, the Service Application can obtain information only as the symbolic data through these interfaces. Also, the Service Application can specify instruction using only the symbolic data. For example, the symbolic data can be directly used for conditional programming sentences such as IF-type statement and SWITCH-type statement and specifying the robot behavior for human-robot interaction.

To make use of an HRI Engine, the Service Application must learn beforehand the functions provided by the HRI Engine, that is, the configuration of the HRI Engine and HRI Components and details on the messages that can be used. In this specification, such information is defined in terms of profiles, whose structures are described in Section 7.5.
7.3 Return Codes

At the PIM level we have modeled errors as operation return codes typed `ReturnCode_t`. Each PSM may map these to either return codes or exceptions. The complete list of return codes is indicated below.

Table 7.1: ReturnCode_t enumeration

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>Successful return.</td>
</tr>
<tr>
<td>ERROR</td>
<td>Generic, unspecified error.</td>
</tr>
<tr>
<td>BAD_PARAMETER</td>
<td>Illegal parameter value.</td>
</tr>
<tr>
<td>UNSUPPORTED</td>
<td>Unsupported operation.</td>
</tr>
<tr>
<td>OUT_OF_RESOURCES</td>
<td>Service ran out of the resources needed to complete the operation.</td>
</tr>
<tr>
<td>TIMEOUT</td>
<td>The operation timed out.</td>
</tr>
</tbody>
</table>

7.4 RoIS Interface

7.4.1 Interaction

7.4.1.1 System Interface

The System Interface manages the connection status between the Service Application and HRI Engine.

7.4.1.1.1 System Connection / Disconnection

The sequence diagram of the interface for performing connection and disconnection between the Service Application and HRI Engine is shown in Figure 6.
To begin with, the Service Application connects with the HRI Engine by connect(). On completing the connection, the Service Application executes get_profile() as needed to obtain profiles related to the functions provided by the HRI Engine.

To terminate use of the HRI Engine, the Service Application disconnects from the HRI Engine by disconnect().

The Service Application can send or receive no messages of any kind via the RoIS Framework until the connection operation with the RoIS Framework is completed. Additionally, the Service Application should not send or receive any messages under any circumstances after requesting a disconnection from the RoIS Framework. These operations are therefore executed in a synchronous manner.

### 7.4.1.1.2 System Error Notification

The sequence diagram of the interface enabling the Service Application to receive error notifications from the HRI Engine is shown in Figure 7.
In the event that an error has occurred in the HRI Engine or an HRI Component, the Service Application receives an error notification by notify_error() in an asynchronous manner. The notify_error() operation passes an “error_id” assigned to each error and “error_type” indicating the type of error. To obtain more detailed error information, the Service Application can execute get_error_detail() specifying that error_id.

The error notification of the HRI Engine is effective from the time connect() is called until disconnect() is called.

The error notification of the HRI Component is effective from the time bind() (or bind_any()) is called until release() is called via the Command Interface. Similarly, in the case of Event Interface, Service Applications can receive the error notification of the HRI Component from subscribe() until unsubscribe().

7.4.1.2 Command Interface

The Command Interface enables the Service Application to issue commands to an HRI Component. The sequence diagram of the Command Interface is shown in Figure 8.
It is assumed that an HRI Component can be used by more than one Service Application. Therefore, the Service Application needs to make a resource reservation for the necessary HRI Component so that it can avoid being operated by another Service Application. For this reason, firstly the Service Application binds the necessary HRI Component. Then, the Service Application requests the HRI Component to execute the operation. Finally, the Service Application releases the HRI Component when the operation is finished. The Command Interface includes these three steps, i.e., “BindComponent”, “Execute” and “Release”. The details of these steps are described as follows.
The Service Application specifies necessary conditions so that an HRI Component that can be used by the HRI Engine can be selected and subjected to a bind operation. Specifically, in the case that the Service Application selects an HRI Component from a list of candidates provided by the HRI Engine, the Service Application specifies conditions by search(), obtains a list of HRI-Component reference IDs (called “component_ref”s), and binds an HRI Component by specifying a component_ref from this list by bind(). Alternatively, in the case that an HRI Component is automatically selected by the HRI Engine, the Service Application specifies conditions by bind_any() and obtains a component_ref that has been bound.

Each operation within the Command Interface executes the selected HRI Component as a target of control by specifying the bound component_ref. This configuration enables the management of HRI-Component operation conditions to be consolidated in the HRI Engine. The Service Application therefore has no need to understand the operation conditions of HRI
Components, and interference from other Service Applications during a series of Command Interface processes can be prevented.

The Service Application may obtain and set HRI-Component parameters by get_parameter() and set_parameter(), respectively.

![Sequence Diagram](Figure 10: Sequence Diagram of “Execute” in Command Interface)

The Service Application issues a command against an HRI Component by using execute() to send a command message that specifies that command. The command message is described as a “command_unit_list” that can specify component both sequential command operation and parallel command operation. The details of “command_unit_list” are described in Section 7.4.3.1.

On receiving the command message from the Service Application, the HRI Engine immediately returns a return value and an ID for that command message (called a “command_id”) and begins performing the specified operation. This operation is executed in an asynchronous manner so that execution time does not affect the operation of the Service Application.
On completion of the specified operation, the Service Application asynchronously receives an operation-completed notification by completed(), which indicates the corresponding command_id and the completion state of that operation in the form of “status.”

The Service Application can obtain detailed execution results as needed by specifying the target command_id by get_command_result().

Once a series of Command Interface processes has been completed, the Service Application specifies the component_ref and releases that HRI Component by release().

In the above way, the Service Application can follow the execution status of each command message that it issues.

The Event Message described below is defined separately to provide notifications on the intermediate state of specific operations.

### 7.4.1.3 Query Interface

The Query Interface enables the Service Application to query the HRI Engine on information it holds. The sequence diagram of the Query Interface is shown in Figure 12.
The Service Application specifies a query message indicating the information to be obtained (called a “query_type”) and conditions for obtaining that information using query() and obtains desired information. This operation is executed in a synchronous manner since a state transition in a robot scenario is generally performed synchronously based on the information obtained by a query message. A query message can be issued at any time.

### 7.4.1.4 Event Interface

The Event Interface enables the Service Application to receive notifications on changes in the state of the HRI Engine. This interface performs “subscribe/unsubscribe” operations to register/cancel notifications and notification operations to pass events to the Service Application. The sequence diagram of the entire Event Interface is shown in Figure 13.
7.4.1.4.1 Event Registration / Cancellation

The Service Application uses subscribe() to register with the HRI Engine the type of the event message to be obtained (called an “event_type”). On receiving the event-message registration request from the Service Application, the HRI Engine immediately returns a return value and an ID for that registration (called a “subscribe_id”). On completing reception of event messages, the Service Application can cancel event-message notifications by using an unsubscribe() operation and specifying the subscribe_id assigned at the time of registration. The HRI Engine makes no notification of event messages that the Service Application is not subscribed to or of event messages that have been unsubscribed. In addition, the HRI Engine simply ignores subscribe requests for event messages that are already subscribed to and unsubscribe requests for event messages that have already been unsubscribed without issuing any errors.

7.4.1.4.2 Event Notification

The Service Application asynchronously receives an event message to which it has subscribed when the HRI Engine executes notify_event(). The notify_event() operation passes an ID assigned for every notification of an event message (called an “event_id”), event_type indicating the type of event message, and the subscribe_id assigned at the time of registering that notification. The Service Application can obtain detailed information on a notified event by performing a get_event_detail() operation with the event_id for that event specified.

Figure 13: Sequence Diagram of Event Interface.
7.4.2 Interfaces

The overall configuration of the interfaces in the RoIS Framework is shown in Figure 14.

Figure 14: RoIS Interfaces.

7.4.2.1 Interfaces for HRI Engine

The interfaces for the HRI Engine are defined in Table 7.2 to Table 7.5.

Table 7.2: System Interface

<table>
<thead>
<tr>
<th>Description:</th>
<th>The interface required to enable the HRI Engine to receive requests related to system management from the Service Application.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived From:</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operations</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>connect</td>
<td>Connects to the HRI Engine.</td>
</tr>
<tr>
<td>disconnect</td>
<td>Disconnects from the HRI Engine.</td>
</tr>
</tbody>
</table>
### Description
The interface required to enable the HRI Engine to receive command-related requests from the Service Application.

### Derived From
None

### Operations

#### search

Searches for an HRI Component matching the conditions for executing a function.

<table>
<thead>
<tr>
<th>in</th>
<th>condition</th>
<th>QueryExpression [ISO19143]</th>
<th>Specifies the conditions for the HRI Component to be searched for.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>component_ref_list</td>
<td>List&lt;RoIS_Identifier&gt;</td>
<td>Holds a list of IDs for components that match specified conditions.</td>
</tr>
</tbody>
</table>

#### bind

Binds the specified HRI Component.

<table>
<thead>
<tr>
<th>in</th>
<th>component_ref</th>
<th>RoIS_Identifier</th>
<th>Specifies the ID of the HRI Component to be bound.</th>
</tr>
</thead>
</table>

#### bind_any

Has the HRI Engine automatically select and bind an HRI Component that matches the conditions for executing a function.

<table>
<thead>
<tr>
<th>in</th>
<th>condition</th>
<th>QueryExpression [ISO19143]</th>
<th>Specifies the conditions of the HRI Component to be selected.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>component_ref</td>
<td>RoIS_Identifier</td>
<td>Holds the ID of the bound HRI Component.</td>
</tr>
</tbody>
</table>

#### release

Releases the specified HRI Component.

<table>
<thead>
<tr>
<th>in</th>
<th>component_ref</th>
<th>RoIS_Identifier</th>
<th>Specifies the ID of the HRI Component to be released.</th>
</tr>
</thead>
</table>

#### get_parameter

Obtains parameters of the bound HRI Component.

<table>
<thead>
<tr>
<th>in</th>
<th>component_ref</th>
<th>RoIS_Identifier</th>
<th>Specifies the ID of the bound HRI Component.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>parameters</td>
<td>ParameterList</td>
<td>Holds the obtained parameters.</td>
</tr>
</tbody>
</table>

#### set_parameter

Sets parameters of the bound HRI Component.

<table>
<thead>
<tr>
<th>in</th>
<th>component_ref</th>
<th>RoIS_Identifier</th>
<th>Specifies the ID of the bound HRI Component.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>parameters</td>
<td>ParameterList</td>
<td>Specifies the parameters to be set.</td>
</tr>
<tr>
<td>out</td>
<td>command_id</td>
<td>String</td>
<td>Holds the command ID assigned for this command message.</td>
</tr>
</tbody>
</table>

#### execute

Sends a command message to the bound HRI Component.

| in   | command_unit_list | CommandUnitSequence | Specifies the command messages to be sent and hold the command IDs for the messages. |

#### get_command_result

Obtains detailed results on completing execution of the command.

---

### Table 7.3: Command Interface

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_profile</td>
<td>Obtains the profile.</td>
</tr>
<tr>
<td>in</td>
<td>condition</td>
</tr>
<tr>
<td></td>
<td>QueryExpression [ISO19143]</td>
</tr>
<tr>
<td></td>
<td>Specifies the conditions of the profile to be obtained.</td>
</tr>
<tr>
<td>out</td>
<td>profile</td>
</tr>
<tr>
<td></td>
<td>HRI_Engine_Profile</td>
</tr>
<tr>
<td></td>
<td>Holds the obtained HRI Engine profile.</td>
</tr>
<tr>
<td>get_error_detail</td>
<td>Obtains details on an error notification from the HRI Engine.</td>
</tr>
<tr>
<td>in</td>
<td>error_id</td>
</tr>
<tr>
<td></td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Specifies the ID identifying the error event assigned at the time of error-event notification.</td>
</tr>
<tr>
<td>in</td>
<td>condition</td>
</tr>
<tr>
<td></td>
<td>QueryExpression [ISO19143]</td>
</tr>
<tr>
<td></td>
<td>Specifies the conditions for the error information to be obtained.</td>
</tr>
<tr>
<td>out</td>
<td>results</td>
</tr>
<tr>
<td></td>
<td>ResultList</td>
</tr>
<tr>
<td></td>
<td>Holds error information.</td>
</tr>
</tbody>
</table>

---

**Table 7.3: Command Interface**

<table>
<thead>
<tr>
<th>Description:</th>
<th>The interface required to enable the HRI Engine to receive command-related requests from the Service Application.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived From</td>
<td>None</td>
</tr>
<tr>
<td>Operations</td>
<td></td>
</tr>
<tr>
<td>search</td>
<td>Searches for an HRI Component matching the conditions for executing a function.</td>
</tr>
<tr>
<td>bind</td>
<td>Binds the specified HRI Component.</td>
</tr>
<tr>
<td>bind_any</td>
<td>Has the HRI Engine automatically select and bind an HRI Component that matches the conditions for executing a function.</td>
</tr>
<tr>
<td>release</td>
<td>Releases the specified HRI Component.</td>
</tr>
<tr>
<td>get_parameter</td>
<td>Obtains parameters of the bound HRI Component.</td>
</tr>
<tr>
<td>set_parameter</td>
<td>Sets parameters of the bound HRI Component.</td>
</tr>
<tr>
<td>execute</td>
<td>Sends a command message to the bound HRI Component.</td>
</tr>
<tr>
<td>get_command_result</td>
<td>Obtains detailed results on completing execution of the command.</td>
</tr>
<tr>
<td>in</td>
<td>command_id</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
</tr>
<tr>
<td>in</td>
<td>condition</td>
</tr>
<tr>
<td>out</td>
<td>results</td>
</tr>
</tbody>
</table>

Table 7.4: Query Interface

Description: The interface required to enable the HRI Engine to receive queries from the Service Application.

Derived From: None

Operations

<table>
<thead>
<tr>
<th>query</th>
<th>Sends a query message to the HRI Engine and obtains information.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>query_type</td>
</tr>
<tr>
<td>in</td>
<td>condition</td>
</tr>
<tr>
<td>out</td>
<td>results</td>
</tr>
</tbody>
</table>

Table 7.5: Event Interface

Description: The interface required to enable the HRI Engine to receive event-related requests from the Service Application.

Derived From: None

Operations

<table>
<thead>
<tr>
<th>subscribe</th>
<th>Registers an event message for which notifications are to be received.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>event_type</td>
</tr>
<tr>
<td>in</td>
<td>condition</td>
</tr>
<tr>
<td>out</td>
<td>subscribe_id</td>
</tr>
<tr>
<td>unsubscribe</td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>subscribe_id</td>
</tr>
<tr>
<td>get_event_detail</td>
<td>Obtain detailed information on this event notification.</td>
</tr>
<tr>
<td>in</td>
<td>event_id</td>
</tr>
<tr>
<td>in</td>
<td>condition</td>
</tr>
<tr>
<td>out</td>
<td>results</td>
</tr>
</tbody>
</table>
7.4.2.2 Interfaces for Service Application

The interface provided on the service-application side is defined in Table 7.6.

<table>
<thead>
<tr>
<th>Table 7.6: Service Application Base Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description:</strong> The interface required to enable the Service Application to receive notifications from the HRI Engine.</td>
</tr>
<tr>
<td><strong>Derived From:</strong> None</td>
</tr>
<tr>
<td><strong>Operations</strong></td>
</tr>
<tr>
<td><strong>notify_event</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
</tr>
<tr>
<td><strong>notify_error</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
</tr>
<tr>
<td><strong>completed</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
</tr>
<tr>
<td><strong>in</strong></td>
</tr>
</tbody>
</table>

*ErrorType* and *Completed_Status* are defined in Table 7.7 and Table 7.8.

<table>
<thead>
<tr>
<th>Table 7.7: ExceptionType enumeration</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENGINE_INTERNAL_ERROR</td>
</tr>
<tr>
<td>COMPONENT_INTERNAL_ERROR</td>
</tr>
<tr>
<td>COMPONENT_NOT_RESPONDING</td>
</tr>
<tr>
<td>USER_DEFINED_ERROR</td>
</tr>
</tbody>
</table>

Note: Corresponding situations of these error types shall be defined with respect to each HRI Engine.

<table>
<thead>
<tr>
<th>Table 7.8: Completed_Status enumeration</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
</tr>
<tr>
<td>ERROR</td>
</tr>
<tr>
<td>ABORT</td>
</tr>
<tr>
<td>OUT_OF_RESOURCES</td>
</tr>
<tr>
<td>TIMEOUT</td>
</tr>
</tbody>
</table>
Note: Corresponding situations of these statuses shall be defined with respect to each command message.

### 7.4.3 Message Data

The data exchanged by the RoIS Interface are summarized in the previous section as the parameters for each operation. Among these data, “message data” for each interface indicates the data that includes the information for the whole purpose of the interface. Thus, “command message” indicates the data exchanged by execute(), “query message” indicates the data exchanged by query(), and “event message” indicates the data exchanged by notify_event(). For the Command Interface and the Event Interface, the result of the command operation and the detail of the event notification are also important. Therefore, these data are defined as “command result message” and “event detail message” respectively. This section describes the data structure of each message.

#### 7.4.3.1 Command Message

The data structure of the command message exchanged by execute() is shown in Figure 15.

![Figure 15: Data Structure of Command Message.](image)

RoIS_Identifier is defined for describing an ID with the reference codebook for the ID. The detail of this data type is depicted in Table 7.9.

**Table 7.9: RoIS_Identifier**

<table>
<thead>
<tr>
<th>Description</th>
<th>A data type for describing an ID that identifying an instance and the reference codebook for the ID.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived From</td>
<td>MD_Identifier [ISO19115]</td>
</tr>
<tr>
<td>Attributes</td>
<td></td>
</tr>
<tr>
<td>codebook reference</td>
<td>String</td>
</tr>
<tr>
<td>version</td>
<td>String</td>
</tr>
</tbody>
</table>
The data configurations are defined in Table 7.10 to Table 7.18.

**Table 7.10: CommandUnitSequence class**

| Description: A data class for specifying a list of commands to the HRI Engine. |
| Derived From: |
| Attributes |
| command unit list | CommandUnit | M | Nord | CommandUnit object consisting of at least one command message. |

**Table 7.11: CommandUnit class**

| Description: An abstract data class for specifying a command or a concurrent combination of commands to the HRI Engine. |
| Derived From: |
| Attributes |
| delay time | Integer | O | 1 | A delay time from receiving the command message till starting the operation. The time shall be specified in millisecond. |

**Table 7.12: CommandMessage class**

| Description: A concrete data class for specifying a command to the HRI Engine. |
| Derived From: CommandUnit |
| Attributes |
| component_ref | RoIS_Identifier | M | 1 | Identifier of the HRI Component. |
| command_type | String | M | 1 | Identifier of the command message type. The operation “execute” in the command interface shall operate similarly to the operation “set_parameter” in the command interface when the command_type is “set_parameter”. |
| command_id | Sting | M | 1 | ID of the command transmission assigned when the HRI Engine receiving the command message. |
| arguments | ArgumentList | O | 1 | Arguments for the command message |

**Table 7.13: ConcurrentCommands class**

| Description: A concrete data class for specifying a combination of commands to the HRI Engine that expresses a procedure for operating several command messages in parallel. |
| Derived From: CommandUnit |
| Attributes |
Each Branch object contains at least one CommandMessage. HRI Engine processes Branch objects in parallel.

Table 7.14: Branch class

Description: A concrete data class for specifying a combination of commands to the HRI Engine that expresses a procedure for operating several command messages sequentially.

Derived From:

Attributes

| command list | CommandMessage | M | N | CommandMessage object consisting of at least one command message. |

ResultList, ArgumentList and ParameterList are defined for treating data values in each message as depicted in the following tables.

Table 7.15: Parameter class

Description: A data class for specifying a parameter.

Derived From: None

Attributes

| name | String | M | N | Parameter name |
| data_type_ref | RoIS_Identifier | M | N | Reference ID of data definition |
| value | Any | M | N | Parameter value |

Table 7.16: ResultList class

Description: A data class for specifying a list of result parameters

Derived From: None

Attributes

| Parameters | Parameter | M | Nord | Result parameters |

Table 7.17: ArgumentList class

Description: A data class for specifying a list of argument parameters

Derived From: None

Attributes

| Parameters | Parameter | M | Nord | Argument parameters |
### 7.4.3.2 Command Result Message

The data configuration of the command result message exchanged by `get_command_result()` is given below.

#### Table 7.19: Command Result Message class

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Description: A data class for specifying a command result message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived From: None</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Derived From</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M Nord Configuration parameters</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>command_id</td>
<td>ID of the command transmission assigned when receiving the command message</td>
<td>M</td>
<td>1</td>
</tr>
<tr>
<td>condition</td>
<td>Conditions of information to be obtained</td>
<td>O</td>
<td>1</td>
</tr>
<tr>
<td>results</td>
<td>Results of command execution</td>
<td>M</td>
<td>1</td>
</tr>
</tbody>
</table>

### 7.4.3.3 Query Message

The data configuration of the query message exchanged by `query()` is given below.

#### Table 7.20: Query Message class

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Description: A data class for specifying a query message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived From: None</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Derived From</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>type of the query message</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>query_type</td>
<td></td>
<td>M</td>
<td>1</td>
</tr>
<tr>
<td>condition</td>
<td>Conditions of information to be obtained</td>
<td>O</td>
<td>1</td>
</tr>
<tr>
<td>results</td>
<td>Obtained information</td>
<td>M</td>
<td>1</td>
</tr>
</tbody>
</table>

### 7.4.3.4 Event Message

The data configuration of the event message exchanged by `notify_event()` is given below.

#### Table 7.21: Event Message class

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Description: A data class for specifying an event message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived From: None</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Derived From</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>
### 7.4.3.5 Event Detail Message

The data configuration of event details exchanged by get_event_detail() is given below.

**Table 7.22: Event Detail Message class**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>M</th>
<th>O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_id</td>
<td>String</td>
<td>M</td>
<td>1</td>
<td>ID of the event notification assigned when sending the event message</td>
</tr>
<tr>
<td>condition</td>
<td>QueryExpression [ISO19143]</td>
<td>O</td>
<td>1</td>
<td>Conditions of information to be obtained</td>
</tr>
<tr>
<td>results</td>
<td>ResultList</td>
<td>M</td>
<td>1</td>
<td>Detailed information on event</td>
</tr>
</tbody>
</table>

### 7.4.3.6 Error Message

The data configuration of event details exchanged by notify_error() is given below.

**Table 7.23: Error Message class**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>M</th>
<th>O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_id</td>
<td>String</td>
<td>M</td>
<td>1</td>
<td>ID of the error notification assigned when sending the event message</td>
</tr>
<tr>
<td>error_type</td>
<td>String</td>
<td>M</td>
<td>1</td>
<td>Type of the error message</td>
</tr>
<tr>
<td>subscribe_id</td>
<td>String</td>
<td>M</td>
<td>1</td>
<td>ID of event registration assigned when registering the event message</td>
</tr>
<tr>
<td>expire</td>
<td>DateTime[W3C-DT]</td>
<td>O</td>
<td>1</td>
<td>Time limit for obtaining detailed results by get_error_detail().</td>
</tr>
</tbody>
</table>

### 7.4.3.7 Error Detail Message

The data configuration of error details exchanged by get_error_detail() is given below.

**Table 7.24: Error Detail Message class**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>M</th>
<th>O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_id</td>
<td>String</td>
<td>M</td>
<td>1</td>
<td>ID of the error notification assigned when sending the event message</td>
</tr>
<tr>
<td>error_type</td>
<td>String</td>
<td>M</td>
<td>1</td>
<td>Type of the error message</td>
</tr>
<tr>
<td>subscribe_id</td>
<td>String</td>
<td>M</td>
<td>1</td>
<td>ID of event registration assigned when registering the event message</td>
</tr>
<tr>
<td>expire</td>
<td>DateTime[W3C-DT]</td>
<td>O</td>
<td>1</td>
<td>Time limit for obtaining detailed results by get_error_detail().</td>
</tr>
</tbody>
</table>
7.5 Profiles

7.5.1 Overview

Profiles define the functions provided by the HRI Engine via the RoIS Framework interfaces, that is, the configuration of the HRI Engine and HRI Components, and the messages that can be used. They are used to obtain information so that the Service Application can make use of HRI-Engine functions.

Figure 16: RoIS Profile. RoIS profile mainly consists of 4 types of profiles, i.e. “HRI_Engine_Profile”, “HRI_Component_Profile”, “Message_Profile”, and “Parameter_Profile”.

An HRI Engine Profile, HRI Component Profile, and Message Profile are defined for the HRI-Engine layer of physical units, the HRI-Component layer of abstract functional units, and the message layer of data exchanged between the Service Application and HRI Components, respectively, in the RoIS Framework. These profiles enable the Service Application to understand the configuration of the HRI Engine.

The main application of each profile is summarized below.

Parameter Profile: This profile defines the parameters of message arguments, results, the HRI Engine, and HRI Components. It defines parameter identifier (parameter name), data type, and default value.

Message Profile: This profile defines messages to be sent and received between the Service Application and HRI Engine via the RoIS Framework. It defines message identifiers (message name) and required arguments and results. Arguments and results are defined by including a Parameter Profile defined for each parameter. The profile for each type of message corresponding to an interface (command message, query message, and event message) is defined as a subclass of this class.

HRI Component Profile: This profile defines a list of messages and parameters possessed by an HRI-Component unit. It defines HRI-Component identifiers (HRI-Component name, ID, etc.). Messages and parameters that can be used by this
HRI Component are defined by specifying Message Profiles and Parameter Profiles. An HRI Component that includes multiple sub-HRI-Components can be defined by specifying other HRI-Component Profiles as sub-profiles.

HRI Engine Profile: This profile defines a list of HRI Components and parameters possessed by an HRI-Engine unit. It defines HRI-Engine identifiers (HRI-Engine name, ID, etc.). HRI Components and parameters that can be used by this HRI Engine are defined by specifying HRI-Component Profiles and Parameter Profiles. An HRI Engine that includes multiple sub-HRI-Engines can be defined by specifying other HRI Engine Profiles as sub-profiles.

The Service Application obtains an HRI Engine profile (or its referent) by get_profile(). It can obtain the HRI Engine Profile of a certain HRI Engine by specifying conditions such as the location of that HRI Engine or the HRI Components possessed by the HRI Engine in ‘condition’.

The Service Application can then learn about the types of available functions through the identifiers of HRI-Component Profiles included in the HRI Engine Profile. Additionally, it can obtain detailed information on messages exchanged by each interface when using a certain HRI Component through Message Profiles included in that HRI-Component Profile.

Specifically, the Service Application begins by searching for desired functions from the identifiers of HRI-Component Profiles included in the obtained HRI Engine Profile. If a command message is to be used, the Service Application searches for an HRI-Component Profile having the same identifier as that obtained at the time of binding.

When exchanging a message, the Service Application specifies the identifier of that message. Detailed information on a message to be exchanged can be obtained by referencing the profile having the same identifier as that message from the Message Profiles corresponding to the interface to be used.

Definitions of identifiers and data types of arguments needed when exchanging a message can be obtained from Parameter Profiles included in that Message Profile.

When exchanging a message, passing a list of values as arguments (or results) based on parameter identifiers and data types defined in these Parameter Profiles guarantees that the data types exchanged between the Service Application and HRI Engine match up.

The same holds for parameters. Passing a list of values as set_parameter() and get_parameter() arguments based on parameter identifiers and data types defined in Parameter Profiles included in an HRI-Engine Profile or HRI-Component Profile guarantees that the data types exchanged between the Service Application and HRI Engine match up. Information on standard values can also be obtained from default values defined in Parameter Profiles.

Details of each profile are described in the following sections.

### 7.5.2 Parameter Profile

The Parameter Profile defines parameters for message arguments and HRI-Engine and HRI-Component parameters. Items to be defined in this profile are listed in Table 7.25.

<table>
<thead>
<tr>
<th>Table 7.25: Parameter_Profile</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Description: Profile for defining each parameter for HRI Engines.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Derived From :</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Attributes</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>String</th>
<th>M</th>
<th>1</th>
<th>Parameter name</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_type_ref</td>
<td>RoIS_Identifier</td>
<td>M</td>
<td>1</td>
<td>Reference ID of data definition</td>
</tr>
<tr>
<td>default_value</td>
<td>Any</td>
<td>O</td>
<td>1</td>
<td>Necessary arguments when issuing this message</td>
</tr>
<tr>
<td>description</td>
<td>String</td>
<td>O</td>
<td>1</td>
<td>Description</td>
</tr>
</tbody>
</table>
7.5.3 Message Profile

The Message Profile defines messages exchanged between the Service Application and HRI Engine via the interfaces in the RoIS Framework. This profile is defined for every message. Items to be defined in this profile are listed in Table 7.26.

**Table 7.26: Message Profile**

| Description: Base profile for defining messages for each interface type. |
| Derived From : None |
| Attributes |

<table>
<thead>
<tr>
<th>name</th>
<th>String</th>
<th>M</th>
<th>I</th>
<th>Message name</th>
</tr>
</thead>
<tbody>
<tr>
<td>results</td>
<td>Parameter_Profile</td>
<td>O</td>
<td>N ord</td>
<td>Defines parameters obtained as execution results in this message (parameters included in get_command_result() in command interface, query() in query interface, and get_event_detail() in event interface). The definition method follows that of the Parameter Profile. Multiple items may be defined.</td>
</tr>
</tbody>
</table>

Messages used in the Command Interface are defined in the Command Message Profile. Items to be defined in the Command Message Profile are listed in Table 7.27.

**Table 7.27: Command Message Profile**

| Description: Profile for defining messages for command interface type. |
| Derived From : Message_Profile |
| Attributes |

| argument | Parameter_Profile | O | N ord | Defines parameters given as arguments in this message (parameters included in arguments of execute() in the command interface). The definition method follows that of the Parameter Profile. Multiple items may be defined. |
| timeout | Integer | O | I | The time between receipt of the message and judgment of failure to start the operation. The time shall be specified in millisecond. |

Messages used in the Command Interface to send the results are defined in the Command Result Message Profile. Items to be defined in the Command Result Message Profile are listed in Table 7.28.
Table 7.28: Command_Result_Message_Profile

<table>
<thead>
<tr>
<th>Description: Profile for defining messages for command interface type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived From: Message_Profile</td>
</tr>
</tbody>
</table>

Messages used in the Query Interface are defined in the Query Message Profile. Items to be defined in the Query Message Profile are listed in Table 7.29.

Table 7.29: Query_Message_Profile

<table>
<thead>
<tr>
<th>Description: Profile for defining messages for query interface type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived From: Message_Profile</td>
</tr>
</tbody>
</table>

Messages used in the Event Interface are defined in the Event Detail Message Profile. Items to be defined in the Event Detail Message Profile are listed in Table 7.30.

Table 7.30: Event_Detail_Message_Profile

<table>
<thead>
<tr>
<th>Description: Profile for defining messages for command interface type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived From: Message_Profile</td>
</tr>
</tbody>
</table>

Messages used in the System Interface are defined in the Error Detail Message Profile. Items to be defined in the Error Detail Message Profile are listed in Table 7.31.

Table 7.31: Error_Detail_Message_Profile

<table>
<thead>
<tr>
<th>Description: Profile for defining messages for system interface type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived From: Message_Profile</td>
</tr>
</tbody>
</table>

7.5.4 HRI Component Profile

The HRI Component Profile defines the abstract functional units to be used by the Service Application corresponding to the functions provided by the HRI Engine. That is, it defines the class of HRI Component and the messages that can be used by that HRI Component. This profile is defined for every HRI Component. Items to be defined in this profile are listed in Table 7.32.

Table 7.32: HRI Component Profile

<table>
<thead>
<tr>
<th>Description: Profile for defining lists of messages and parameters for each HRI Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived From: IO_IdentifiedObject [ISO19111]</td>
</tr>
<tr>
<td>Attributes</td>
</tr>
</tbody>
</table>
message | Message_Profile | M | N | Defines a message profile for a message of the HRI Component. The definition method follows that of the Message Profile. Multiple items may be defined.
---|---|---|---|---
sub_component | HRI_Component_Profile | O | 1 | Specifies an HRI Component profile when included in the definition of another HRI Component profile. Only one item may be defined.
parameter | Parameter_Profile | O | N ord | Defines the parameter profile for a parameter of this HRI Component. The definition method follows that of the Parameter Profile. Multiple items may be defined.

### 7.5.5 HRI Engine Profile

The HRI Engine Profile defines the class of an HRI Engine or sub HRI Engine and the HRI Components that can be used by that HRI Engine. This profile is defined for every HRI Engine. Items to be defined in this profile are listed in Table 7.33.

**Table 7.33: HRI_Engine_Profile**

<table>
<thead>
<tr>
<th>Description</th>
<th>Profile for defining lists of logical units and parameters for each HRI Engine and sub HRI Engine.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived From</td>
<td>IO_IdentifiedObject [ISO19111]</td>
</tr>
<tr>
<td>Attributes</td>
<td></td>
</tr>
<tr>
<td>component</td>
<td></td>
</tr>
<tr>
<td>HRI_Component_Profile</td>
<td>M</td>
</tr>
<tr>
<td>sub_profile</td>
<td></td>
</tr>
<tr>
<td>HRI_Engine_Profile</td>
<td>O</td>
</tr>
</tbody>
</table>

### 7.6 Common Messages

In this specification, messages received via an interface of the HRI Engine are called HRI-Component methods and common messages are defined as the methods.

In the RoIS Framework, the HRI Components shown in Table 7.34 are defined as Basic HRI Components. The Basic HRI Components are HRI Components that are commonly used to obtain information and to control robot behaviors for the human-robot interaction. The Basic HRI Component shall be a functional unit that is developed with mature technologies from the viewpoint of the usage. Methods for each Basic HRI Component shall be simple as possible. Mandatory parameters for the operation shall be minimized. The Basic HRI Component shall be operated only with the mandatory parameter. If the component can provide additional information or configuration parameter, those parameters may be provided as optional
parameter. The other HRI Components may be provided as “User-defined HRI Component”. Examples of “User-defined HRI Component” are described in Annex C.

Note that it is not mandatory for an HRI Engine to implement all of these Basic HRI Components. It is sufficient that they only have the HRI Component Profiles of the actually-implemented HRI Components.

Table 7.34: Basic HRI Components

<table>
<thead>
<tr>
<th>HRI Component Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system information</td>
<td>Provides the information of the system such as status of the system and position of the physical unit.</td>
</tr>
<tr>
<td>person detection</td>
<td>Detects number of people</td>
</tr>
<tr>
<td>person localization</td>
<td>Detect position of people</td>
</tr>
<tr>
<td>person identification</td>
<td>Identifies ID (name) of people</td>
</tr>
<tr>
<td>face detection</td>
<td>Detects number of human faces</td>
</tr>
<tr>
<td>face localization</td>
<td>Detects position of human faces</td>
</tr>
<tr>
<td>sound detection</td>
<td>Detects number of sound sources</td>
</tr>
<tr>
<td>sound localization</td>
<td>Detects position of sound sources</td>
</tr>
<tr>
<td>speech recognition</td>
<td>Recognizes person’s speech</td>
</tr>
<tr>
<td>gesture recognition</td>
<td>Recognizes person’s gesture</td>
</tr>
<tr>
<td>speech synthesis</td>
<td>Generates robot speech.</td>
</tr>
<tr>
<td>reaction</td>
<td>Performs specified reaction.</td>
</tr>
<tr>
<td>navigation</td>
<td>Moves to specified target location</td>
</tr>
<tr>
<td>follow</td>
<td>Follows a specified target object</td>
</tr>
<tr>
<td>move</td>
<td>Moves to specified distance or curve</td>
</tr>
</tbody>
</table>

Each HRI Component incorporates the following methods and parameters in common.

Table 7.35: RoIS_Common

<table>
<thead>
<tr>
<th>Description: common method for all HRI Components.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Command Method</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>Start the functionality of the HRI Component.</td>
</tr>
<tr>
<td>stop</td>
<td>Stop the functionality of the HRI Component.</td>
</tr>
<tr>
<td>suspend</td>
<td>Pause the functionality of the HRI Component.</td>
</tr>
<tr>
<td>resume</td>
<td>Resume the functionality of the HRI Component.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query Method</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>component_status</td>
<td>Obtain status information of the component.</td>
</tr>
<tr>
<td>result status</td>
<td>Component_Status</td>
</tr>
</tbody>
</table>
Component status is defined as follows.

<table>
<thead>
<tr>
<th>Component Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNINITIALIZED</td>
<td>The component is not initialized.</td>
</tr>
<tr>
<td>READY</td>
<td>The component is ready to use.</td>
</tr>
<tr>
<td>BUSY</td>
<td>The component is used by other application(s).</td>
</tr>
<tr>
<td>WARNING</td>
<td>Warning against the use of the component</td>
</tr>
<tr>
<td>ERROR</td>
<td>Generic, unspecified error.</td>
</tr>
</tbody>
</table>

Methods and parameters of each HRI Component described in this PIM are documented in the following sections.

### 7.6.1 System Information

![Diagram of System Information]

Figure 17: System Information

Table 7.37: System information
Description: This is a component for providing system information. The system information includes the status and the location of the system. This information belongs to the HRI Engine that is treated as a unified physical unit of several HRI Components. Therefore this component is different from other HRI Components and does not include RoIS_Common methods.

Localization of a physical unit (i.e., robot, sensor, and actuator) is one of the essential functions for providing robotic services in physical space. An HRI Engine that is defined as a physical unit shall include this HRI Component to inform Service Applications about its location information. The location information depends on the physical elements of the HRI Engine; for example, if the HRI Engine is defined as a movable robot, this component may provide at least the position of the robot, and if the HRI Engine consists of sensors that are mounted in a wide room extensively, this component may provide at least the reference position. When possible, the HRI Component may provide the location information of each sensor or actuator as a list of location data.

Query Method

<table>
<thead>
<tr>
<th>robot_position</th>
<th>Returns location information.</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>timestamp</td>
</tr>
<tr>
<td>result</td>
<td>robot_ref</td>
</tr>
<tr>
<td>result</td>
<td>position data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>engine_status</th>
<th>Returns status information of the HRI Engine.</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>status</td>
</tr>
<tr>
<td>result</td>
<td>operable time</td>
</tr>
</tbody>
</table>
7.6.2 Person Detection

Table 7.38: person detection

<table>
<thead>
<tr>
<th>Event Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>person_detected</td>
<td>Notifies number of people when the number has changed.</td>
</tr>
<tr>
<td>result</td>
<td>timestamp</td>
</tr>
<tr>
<td>result</td>
<td>number</td>
</tr>
</tbody>
</table>
7.6.3 Person Localization

Description: This is a component for detecting position of persons. This component notifies position of the detected people when the position has been localized.

This functionality is essential for typical robotic services; for example, when a robot finds a person close to it, the robot may approach to the person and start asking if there is something the robot can do for the person. In some advanced robotic services, an environmental sensing system may find out a person in lost and order robots to approach the person for help.

Command Method

<table>
<thead>
<tr>
<th>Argument</th>
<th>Detection threshold</th>
<th>Integer</th>
<th>O</th>
<th>This component notifies an event if the distance of movement since previous event notification exceeds the threshold value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
<td>Minimum interval</td>
<td>Integer</td>
<td>O</td>
<td>This component notifies an event if the period since previous event notification exceeds the value of minimal interval.</td>
</tr>
</tbody>
</table>

Query Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Obtains person localization parameters.</th>
</tr>
</thead>
</table>

Table 7.39: person localization
This component notifies an event if the distance of movement since previous event notification exceeds the threshold value.

This component notifies an event if the distance of movement since previous event notification exceeds the threshold value.

Event Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>person_localized</td>
<td>Notifies position of people when the position has localized.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>result</th>
<th>timestamp</th>
<th>DateTime [W3C-DT]</th>
<th>M</th>
<th>Measurement time.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>result</th>
<th>person_ref</th>
<th>List&lt;RoIS_Identifier&gt;</th>
<th>M</th>
<th>List of detected person IDs. Reference information related to the ID shall be provided with each ID. By referring to the reference for the IDs, the Service Application can understand the relationship between the obtained IDs and the other IDs that are obtained from another component.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>result</th>
<th>position data</th>
<th>List&lt;Data[RLS]&gt;</th>
<th>M</th>
<th>List of detected person data. Each data entry at least contains position of the detected person. This may also be accompanied with additional information such as pose of the detected person. It may also contain certainty of the detection act.</th>
</tr>
</thead>
</table>

### 7.6.4 Person Identification

```plaintext
<<interface>>
RoIS_Common
+ start(): Returncode_t
+ stop(): Returncode_t
+ suspend(): Returncode_t
+ resume(): Returncode_t
+ component_status(out status: Component_Status): Returncode_t
```

```plaintext
<<interface>>
Person_Identification
+ person_identified(in timestamp: DateTime, in person_ref: List<RoIS_Identifier>): void
```

```plaintext
<<enumeration>>
Component_Status
<<enum>> UNINITIALIZED
<<enum>> READY
<<enum>> BUSY
<<enum>> WARNING
<<enum>> ERROR
```

```plaintext
<<HRI_Component>>
Person_Identification_Component
```

```plaintext
<<HRI_Engine>>
Engine1
```
Figure 20: person identification

Table 7.40: person identification

<table>
<thead>
<tr>
<th>Event Method</th>
<th>person_identified</th>
<th>Notifies ID of people when the ID has identified.</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>timestamp</td>
<td>DateTime [W3C-DT]</td>
</tr>
<tr>
<td>result</td>
<td>person ref</td>
<td>List&lt;RoIS_Identifier&gt;</td>
</tr>
</tbody>
</table>
7.6.5 Face Detection

Description: This is a component for detecting number of human faces. This component notifies a number of the detected faces when the number has changed.

This functionality is similar to “person_detection” component but it is treated as a separate component. This is because often the detection of human face has an individual meaning in the Service Applications. For example, if a robot detect a person but the person is not facing to the robot, the robot may not talk to the person. In such a case, the robot may move to the other direction of the person or wait until the person turns to the robot. Therefore this functionality is also essential for various robotic services.

Event Method

<table>
<thead>
<tr>
<th>Event Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>face_detected</td>
<td>Notifies number of human face when the number has changed.</td>
</tr>
</tbody>
</table>

Table 7.41: face detection

<table>
<thead>
<tr>
<th>Event Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>timestamp</td>
</tr>
<tr>
<td>result</td>
<td>number</td>
</tr>
</tbody>
</table>
7.6.6 Face Localization

Description: This is a component for detecting position of human faces. This component notifies position of the detected human face(s) when the position has been localized.

This functionality is similar to “person_localization” component but it is treated as a separate component. This is because often the position of human face has an individual meaning in the Service Applications. For example, if a robot is smaller than human, the robot may need to look up the person. In such case, the position of the face is needed separately from the position of the person. Therefore this functionality is also essential for various robotic services.

<table>
<thead>
<tr>
<th>Command Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>set_parameter</strong></td>
<td>Specifies face localization parameters.</td>
</tr>
<tr>
<td>argument</td>
<td>detection threshold</td>
</tr>
<tr>
<td>argument</td>
<td>minimum interval</td>
</tr>
</tbody>
</table>

**Query Method**

| get_parameter | Obtains face localization parameters. |
| result | detection threshold | Integer | O | This component notifies an event if the |
distance of movement since previous event notification exceeds the threshold value.

**Result**

- minimum interval: Integer

This component notifies an event if the period since previous event notification exceeds the value of minimal interval.

**Event Method**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>face_localized</td>
<td>Notifies position of human face when the position has localized.</td>
</tr>
</tbody>
</table>

**Result**

<table>
<thead>
<tr>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td>DateTime [W3C-DT]</td>
</tr>
<tr>
<td>face ref</td>
<td>List&lt;RoIS_Ientifier&gt;</td>
</tr>
<tr>
<td>position data</td>
<td>List&lt;Data[RLS]&gt;</td>
</tr>
</tbody>
</table>

Measurement time.

List of detected human face IDs. Reference information related to the ID shall be provided with each ID. By referring to the reference for the IDs, the Service Application can understand the relationship between the obtained IDs and the other IDs that are obtained from another component.

List of detected human face data. Each data entry at least contains position of the detected face. This may also be accompanied with additional information such as pose of the detected face. It may also contain certainty of the detection act.

### 7.6.7 Sound Detection

**Class SoundDetection**

```mermaid
class Sound_Detection
    <<interface>>
    RoIS_Common
    + start(): Returncode_t
    + stop(): Returncode_t
    + suspend(): Returncode_t
    + resume(): Returncode_t
    + component_status(out status: Component_Status): Returncode_t

tagComponent
    Sound_Detection_Component
    0..*

<<interface>>
    Sound_Detection
    + sound_detected(timestamp: DateTime, number: Integer): void

<<HRI_Component>>
    Sound_Detection_Component

<<HRI_Engine>>
    Engine1

<<enumeration>>
    Component_Status
    <<enum>> UNINITIALIZED
    <<enum>> READY
    <<enum>> BUSY
    <<enum>> WARNING
    <<enum>> ERROR
```
Table 7.43: sound detection

Description: This is a component for detecting number of sound sources. This component notifies a number of detected sound sources when the number has changed.

This functionality is essential for typical robotic services; for example, in the case of home security service, the robot may watch for intruders coming or sound an alarm when it hears something.

Event Method

<table>
<thead>
<tr>
<th>sound_detected</th>
<th>Notifies number of sound sources when the number has changed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>timestamp</td>
</tr>
<tr>
<td>result</td>
<td>number</td>
</tr>
</tbody>
</table>

7.6.8 Sound Localization

Table 7.44: sound localization
Description: This is a component for detecting position of sound sources. This component notifies position of detected sound source(s) when the position has been localized.

Often this functionality is used to detect the location of the speaker(s) by detecting the speaker’s voice since a person talks to the robot when he/she wants to start interaction.

<table>
<thead>
<tr>
<th>Command Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_parameter</td>
</tr>
<tr>
<td>argument</td>
</tr>
<tr>
<td>This component notifies an event if the distance of movement since previous notification exceeds the threshold value.</td>
</tr>
<tr>
<td>argument</td>
</tr>
<tr>
<td>This component notifies an event if the period since previous event notification exceeds the threshold value.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_parameter</td>
</tr>
<tr>
<td>result</td>
</tr>
<tr>
<td>This component notifies an event if the distance of movement since previous notification exceeds the threshold value.</td>
</tr>
<tr>
<td>result</td>
</tr>
<tr>
<td>This component notifies an event if the period since previous event notification exceeds the value of minimal interval.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>sound_localized</td>
</tr>
<tr>
<td>result</td>
</tr>
<tr>
<td>Measurement time.</td>
</tr>
<tr>
<td>result</td>
</tr>
<tr>
<td>List of detected sound source IDs. Reference information related to the ID shall be provided with each ID. By referring to the reference for the IDs, the Service Application can understand the relationship between the obtained IDs and the other IDs that are obtained from another component.</td>
</tr>
<tr>
<td>result</td>
</tr>
<tr>
<td>List of detected sound source data. Each data entry at least contains position of the detected sound source. It may also contain certainty of the detection act.</td>
</tr>
</tbody>
</table>
7.6.9 Speech Recognition

Description: This is a component for recognizing human speech. This component notifies text data of the recognized speech when the speech has been recognized.

This functionality is essential for human robot interactions, from simply ordering the robot to do something to giving enough information to the Service Application for appropriate services.

Here, we assume speech recognition algorithm which is not configurable by a descriptive grammar (e.g. W3C-SRGS). See Annex C for speech recognition algorithm which can be configured by a descriptive grammar. Mandatory requirement for the speech recognition component is to return result in string format.

Command Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_parameter</td>
<td>Specifies speech recognition paramters.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>languages</td>
<td>Set&lt;String&gt; [ISO639-1]</td>
<td>Specifies languages the speech recognizer will recognize.</td>
</tr>
<tr>
<td>grammar</td>
<td>String</td>
<td>Specifies grammar for the speech recognizer.</td>
</tr>
<tr>
<td>rule</td>
<td>String</td>
<td>Specifies active rule in the grammar.</td>
</tr>
</tbody>
</table>

Query Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_parameter</td>
<td>Obtains speech recognition paramters.</td>
</tr>
<tr>
<td>result</td>
<td>recognizable languages</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------</td>
</tr>
<tr>
<td>result</td>
<td>languages</td>
</tr>
<tr>
<td>result</td>
<td>grammar</td>
</tr>
<tr>
<td>result</td>
<td>rule</td>
</tr>
</tbody>
</table>

Event Method

**speech_recognized**
Notifies recognized result when the speech has been recognized.

<table>
<thead>
<tr>
<th>result</th>
<th>timestamp</th>
<th>DateTime [W3C-DT]</th>
<th>M</th>
<th>Time when the recognition has completed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>recognized text</td>
<td>List&lt;String&gt;</td>
<td>M</td>
<td>List of speech recognition results. The result is provided as string data. For the speech recognition algorithm which can only output one candidate, returning a list filled with one result is recommended. String of recognized text can contain either a word or a sentence.</td>
</tr>
</tbody>
</table>

**speech_input_started**
Notifies the recognizer has detected start of speech input.

<table>
<thead>
<tr>
<th>result</th>
<th>timestamp</th>
<th>DateTime [W3C-DT]</th>
<th>M</th>
<th>Time when the speech input has started.</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>timestamp</td>
<td>DateTime [W3C-DT]</td>
<td>M</td>
<td>Time when the speech input has ended.</td>
</tr>
</tbody>
</table>

**7.6.10 Gesture Recognition**

```java
<Interface>
GestureRecognition
+
getRecognizableGesture : Set<RoiIdentifier> : Returncode_t
+ gestureRecognized(RecognizableGesture : Set<RoiIdentifier> : DateTime : List<RoiIdentifier>) : void

(org)GestureRecognition_Component

<Enum>
Component_Status
+ UNINITIALIZED
+ READY
+ BUSY
+ WARNING
+ ERROR
```

Figure 26: gesture recognition
Table 7.46: gesture recognition

Description: This is a component for recognizing human gesture. This component notifies ID of the recognized gesture when the gesture has been recognized.

This functionality is essential for human robot interactions. In the case of noisy environment or far field interaction, the user may interact with the robot by using gesture.

The meaning of gesture is different among such as countries and situations. Also the recognizable gestures may be different by gesture recognition algorithms. The result shall be simply provided as gesture ID and the Service Application shall understand the meaning of the ID by the reference for the ID.

Query Method

<table>
<thead>
<tr>
<th>get_parameter</th>
<th>Obtains speech recognition parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>recognizable gestures</td>
</tr>
<tr>
<td>Set&lt;RoIS_Identifier&gt;</td>
<td>M</td>
</tr>
<tr>
<td>result</td>
<td>Obtain gestures the gesture recognizer can recognize. The gesture is expressed as ID and the reference for the ID shall be provided with each ID.</td>
</tr>
<tr>
<td>Event Method</td>
<td>Notifies recognized result when the gesture has been recognized.</td>
</tr>
<tr>
<td>result</td>
<td>timestamp</td>
</tr>
<tr>
<td>DateTime [W3C-DT]</td>
<td>M</td>
</tr>
<tr>
<td>result</td>
<td>gesture ref</td>
</tr>
<tr>
<td>List&lt;RoIS_Identifier&gt;</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>List of gesture recognition results. The result is provided as gesture types. The type is specified by gesture IDs. Reference information related to the ID shall be provided with each ID. For the gesture recognition algorithm which can only output one candidate, returning a list filled with one result is recommended.</td>
</tr>
</tbody>
</table>

7.6.11 Speech Synthesis

Figure 27: speech synthesis
Table 7.47: speech synthesis

Description: This is a component for generating synthesized speech. This component acts to generate synthesized speech by specifying the speech text.

This functionality is essential for human robot interactions. Naturally the robot talks to the user when it communicates with the user.

Here, we assume speech synthesis algorithm which can synthesize a voice in multiple characters (e.g. male, female, robotic). W3C-SSML format is used to specify the language and the prosodic parameters. For speech synthesis algorithm which cannot specify the prosodic parameters, XML tags in W3C-SSML format shall be skipped.

Command Method

<table>
<thead>
<tr>
<th>set_parameter</th>
<th>Specifies speech synthesis parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>argument</td>
<td></td>
</tr>
<tr>
<td>speech_text</td>
<td>String</td>
</tr>
<tr>
<td>SSML text</td>
<td>String [W3C-SSML]</td>
</tr>
<tr>
<td>volume</td>
<td>Integer</td>
</tr>
<tr>
<td>language</td>
<td>String [ISO639-1]</td>
</tr>
<tr>
<td>character</td>
<td>RoIS_Identifier</td>
</tr>
</tbody>
</table>

Query Method

<table>
<thead>
<tr>
<th>get_parameter</th>
<th>Obtains speech synthesis parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td></td>
</tr>
<tr>
<td>speech_text</td>
<td>String</td>
</tr>
<tr>
<td>SSML text</td>
<td>String [W3C-SSML]</td>
</tr>
<tr>
<td>volume</td>
<td>Integer</td>
</tr>
<tr>
<td>language</td>
<td>String [ISO639-1]</td>
</tr>
<tr>
<td>character</td>
<td>RoIS_Identifier</td>
</tr>
<tr>
<td>synthesizable_languages</td>
<td>Set&lt;String&gt; [ISO639-1]</td>
</tr>
<tr>
<td>synthesizable_characters</td>
<td>Set&lt;RoIS_Identifier&gt;</td>
</tr>
</tbody>
</table>

Condition: These elements shall be selected according to the speech text format.
**7.6.12 Reaction**

**Table 7.48: reaction**

Description: This is a component for executing specified reaction. This component acts to execute specified reaction by specifying the reaction ID.

This functionality is useful for human robot interactions. Generally it is difficult for the Service Application programmers to specify the robot reaction in detail since it depends on the hardware architecture. Therefore, this component provides a simple way to specify the robot reaction. For example, if the Service Application needs to express “yes”/“no” to the user, the Service Application programmer can execute the reaction only by specifying the reaction ID for “yes”/“no” reaction without regard for the expression method, such as nodding yes/no or showing a message for yes/no on its display.

The meaning of reaction is different among such as countries. Also the executable reactions may vary from robot to robot. The reaction shall be simply specified by reaction ID and the Service Application shall understand the meaning of the ID by the reference for the ID.

**Command Method**

<table>
<thead>
<tr>
<th>set_parameter</th>
<th>Specifies reaction parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>argument</td>
<td></td>
</tr>
<tr>
<td>reaction ref</td>
<td>RoIS_Identifier M</td>
</tr>
</tbody>
</table>

**Query Method**

| get_parameter | Obtains reaction parameters. |
### 7.6.13 Navigation

**Description:** This is a component for commanding navigation toward specified destinations. This component acts to move to the destination by specifying the position data of the destination. An HRI Engine (typically a robot) may include this component when the HRI Engine has the ability to move in the physical world.

Navigation function is essential for typical robotic services to specify the robot movement toward the destination. This component allows Service Applications to command robots to perform navigation without concerning the actual navigational device. Target position shall be specified as a list of spatial positions. The actual paths to be navigated between each position and strategies such as for path generation or for obstacle avoidance are left to the component implementation.

This component shall finish its operation when the robot arrives at the final position.

**Command Method**

<table>
<thead>
<tr>
<th>Command Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_parameter</td>
<td>Specifies parameters for navigation.</td>
</tr>
<tr>
<td>argument</td>
<td>target_position</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>argument</td>
<td>time_limit</td>
</tr>
<tr>
<td>argument</td>
<td>routing_policy</td>
</tr>
</tbody>
</table>

**Query Method**

| get_parameter       | Obtains parameters for navigation. |

<table>
<thead>
<tr>
<th>result</th>
<th>target_position</th>
<th>List&lt;Data&gt; [RLS]</th>
<th>M</th>
<th>List of specified target position data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>time_limit</td>
<td>Integer</td>
<td>O</td>
<td>Time limit for determining whether it is impossible to continue the navigation. The time shall be specified in millisecond.</td>
</tr>
<tr>
<td>result</td>
<td>routing_policy</td>
<td>String</td>
<td>O</td>
<td>Policy for determining the navigation route. For example, there may be the routing policies such as “time priority” or “distance priority”</td>
</tr>
</tbody>
</table>
7.6.14 Follow

Table 7.50: follow

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target_object_ref</td>
<td>RoIS_Identifier. The object is specified by object IDs. The reference information related to the ID shall be specified with each ID.</td>
</tr>
<tr>
<td>distance</td>
<td>Integer. Minimum distance between the target and the robot. When the robot comes closer than the limit distance, the robot suspends following. The distance shall be specified in millimeter.</td>
</tr>
<tr>
<td>time_limit</td>
<td>Integer. Time limit for determining whether it is impossible to continue following. If this parameter is not specified, the</td>
</tr>
</tbody>
</table>
default value may be used. The time shall be specified in milliseconds.

<table>
<thead>
<tr>
<th>Query Method</th>
<th>get_parameter</th>
<th>Obtains parameters for follow.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>result</strong></td>
<td><strong>target object ref</strong></td>
<td><strong>RoIS_Identifier</strong></td>
</tr>
<tr>
<td><strong>result</strong></td>
<td><strong>distance</strong></td>
<td><strong>Integer</strong></td>
</tr>
<tr>
<td><strong>result</strong></td>
<td><strong>time_limit</strong></td>
<td><strong>Integer</strong></td>
</tr>
</tbody>
</table>

### 7.6.15 Move

![Class Diagram for Move](image)

**Figure 31: move**

**Table 7.51: move**

Description: This is a component for moving based on a specified motion. The motion is simply specified by a line or a curve. An HRI Engine (typically a robot) may include this component when the HRI Engine has the ability to move in the physical world.
Move function is essential for typical robotic services to specify a little motion for moving over a little from the current position. This component shall finish its operation when the specified motion finishes.

**Command Method**

<table>
<thead>
<tr>
<th>set_parameter</th>
<th>Specifies parameters for move.</th>
</tr>
</thead>
<tbody>
<tr>
<td>argument line</td>
<td>List&lt;Integer&gt;</td>
</tr>
<tr>
<td>argument curve</td>
<td>List&lt;Integer&gt;</td>
</tr>
<tr>
<td>argument time</td>
<td>Integer</td>
</tr>
</tbody>
</table>

- **argument line**
  - List<Integer> `C`
  - Distance and orientation for specifying the line. The distance shall be specified in millimeter and the orientation shall be specified in degree.

- **argument curve**
  - List<Integer> `C`
  - Radius and direction for specify the curve. The radius shall be specified in millimeter and the direction shall be specified in degree.

- **argument time**
  - Integer `O`
  - Operating time for the motion. The time shall be specified in milliseconds.

**Query Method**

<table>
<thead>
<tr>
<th>get_parameter</th>
<th>Obtains parameters for move.</th>
</tr>
</thead>
<tbody>
<tr>
<td>result line</td>
<td>List&lt;Integer&gt;</td>
</tr>
<tr>
<td>result curve</td>
<td>List&lt;Integer&gt;</td>
</tr>
<tr>
<td>result time</td>
<td>Integer</td>
</tr>
</tbody>
</table>

- **result line**
  - List<Integer> `C`
  - Specified distance and orientation for specifying the line. The distance shall be specified in millimeter and the orientation shall be specified in degree.

- **result curve**
  - List<Integer> `C`
  - Specified radius and direction for specify the curve. The radius shall be specified in millimeter and the direction shall be specified in degree.

- **result time**
  - Integer `O`
  - Specified operating time. The time shall be specified in milliseconds.

Condition: These elements shall be selected according to the motion.
7.7 Platform Specific Model

7.7.1 C++ PSM

/**************************************************************/
/* RoIS_HRI.h (for HRI Engine) */
/**************************************************************/
#include <vector>
#include <string>

namespace RoIS_HRI
{
    enum ReturnCode_t
    {
        OK,
        ERROR,
        BAD_PARAMETER,
        UNSUPPORTED,
        OUT_OF_RESOURCES,
        TIMEOUT
    };
    typedef std::string RoIS_Identifier;
    typedef std::vector<RoIS_Identifier> RoIS_IdentifierList;
    typedef std::string Condition_t;
    typedef std::string HRI_Engine_Profile;
    typedef std::string CommandUnitSequence;
    struct Result
    {
        std::string name;
        RoIS_Identifier data_type_ref;
        std::string value;
    };
    struct Parameter
    {
        std::string name;
        RoIS_Identifier data_type_ref;
        std::string value;
    };
    struct Argument
    {
        std::string name;
        RoIS_Identifier data_type_ref;
        std::string value;
    };
    typedef std::vector<Result> ResultList;
    typedef std::vector<Parameter> ParameterList;
    typedef std::vector<Argument> ArgumentList;

    /* For System Interface */
    class SystemIF
    {
    public:
        ReturnCode_t connect();
        ReturnCode_t disconnect();
        ReturnCode_t get_profile(
            Condition_t condition,
            HRI_Engine_Profile& profile
        );
    };
}
ReturnCode_t get_error_detail(
    std::string error_id,
    Condition_t condition,
    ResultList& results
):
}

/* For Command Interface */
class CommandIF{
public:
    ReturnCode_t search(
        Condition_t condition,
        RoIS_IdentifierList& component_ref_list);
    ReturnCode_t bind(
        RoIS_Identifier component_ref);
    ReturnCode_t bind_any(
        Condition_t condition,
        RoIS_Identifier& component_ref);
    ReturnCode_t release(
        RoIS_Identifier component_ref);
    ReturnCode_t get_parameter(
        RoIS_Identifier component_ref,
        ParameterList& parameters);
    ReturnCode_t set_parameter(
        RoIS_Identifier component_ref,
        ParameterList parameters,
        std::string& command_id);
    ReturnCode_t execute(
        CommandUnitSequence command_unit_list);
    ReturnCode_t get_command_result(
        std::string command_id,
        Condition_t condition,
        ResultList& results);
};

/* For Query Interface */
class QueryIF{
public:
    ReturnCode_t query(
        std::string query_type,
        Condition_t condition,
        ResultList& results);
};

/* For Event Interface */
class EventIF{
public:
    ReturnCode_t subscribe(

std::string event_type,
    Condition_t condition,
    std::string& subscribe_id
);  
    ReturnCode_t unsubscribe(
    std::string subscribe_id
);  
    ReturnCode_t get_event_detail(
    std::string event_id,
    Condition_t condition,
    ResultList& results
    );
};
};
}

/**************************
RoIS_Service.h (for Service Application) */
/*****************************/
#include <vector>
#include <string>
namespace RoIS_Service
{
    enum Completed_Status
    {
        OK,
        ERROR,
        ABORT,
        OUT_OF_RESOURCES,
        TIMEOUT
    };  
    enum ErrorType
    {
        ENGINE_INTERNAL_ERROR,
        COMPONENT_INTERNAL_ERROR,
        COMPONENT_NOT_RESPONDING,
        USER_DEFINED_ERROR
    };

    /* For Service Application Interface */
    class ServiceApplicationBase{
    public:
        void notify_error( 
        std::string error_id,
        ErrorType error_type 
        );
        void completed(
        std::string command_id,
        Completed_Status status
        );
        void notify_event( 
        std::string event_id,
        std::string event_type,
        std::string subscribe_id,
        DateTime expire 
        );
};
namespace RoIS_Common{
enum Component_Status
{
    UNINITIALIZED,
    READY,
    BUSY,
    WARNING,
    ERROR
};
class Command{
pUBLIC:
    virtual ReturnCode_t start();
    virtual ReturnCode_t stop();
    virtual ReturnCode_t suspend();
    virtual ReturnCode_t resume();
};
class Query{
pUBLIC:
    virtual ReturnCode_t component_status(
        Component_Status& status,
    );
};
class Event{
};
};
namespace RoIS_System_Information{
class Query {
pUBLIC:
    ReturnCode_t robot_position{
        DateTime& timestamp,
        RoIS_IdentifierList& robot_ref,
        std::vector<RoLo::Architecture::Data>& position_data
    };
    ReturnCode_t engine_status{
        Component_Status& status,
        DateTime& operatable_time
    };
};
}
#include <RoIS_Common.h>
namespace Person_Detection
{
    class Command : public RoIS_Common::Command {
    public:
        void person_detected(
            DateTime timestamp,
            Integer number
        );
    };
    class Query : public RoIS_Common::Query {
    public:
        void person_localized(
            DateTime timestamp,
            RoIS_IdentifierList person_ref,
            std::vector<RoLo::Architecture::Data> position_data
        );
    };
    class Event : public RoIS_Common::Event {
    public:
        void person_detected(
            DateTime timestamp,
            Integer number
        );
    };
};

#include <RoIS_Common.h>
#include <RLS/Architecture.hpp>
namespace Person_Localization
{
    class Command : public RoIS_Common::Command {
    public:
        ReturnCode_t set_parameter(
            Integer detection_threshold,
            Integer minimum_interval
        );
    };
    class Query : public RoIS_Common::Query {
    public:
        ReturnCode_t get_parameter(
            Integer& detection_threshold,
            Integer& minimum_interval
        );
    };
    class Event : public RoIS_Common::Event {
    public:
        void person_localized(
            DateTime timestamp,
            RoIS_IdentifierList person_ref,
            std::vector<RoLo::Architecture::Data> position_data
        );
    };
};

#include <RoIS_Common.h>
namespace Person_Identification
{
class Command : public RoIS_Common::Command{
};
class Query : public RoIS_Common::Query{
};
class Event : public RoIS_Common::Event{
public:
    void person_identified(
        DateTime timestamp,
        RoIS_IdentifierList person_ref,
    );
};

/*********************************/
/* RoIS_Person_Identification.h */
/*********************************/
#include <RoIS_Common.h>
namespace Person_Identification
{
    class Command : public RoIS_Common::Command{
    };
    class Query : public RoIS_Common::Query{
    };
    class Event : public RoIS_Common::Event{
public:
    void person_identified(
        DateTime timestamp,
        RoIS_IdentifierList person_ref,
    );
    };
};

/*********************************/
/* RoIS_Face_Detection.h */
/*********************************/
#include <RoIS_Common.h>
namespace Face_Detection
{
    class Command : public RoIS_Common::Command{
    };
    class Query : public RoIS_Common::Query{
    };
    class Event : public RoIS_Common::Event{
public:
    void face_detected(
        DateTime timestamp,
        Integer number
    );
    };
};

/*********************************/
/* RoIS_Face_Localization.h */
/*********************************/
#include <RLS/Architecture.hpp>
#include <Rs/Architecture.hpp>
/* http://www.omg.org/spec/RLS/20090601/Architecture.hpp */
namespace Face_Localization
{
    class Command : public RoIS_Common::Command{
        ReturnCode_t set_parameter(
            Integer detection-threshold,
            integer minimum-interval
        );
    };
    class Query : public RoIS_Common::Query{
        ReturnCode_t get_parameter(
            Integer& detection-threshold,
            Integer& minimum-interval
        );
    };
    class Event : public RoIS_Common::Event{
        public:
            void face_localized(
                DateTime timestamp,
                RoIS_IdentifierList face_ref,
                std::vector<RoLo::Architecture::Data> position_data
            );
    };
};

.isTrue/**/  /* RoIS_Sound_Detection.h */
.isTrue/**/  /* include <RoIS_Common.h> */
namespace Sound_Detection
{
    class Command : public RoIS_Common::Command{
    };
    class Query : public RoIS_Common::Query{
    };
    class Event : public RoIS_Common::Event{
        public:
            void sound_detected(
                DateTime timestamp,
                Integer number
            );
    };
};

.isTrue/**/  /* RoIS_Sound_Localization.h */
.isTrue/**/  /* include <RoIS_Common.h> */
.isTrue/**/  /* include <RLS/Architecture.hpp> */
.isTrue/**/  /* http://www.omg.org/spec/RLS/20090601/Architecture.hpp */
namespace Sound_Localization
{
    class Command : public RoIS_Common::Command{
        ReturnCode_t set_parameter(
            Integer detection-threshold,
            Integer minimum-interval
        );
    };
}
class Query : public RoIS_Common::Query{
    ReturnCode_t get_parameter(
        Integer& detection_threshold,
        Integer& minimum_interval
    );
};
class Event : public RoIS_Common::Event{
public:
    void sound_located(
        DateTime timestamp,
        RoIS_IdentifierList sound_ref,
        std::vector<RoLo::Architecture::Data> position_data
    );
};

/**************************************/
/* RoIS_Speech_Recognition.h */
/**************************************/
#include <RoIS_Common.h>
namespace Speech_Recognition
{
    class Command : public RoIS_Common::Command{
        ReturnCode_t set_parameter(
            std::vector<std::string> languages,
            std::string grammer,
            std::string rule
        );
    };
    class Query : public RoIS_Common::Query{
        public:
            ReturnCode_t get_parameter(
                std::vector<std::string>& recognizable_languages,
                std::vector<std::string>& languages,
                std::string& grammer,
                std::string& rule
            );
    };
    class Event : public RoIS_Common::Event{
public:
        void speech_recognized(
            DateTime timestamp,
            vector<std::string> recognized_text
        );
        void speech_input_started(
            DateTime timestamp
        );
        void speech_input_finished(
            DateTime timestamp
        );
    };
};

/***************************************/
/* RoIS_Gesture_Recognition.h */
/***************************************/
#include <RoIS_Common.h>
namespace Gesture_Recognition
{
    class Command : public RoIS_Common::Command{
    };
    class Query : public RoIS_Common::Query{
        public:
            ReturnCode_t get_parameter(
                RoIS_IdentifierList& recognizable_gestures
            );
    };
    class Event : public RoIS_Common::Event{
        public:
            void gesture_recognized(
                DateTime timestamp,
                RoIS_IdentifierList gesture_ref
            );
    };
}/* RoIS_Speech_Synthesis.h */
/*******************************************************************************/
#include <RoIS_Common.h>
namespace Speech_Synthesis
{
    class Command : public RoIS_Common::Command{
        public:
            ReturnCode_t set_parameter(
                std::string SSML_text,
                std::string speech_text,
                Integer volume,
                std::string language,
                RoIS_Identifier character
            );
    };
    class Query : public RoIS_Common::Query{
        public:
            ReturnCode_t get_parameter(
                std::string& speech_text,
                std::string& SSML_text,
                Integer& volume,
                RoIS_Identifier& character,
                vector<std::string>& synthesizable_languages,
                RoIS_IdentifierList& synthesizable_characters
            );
    };
    class Event : public RoIS_Common::Event{
    };
}="/******************************************************************************/

/****** RoIS_Reaction.h */
/*********************************************************************************/
#include <RoIS_Common.h>
namespace Reaction
{
    class Command : public RoIS_Common::Command{
        public:
            ReturnCode_t set_parameter(
                RoIS_IdentifierList reaction_ref
            );
    }
    class Query : public RoIS_Common::Query{
        public:
            ReturnCode_t get_parameter(
                RoIS_IdentifierList& available_reactions,
                RoIS_Identifier& reaction_ref
            );
    }
    class Event : public RoIS_Common::Event{
    }
};

/**************************
/ * RoIS_Navigation.h  */
/**************************
#include <RoIS_Common.h>
#include <RLS/Architecture.hpp>
/* http://www.omg.org/spec/RLS/20090601/Architecture.hpp */
namespace Navigation
{
    class Command : public RoIS_Common::Command{
        public:
            ReturnCode_t set_parameter(
                vector<RoLo::Architecture::Data> target_position,
                Integer time_limit,
                std::string routing_policy
            );
    }
    class Query : public RoIS_Common::Query{
        public:
            ReturnCode_t get_parameter(
                vector<RoLo::Architecture::Data>& target_position,
                Integer& time_limit,
                std::string& routing_policy
            );
    }
    class Event : public RoIS_Common::Event{
    }
};

/**************************
/ * RoIS_Follow.h  */
/**************************
#include <RoIS_Common.h>
namespace Follow
{
    class Command : public RoIS_Common::Command{
        public:
            ReturnCode_t set_parameter(
7.7.2 CORBA PSM

CORBA IDL for this framework is given as follows:

```cpp
#include <RoIS_Common.h>
namespace Move{
    class Command : public RoIS_Common::Command{
        public:
            ReturnCode_t set_parameter(
                List<Integer> line,    
                List<Integer> curve,   
                Integer time
            );
    };
    class Query : public RoIS_Common::Query{
        public:
            ReturnCode_t get_parameter(
                List<Integer>& line,    
                List<Integer>& curve,   
                Integer& time
            );
    };
    class Event : public RoIS_Common::Event{
    };
};
/**--------------------------*/
/* RoIS_Move.h */
/**--------------------------*/
#include <RoIS_Common.h>
namespace Move{
    class Command : public RoIS_Common::Command{
        public:
            ReturnCode_t set_parameter(
                List<Integer> line,    
                List<Integer> curve,   
                Integer time
            );
    };
    class Query : public RoIS_Common::Query{
        public:
            ReturnCode_t get_parameter(
                List<Integer>& line,    
                List<Integer>& curve,   
                Integer& time
            );
    };
    class Event : public RoIS_Common::Event{
    };
};
/**--------------------------*/
/* RoIS_HRI.idl (for HRI Engine) */
/**--------------------------*/
module RoIS_HRI{
    module RoIS_HRI{
    };
}
enum ReturnCode_t
{
    OK,
    ERROR,
    BAD_PARAMETER,
    UNSUPPORTED,
    OUT_OF_RESOURCES,
    TIMEOUT
};
typedef String RoIS_Identifier;
typedef sequence<RoIS_Identifier> RoIS_IdentifierList;
typedef String Condition_t;
typedef String HRI_Engine_Profile;
typedef String CommandUnitSequence;
struct Result {
    String name;
    RoIS_Identifier data_type_ref;
    any value;
};
struct Parameter {
    String name;
    RoIS_Identifier data_type_ref;
    any value;
};
struct Argument {
    String name;
    RoIS_Identifier data_type_ref;
    any value;
};
typedef sequence<Result> ResultList;
typedef sequence<Parameter> ParameterList;
typedef sequence<Argument> ArgumentList;

/* For System Interface */
interface SystemIF{
    ReturnCode_t connect();
    ReturnCode_t disconnect();
    ReturnCode_t get_profile(
        in Condition_t condition,
        out HRI_Engine_Profile profile
    );
    ReturnCode_t get_error_detail(
        in String error_id,
        in Condition_t condition,
        out ResultList results
    );
};

/* For Command Interface */
interface CommandIF{
    ReturnCode_t search(
        in Condition_t condition,
        out RoIS_IdentifierList component_ref_list
    );
    ReturnCode_t bind(
        in RoIS_Identifier component_ref
    );
ReturnCode_t bind_any(
    in Condition_t condition,
    out RoIS_Identifier component_ref
);
ReturnCode_t release(
    in RoIS_Identifier component_ref
);
ReturnCode_t get_parameter(
    in RoIS_Identifier component_ref,
    out ParameterList parameters
);
ReturnCode_t set_parameter(
    in RoIS_Identifier component_ref,
    in ParameterList parameters,
    out String command_id
);
ReturnCode_t execute(
    in CommandUnitSequence command_unit_list
);
ReturnCode_t get_command_result(
    in String command_id,
    in Condition_t condition,
    out ResultList results
);

/* For Query Interface */
interface QueryIF{
    ReturnCode_t query(
        in String query_type,
        in Condition_t condition,
        out ResultList results
    );
};

/* For Event Interface */
interface EventIF{
    ReturnCode_t subscribe(
        in String event_type,
        in Condition_t condition,
        out String subscribe_id
    );
    ReturnCode_t unsubscribe(
        in String subscribe_id
    );
    ReturnCode_t get_event_detail(
        in String event_id,
        in Condition_t condition,
        out ResultList results
    );
};

FLICTS

/* RoIS_Service.idl (for Service Application) */
module RoIS_Service
{
enum Completed_Status
{
    OK,
    ERROR,
    ABORT,
    OUT_OF_RESOURCES,
    TIMEOUT
};
enum ErrorType
{
    ENGINE_INTERNAL_ERROR,
    COMPONENT_INTERNAL_ERROR,
    COMPONENT_NOT_RESPONDING,
    USER_DEFINED_ERROR
};
/
/* For Service Application Interface */
/
interface ServiceApplicationBase{
    void notify_error(
        in String error_id,
        in ErrorType error_type
    );
    void completed(
        in String command_id,
        in Completed_Status status
    );
    void notify_event(
        in String event_id,
        in String event_type,
        in String subscribe_id,
        in DateTime_t expire
    );
};
}/

module RoIS_Common{
enum Component_Status
{
    UNINITIALIZED,
    READY,
    BUSY,
    WARNING,
    ERROR
}
interface Command{
    ReturnCode_t start();
    ReturnCode_t stop();
    ReturnCode_t suspend();
    ReturnCode_t resume();
};
interface Query{
ReturnCode_t component_status(
    out Component_Status status
);
}

interface Event{
};

/******************************************************************************
/* RoIS_System_Information.idl */
/******************************************************************************
module System_Information{
interface Query{
    ReturnCode_t robot_position{
        out DateTime timestamp,
        out sequence<RoIS_Identifier> robot_ref,
        out sequence<RoLo::Architecture::Data> position_data
    };
    ReturnCode_t engine_status{
        out Component_Status status,
        out DateTime operatable_time
    };
};
};

/******************************************************************************
/* RoIS_Person_Detection.idl */
/******************************************************************************
module Person_Detection{
interface Command : RoIS_Common::Command{
};
interface Query : RoIS_Common::Query{
};
interface Event : RoIS_Common::Event{
    void person_detected(
        in DateTime timestamp,
        in Integer number
    );
};
};

/******************************************************************************
/* RoIS_Person_Localization.idl */
/******************************************************************************
module Person_Localization{
interface Command : RoIS_Common::Command{

    ReturnCode_t set_parameter(
        in Integer detection_threshold,
        in Integer minimum_interval
    );
};
interface Query : RoIS_Common::Query{

    ReturnCode_t get_parameter(

out Integer detection_threshold,
out Integer minimum_interval

);}
interface Event : RoIS_Common::Event{
    void person_localized(
        in DateTime timestamp,
        in RoIS_IdentifierList person_ref,
        in sequence<RoLo::Architecture::Data> position_data
    );
};

/****************************************/
/* RoIS_Person_Identification.idl */
/****************************************/
module Person_Identification
{
    interface Command : RoIS_Common::Command{
    
    }
    
    interface Query : RoIS_Common::Query{
    
    }
    
    interface Event : public RoIS_Common::Event{
        void person_identified(
            in DateTime timestamp,
            in RoIS_IdentifierList person_ref,
        );
    }
};

/**********************************
/* RoIS_Face_Detection.idl */
/**********************************/
module Face_Detection
{
    interface Command : RoIS_Common::Command{
    
    }
    
    interface Query : RoIS_Common::Query{
    
    }
    
    interface Event : RoIS_Common::Event{
        void face_detected(
            in DateTime timestamp,
            in Integer number
        );
    }
};

/**********************************
/* RoIS_Face_Localization.idl */
/**********************************/
module Face_Localization
{
    interface Command : RoIS_Common::Command{
        
    }
    
    interface Query : RoIS_Common::Query{
    
    }
    
    interface Event : RoIS_Common::Event{
        ReturnCode_t set_parameter(
            in Integer detection_threshold,
            in Integer minimum_interval
        );
    }
}
interface Query : RoIS_Common::Query{
    ReturnCode_t get_parameter(
        out Integer detection_threshold,
        out Integer minimum_interval
    );
};

interface Event : RoIS_Common::Event{
    void face_localized(
        in DateTime timestamp,
        in RoIS_IdentifierList face_ref,
        in sequence<RoLo::Architecture::Data> position_data
    );
};

/**********************************
/* RoIS_Sound_Detection.idl */
***********************************/
module Sound_Detection
{
    interface Command : RoIS_Common::Command{
    }
    interface Query : RoIS_Common::Query{
    }
    interface Event : RoIS_Common::Event{
        void sound_detected(
            in DateTime timestamp,
            in Integer number
        );
    }
};

/**********************************
/* RoIS_Sound_Localization.idl */
***********************************/
module Sound_Localization
{
    interface Command : RoIS_Common::Command{
        ReturnCode_t set_parameter(
            in Integer detection_threshold,
            in Integer minimum_interval
        );
    }
    interface Query : RoIS_Common::Query{
        ReturnCode_t get_parameter(
            out Integer detection_threshold,
            out Integer minimum_interval
        );
    }
    interface Event : RoIS_Common::Event{
        void sound_localized(
            in DateTime timestamp,
            in RoIS_IdentifierList sound_ref,
            in sequence<RoLo::Architecture::Data> position_data
        );
    }
};
module Speech_Recognition
{
    interface Command : RoIS_Common::Command{
        ReturnCode_t set_parameter(
            in sequence<String> languages,
            in String grammar,
            in String rule
        );
    };
    interface Query : RoIS_Common::Query{
        ReturnCode_t get_parameter(
            out sequence<String> recognizable_languages,
            out sequence<String> languages,
            out String grammar,
            out String rule
        );
    };
    interface Event : RoIS_Common::Event{
        void speech_recognized(
            in DateTime timestamp,
            in sequence<String> recognized_text
        );
        void speech_input_started(
            in DateTime timestamp
        );
        void speech_input_finished(
            in DateTime timestamp
        );
    };
};

module Gesture_Recognition
{
    interface Command : RoIS_Common::Command{
    };
    interface Query : RoIS_Common::Query{
        ReturnCode_t get_parameter(
            out RoIS_IdentifierList recognizable_gestures
        );
    };
    interface Event : RoIS_Common::Event{
        void gesture_recognized(
            in DateTime timestamp,
            in RoIS_IdentifierList gesture_ref
        );
    };
};
module Speech_Synthesis
{
    interface Command : RoIS_Common::Command{
        ReturnCode_t set_parameter(
            in String speech_text,
            in String SSML_text,
            in Integer volume,
            in String language,
            in RoIS_Identifier character
        );
    };
    interface Query : RoIS_Common::Query{
        ReturnCode_t get_parameter(
            out String speech_text,
            out String SSML_text,
            out Integer volume,
            out String language,
            out RoIS_Identifier character,
            out sequence<String> synthesizable_languages,
            out sequence<RoIS_Identifier> synthesizable_characters
        );
    };
    interface Event : RoIS_Common::Event{
    };
};

module Reaction
{
    interface Command : RoIS_Common::Command{
        ReturnCode_t set_parameter(
            in RoIS_IdentifierList reaction_ref
        );
    };
    interface Query : RoIS_Common::Query{
        ReturnCode_t get_parameter(
            out RoIS_IdentifierList available_reactions,
            out RoIS_Identifier reaction_ref
        );
    };
    interface Event : RoIS_Common::Event{
    };
};

module Navigation
interface Command : RoIS_Common::Command{
    ReturnCode_t set_parameter(
        in sequence target_position,
        in Integer time_limit,
        in String routing_policy
    );
};
interface Query : RoIS_Common::Query{
    ReturnCode_t get_parameter(
        out sequence<RoLo::Architecture::Data> target_position,
        out Integer time_limit,
        out String routing_policy
    );
};
interface Event : RoIS_Common::Event{
};
/
/**********************
/* RoIS_Follow.idl */
/**********************
module Follow
{
    interface Command : RoIS_Common::Command{
        ReturnCode_t set_parameter(
            in RoIS_Identifier target_object_ref,
            in Integer distance,
            in Integer time_limit
        );
    };
    interface Query : RoIS_Common::Query{
        ReturnCode_t get_parameter(
            out RoIS_Identifier target_object_ref,
            out Integer distance,
            out Integer time_limit
        );
    };
}/
/**************************
/* RoIS_Move.idl */
/**************************
module Move
{
    interface Command : RoIS_Common::Command{
        ReturnCode_t set_parameter(
            in sequence<Integer> line,
            in sequence<Integer> curve,
            in Integer time
        );
    };
    interface Query : RoIS_Common::Query{
        ReturnCode_t get_parameter(
            out sequence<Integer> line,
            out sequence<Integer> curve,
            out Integer time
        );
    };
}
interface Event : RoIS_Common::Event{

};

};

7.7.3 XML PSM

XML schema for this framework is given as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:rois="http://www.omg.org/rois/201206"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    targetNamespace="http://www.omg.org/rois/201206"
    elementFormDefault="qualified" attributeFormDefault="qualified">
    <xsd:import namespace="http://www.opengis.net/gml/3.2" schemaLocation="http://schemas.opengis.net/gml/3.2.1/gml.xsd"/>
    <!-- Profile -->
    <xsd:complexType name="RoISIdentifierType">
        <xsd:attribute name="authority" type="xsd:string" use="optional"/>
        <xsd:attribute name="code" type="xsd:string" use="required"/>
        <xsd:attribute name="codebook_ref" type="xsd:string" use="optional"/>
        <xsd:attribute name="version" type="xsd:string" use="optional"/>
    </xsd:complexType>
    <xsd:element name="HRIEngineProfile" type="rois:HRIEngineProfileType"/>
    <xsd:complexType name="HRIEngineProfileType">
        <xsd:complexContent>
            <xsd:extension base="gml:IdentifiedObjectType">
                <xsd:sequence>
                    <xsd:element name="SubProfile" type="rois:HRIEngineProfileType" minOccurs="0" maxOccurs="unbounded"/>
                    <xsd:element name="HRIComponent" type="xsd:ID" minOccurs="1" maxOccurs="unbounded"/>
                    <xsd:element ref="rois:ParameterProfile" minOccurs="0" maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="HRIComponentProfile" type="rois:HRIComponentProfileType"/>
    <xsd:complexType name="HRIComponentProfileType">
        <xsd:complexContent>
            <xsd:extension base="gml:IdentifiedObjectType">
                <xsd:sequence>
                    <xsd:element name="SubComponentProfile" type="xsd:ID" minOccurs="0" maxOccurs="unbounded"/>
                    <xsd:element name="MessageProfile" type="rois:MessageProfileType" minOccurs="1" maxOccurs="unbounded"/>
                    <xsd:element ref="rois:ParameterProfile" minOccurs="0" maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:element name="ParameterProfile" type="rois:ParameterProfileType"/>
    <xsd:complexType name="ParameterProfileType">
```
<xsd:complexType>
  <xsd:element name="CommandUnitSequence" type="rois:CommandUnitSequenceType"/>
  <xsd:complexType name="CommandUnitSequenceType">
    <xsd:sequence>
      <xsd:element name="command_unit_list" type="rois:CommandBaseType" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:complexType>

<xsd:element name="CommandBase" type="rois:CommandBaseType" abstract="true"/>
<xsd:complexType name="CommandBaseType">
  <xsd:attribute name="delay_time" type="xsd:integer" use="optional"/>
</xsd:complexType>

<xsd:element name="CommandMessageType" substitutionGroup="rois:CommandBase"/>
<xsd:complexType name="CommandMessageType">
  <xsd:complexContent>
    <xsd:extension base="rois:CommandBaseType">
      <xsd:sequence>
        <xsd:element name="component_ref" type="rois:RoISIdentifierType" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="arguments" type="rois:ArgumentListType" minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="command_id" type="xsd:string" use="required"/>
      <xsd:attribute name="command_type" type="xsd:string" use="required"/>
    </xsd:extension>
  </xsd:complexType>
</xsd:complexType>

<xsd:element name="ConcurrentCommands" type="rois:ConcurrentCommandsType" substitutionGroup="rois:CommandBase"/>
<xsd:complexType name="ConcurrentCommandsType">
  <xsd:complexContent>
    <xsd:extension base="rois:CommandBaseType">
      <xsd:sequence>
        <xsd:element name="command_list" type="rois:CommandMessageType" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexType>
</xsd:complexType>

<xsd:element name="Branch" type="rois:BranchType"/>
<xsd:complexType name="BranchType">
  <xsd:sequence>
    <xsd:element name="command_list" type="rois:CommandMessageType" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>
Part II

Annex A Examples of Profile in XML (informative)

The following shows examples of describing each type of profile in XML.

A.1 Parameter Profile

This is an example of a Parameter Profile for a parameter described in XML.

```
<rois:ParameterProfile rois:description="Maximum detectable number of person" rois:default_value="10"
                   rois:name="max_number">
    <rois:data_type_ref rois:code="urn:x-rois:def:DataType:ATR::Integer"/>
</rois:ParameterProfile>
```

This Parameter Profile defines the maximum detectable number of persons as a parameter in the person detection function. This parameter is defined as a parameter exchanged by RoIS interface method, such as the argument parameter for ‘set_parameter()’ and the result parameter for ‘get_parameter()’.

The parameter name is defined as ‘max_number’ in the attribute ‘rois:name’ of the <rois:ParameterProfile> tag, and a description of this parameter is given in the attribute ‘rois:description’. In addition, when a default value for the parameter is specified, the value can be specified using the attribute ‘rois:default_value’ in the <rois:ParameterProfile> tag. Data type of the parameter is specified using the <rois:data_type_ref> tag within the <rois:ParameterProfile> tag. Here, the data type of ‘max_number’ is defined as ‘urn:x-rois:def:DataType:ATR::Integer’in the attribute ‘rois:code’ of the <rois:data_type_ref> tag.

Note that ‘data_type_ref’ is an ID used for referencing a separately defined data type. Here, for example, ‘urn:x-rois:def:DataType:ATR::Integer’ in the data type list is defined as integer type.

A.2 Message Profile

A.2.1 Command Message Profile

This is an example of a Message Profile for a message used in the Command Interface described in XML.

```
<rois:MessageProfile xsi:type="rois:CommandMessageProfileType" rois:name="change_speech_speed">
    <rois:Arguments rois:description="utterance speed" rois:name="speed">
        <rois:data_type_ref rois:code="urn:x-rois:def:DataType:ATR::Integer"/>
    </rois:Arguments>
</rois:MessageProfile>
```

This Message Profile defines a command message for change rate of speech in the speech synthesis component. Message Profile is defined using <rois:MessageProfile> . When the message is used in the Command Interface, the type of the message is specified as ‘rois:CommandMessageProfileType’ in the attribute ‘xsi:type’ of the <rois:MessageProfile> tag.
The message name is defined as ‘change_speech_speed’ in the ‘rois:name’ attribute of the <rois:MessageProfile> tag. In a Command Message Profile, an argument parameter for a command message is defined using a <rois:Arguments> tag within the <rois:CommandMessageProfile> tag. The description form of <rois:Arguments> follows the Parameter Profile.

Here, an integer parameter is defined as the argument parameter when issuing the command message. The parameter name is defined as “speed” in the attribute ‘rois:name’ of the <rois:Arguments> tag, and a description of this parameter is given in the attribute ‘rois:description’. In addition, the <rois:data_type_ref> tag within the <rois:Arguments> tag defines the data type as ‘urn:x-rois:def:DataType:ATR::Integer’.

A.2.2 Event Message Profile

This is an example of a Message Profile for a message used in the Event Interface described in XML.

```xml
<rois:MessageProfile xsi:type="rois:EventMessageProfileType" rois:name="person_detected">
    <rois:Results rois:name="timestamp">
        <rois:data_type_ref rois:code="urn:x-rois:def:DataType:ATR::DateTime"/>
    </rois:Results>
    <rois:Results rois:name="number">
        <rois:data_type_ref rois:code="urn:x-rois:def:DataType:ATR::Integer"/>
    </rois:Results>
</rois:MessageProfile>
```

This Message Profile defines an event message notifying that a person has been detected in the person detection component. Message Profile is defined using <rois:MessageProfile>. When the message is used in the Event Interface, the type of the message is specified as ‘rois:EventMessageProfileType’ in the attribute ‘xsi:type’ of the <rois:MessageProfile> tag.

The message name is defined as ‘person_detected’ in the attribute ‘rois:name’ of the <rois:MessageProfile> tag. In an event message, a result parameter used in ‘get_event_detail()’ performed in conjunction with event notification is defined using a <Results> tag within the <rois:MessageProfile> tag. The description form of <rois:Results> follows the Parameter Profile.

Two parameters are defined here for the result parameters. Each definition uses the attribute ‘rois:name’ of the <rois:Results> tag and the <rois:data_type_ref> tag within the <rois:Results> tag for defining the result parameter name and the data type, respectively. Specifically, the data type indicating detection time is defined as ‘urn:x-rois:def:DataType:ATR::DateTime’ for the result parameter ‘timestamp’ and that indicating the number of the detected person is defined as ‘urn:x-rois:def:DataType:ATR::Integer’ for the result parameter ‘number’.

Note that data_type_ref is an ID used for referencing a separately defined data type. Here, for example, ‘urn:x-rois:def:DataType:ATR::DateTime’ in the data type list is defined as DateTime_type.

A.2.3 Query Message Profile

This is an example of a Message Profile for a message used in the Query Interface described in XML.

```xml
<rois:MessageProfile xsi:type="rois:QueryMessageProfileType" rois:name="engine_status">
    <rois:Results rois:name="status">
        <rois:data_type_ref rois:code="urn:x-rois:def:DataType:ATR::Component_Status"/>
    </rois:Results>
</rois:MessageProfile>
```

This Message Profile defines a query message used in the Query Interface. Message Profile is defined using <rois:MessageProfile>. When the message is used in the Query Interface, the type of the message is specified as ‘rois:QueryMessageProfileType’ in the attribute ‘xsi:type’ of the <rois:MessageProfile> tag.

The message name is defined as ‘engine_status’ in the attribute ‘rois:name’ of the <rois:MessageProfile> tag. In a Query Message Profile, a query parameter used in ‘get_query_detail()’ performed in conjunction with query execution is defined using a <Results> tag within the <rois:MessageProfile> tag. The description form of <rois:Results> follows the Parameter Profile.
This Message Profile defines a basic message for performing a query on HRI Engine status.

Message Profile is defined using `<rois:MessageProfile>` . When the message is used in the Query Interface, the type of the message is specified as ‘rois:QueryMessageProfileType’ in the attribute ‘xsi:type’ of the `<rois:MessageProfile>` tag.

The message name is defined as ‘engine_status’ in the attribute ‘rois:name’ of the `<rois:MessageProfile>` tag. In a Query Message Profile, a result parameter used in ‘query()’ is defined using the `<rois:Results>` tag within the `<rois:MessageProfile>` tag. The description form of `<rois:Results>` follows the Parameter Profile.

Two result parameters are defined in this profile, i.e., the status and the operable time of the HRI Engine. The names of these result parameters are defined using the attribute ‘rois:name’ of `<rois:Results>` tag and `<rois:data_type_ref>` tag within the `<rois:Results>` tag, respectively. The data type for these result parameters are defined as ‘urn:x-rois:def:DataType:ATR::Component_Status’ and ‘urn:x-rois:def:DataType:ATR::DateTime’ by using `<rois:data_type_ref>` tag.

Note that data_type_ref is an ID used for referencing a separately defined data type. In this case, ‘urn:x-rois:def:DataType:ATR::Component_Status’ in the data type list is defined as Component_Status type.

### A.3 HRI Component Profile

This is an example of an HRI Component Profile described in XML.

```xml
<rois:HRIComponentProfile >
  <gml:identifier>urn:x-rois:def:HRIComponent:ATR::PersonDetection</gml:identifier>
  <gml:name>person_detection</gml:name>

// ===== Command Message ======
<rois:MessageProfile xsi:type="rois:CommandMessageProfileType" rois:name="start"/>
<rois:MessageProfile xsi:type="rois:CommandMessageProfileType" rois:name="stop"/>
<rois:MessageProfile xsi:type="rois:CommandMessageProfileType" rois:name="suspend"/>
<rois:MessageProfile xsi:type="rois:CommandMessageProfileType" rois:name="resume"/>

// ===== Query Message ======
<rois:MessageProfile xsi:type="rois:QueryMessageProfileType" rois:name="component_status">
  <rois:Results rois:name="status">
    <rois:data_type_ref rois:code="urn:x-rois:def:DataType:ATR::Component_Status"/>
  </rois:Results>
</rois:MessageProfile>

// ===== Event Message ======
<rois:MessageProfile xsi:type="rois:EventMessageProfileType" rois:name="person_detected">
  <rois:Results rois:name="timestamp">
    <rois:data_type_ref rois:code="urn:x-rois:def:DataType:ATR::DateTime"/>
  </rois:Results>
</rois:MessageProfile>
```
This profile defines, in particular, a list of messages belonging to the person detection function as an example of an HRI Component. The HRI Component name is defined as ‘person_detection’ and the HRI Component ID as `urn:x-rois:def:HRIComponent:ATR::PersonDetection` in the `<gml:name>` tag and the `<gml:identifier>` tag, respectively, within the `<rois:HRIComponentProfile>` tag. The messages and parameters that can be used by the HRI Component are defined using the `<rois:MessageProfile>` tag and `<rois:ParameterProfile>` tag, respectively, within the `<rois:HRIComponentProfile>` tag. Definition of a message by the `<rois:MessageProfile>` tag and definition of a parameter by the `<rois:ParameterProfile>` tag follow the definition of the Message Profile and the Parameter Profile, respectively. Here, the person_detection HRI Component is defined as having four command messages (start, stop, pause, and resume), one query message (component_status), and one event message (person_detected) for a total of six messages. It is also defined as having one parameter (max_number) which is exchanged by ‘set_parameter()’ and ‘get_parameter()’ method.

Furthermore, when defining an HRI Component that adds original messages and parameters to those belonging to this person_detection HRI Component, the HRI Component Profile can be defined as shown by the following example.

```
<rois:HRIComponentProfile>
  <gml:identifier>urn:x-rois:def:HRIComponent:ATR::PersonMonitor</gml:identifier>
  <gml:name>person_monitor</gml:name>
  // ===== Include HRI Component Profile =====
  <rois:SubComponentProfile>urn:x-rois:def:HRIComponent:ATR::PersonDetection</rois:SubComponentProfile>
  // ===== Event Message =====
  <rois:MessageProfile xsi:type="rois:EventMessageProfileType" rois:name="person_disappeared"/>
</rois:HRIComponentProfile>
```

This HRI Component Profile defines an HRI Component called ‘person_monitor.’ This HRI Component adds to the messages of the person_detection HRI Component by also having an event message called “person_disappeared” that sends a notification advising that a person can no longer be detected. In this case, the person_detection HRI Component can be included as a sub HRI Component Profile so that the same message definitions can be omitted. A sub HRI Component Profile is included by specifying the ID of that HRI Component Profile using the `<rois:SubComponentProfile>` tag within the `<rois:HRIComponentProfile>` tag.

### A.4 HRI Engine Profile

This is an example of an HRI Engine Profile described in XML.
This HRI Engine Profile defines an HRI Engine called ‘MainHRI’ having two HRI Components: ‘person_detection’ and ‘person_identification’. The profile name is defined as ‘MainHRI’ and the HRI Engine Profile ID as “urn:x-rois:def:HRIEngine:ATR::MainHRI” in the <gml:name> tag and the <gml:identifier> tag, respectively, within the <rois:HRIEngineProfile> tag. The HRI Component Profiles in this HRI Engine are defined by specifying the ID of that HRI Component Profile by the <rois:HRIComponent> within the <rois:HRIEngineProfile> tag.

A system consisting of more than one HRI Engine can be defined in the following way.
The above example defines a system called “mainHRI” that includes two HRI Engines ‘SubHRI01’ having two HRI Components (person detection and person identification) and ‘SubHRI02’ having three HRI Components (person detection, person identification, and face detection). The HRI Engine Profile of ‘MainHRI’ includes the HRI Engine Profile of ‘HRI01’ and that of ‘HRI02’ as sub profiles by specifying the IDs of the corresponding HRI Component Profiles using the <rois:SubProfile> tag within the <rois:HRIEngineProfile> tag.
Annex B Examples of CommandUnitSequence in XML  
(informative)

B.1 CommandUnitSequence

This is an example of a CommandUnitSequence description for execute() in the command interface.

```xml
<rois:CommandUnitSequence>
  <rois:command_unit_list xsi:type="rois:CommandMessageType" rois:command_type="A"/>
  <rois:command_unit_list xsi:type="rois:CommandMessageType" rois:command_type="B"/>
  <rois:command_unit_list xsi:type="rois:ConcurrentCommandsType">
    <rois:branch_list xsi:type="rois:BranchType"/>
      /* Parallel Command Branch 1 */
    <rois:command_list xsi:type="rois:CommandMessageType" rois:command_type="C"/>
    <rois:command_list xsi:type="rois:CommandMessageType" rois:command_type="D"/>
    </rois:branch_list>
    <rois:branch_list xsi:type="rois:BranchType"/>
      /* Parallel Command Branch 2 */
    <rois:command_list xsi:type="rois:CommandMessageType" rois:command_type="E"/>
  </rois:branch_list>
</rois:command_unit_list>
<rois:command_unit_list xsi:type="rois:CommandMessageType" rois:command_type="F"/>
</rois:CommandUnitSequence>
```

CommandUnitSequence specifies a procedure for operating several command messages using a <rois:CommandUnitSequence> tag. A CommandUnitSequence is composed of a series of command unit lists and each command unit list is specified as either 'rois:CommandMessageType' or 'rois:ConcurrentCommandType.'

When the command unit list specifies a single command message, ‘xsi:type’ in the <rois:command_unit_list> is specified as ‘rois:CommandMessageType,’ while the command unit list specifies a parallel operation of several command lists, the

![Diagram](image)

Figure B.1: Structure of CommandUnitSequence example.
attribute ‘xsi:type’ is specified as ‘rois:ConcurrentCommandsType.’

ConcurrentCommands is composed of multiple Branches, whose attribute ‘xsi:type’ is specified as ‘rois:BranchType,’ and all the Branches are executed in parallel. In each Branch, several elements of ‘rois:CommandMessageType’ are listed using <rois:command_list> tag to be executed sequentially. A command unit list following the ConcurrentCommands should wait until all commands in all Branches in the ConcurrentCommands are completed.

This example specifies a procedure for operating six command messages, i.e., ‘A’ to ‘F,’ illustrated in Figure . In this procedure, the attribute ‘xsi:type’ of the first two <rois:command_unit_list> tags are specified as ‘rois:CommandMessageType,’ that is, two commands ‘A’ and ‘B’ are sequentially operated..

The next <rois:command_unit_list> is specified as ‘rois:ConcurrentCommandsType’ with the attribute ‘xsi:type,’ that is, it contains parallel operation branches in it. Two <rois:branch_list> tags, i.e., ‘Parallel Command Branch 1’ and ‘Parallel Command Branch 2,’ are operated in parallel. In the former element of <rois:branch_list>, two command messages, i.e., command message ‘C’ and ‘D,’ are specified using <rois:command_list xsi:type=”rois:CommandMessageType”> tags so that the command message C and D are operated sequentially. The latter element of <rois:branch_list> contains command message ‘E,’ that is executed independent from the former branch.

The last occurrence of <rois:command_unit_list>, that is specified as ‘rois:CommandMessageType’ with ‘xsi:type’ attribute, is executed after execution of both branches.

B.2 CommandMessage

This is an example of a CommandMessage description for the CommandUnitList.

```
<rois:command_list xsi:type="rois:CommandMessageType" rois:command_type="set_parameter" rois:command_id="" >
  <rois:component_ref rois:version="0.1" rois:codebook_ref="urn:x-rois:def:DataType:ATR::ComponentType" rois:code="speech_synthesis"/>
  <rois:arguments>
    <rois:parameter rois:name="speech_text">
      <rois:data_type_ref rois:code="urn:x-rois:def:DataType:ATR::String"/>
      <rois:value>hello</rois:value>
    </rois:parameter>
    <rois:parameter>
      <rois:parameter rois:name="volume">
        <rois:data_type_ref rois:code="urn:x-rois:def:DataType:ATR::Integer"/>
        <rois:value>10</rois:value>
      </rois:parameter>
      <rois:parameter rois:name="language">
        <rois:data_type_ref rois:code="urn:x-rois:def:DataType:ATR::Integer"/>
        <rois:value>en</rois:value>
      </rois:parameter>
    </rois:parameter>
  </rois:arguments>
</rois:command_list>
```

A command message is defined using a <rois:command_list> tag with the attribute ‘xsi:type’ of ‘rois:CommandMessageType’. This example defines a “set_parameter” message for the speech synthesis component. The
command method of the HRI Component is specified as “set_parameter” in the `<rois:command_type>` tag. The `<rois:component_ref>` within `<rois:command_list>` tag defines the reference ID of the HRI Component as “speech_synthesis”. Note that the reference ID is obtained when the Service Application bind the HRI Component. The reference ID is expressed using RoIS_Identifier. If there is a reference codebook for the reference IDs, the codebook and its version are specified in the attribute ‘rois:codebook and ‘rois:version’ in the `<rois:component_ref>` tag. Here, the codebook and the version are specified as ‘urn:x-rois:def:DataType:ATR::ComponentType’ and ‘0.1’, respectively.

The HRI Engine set a command ID of this message in the attribute ‘rois:command_id’ of the `<rois:command_list>` tag when the HRI Engine receives this message. Therefore the Service Application does not need to define any value in this tag.

Three argument parameters are specified for this command message. These arguments are defined using the `<rois:parameter>` tags within the `<rois:arguments>` tag. The name of each parameter is specified in the attribute ‘rois:name’ of the `<rois:parameter>` tag and the value is specified using `<rois:value>` tag within the `<rois:parameter>` tag. Here, the parameters ‘speech text,’ ‘volume’ and ‘language’ are specified as ‘hello,’ ‘10’ and ‘en’, respectively. Note that, data type is expressed in ISO639-1 and ‘en’ means English.
Annex C Examples of User-Defined HRI Component
(informative)

C.1 Speech Recognition (W3C-SRGS)

Table C.1: speech_recognition(W3C-SRGS)

Description: Recognize speech input. Here, we assume speech recognition algorithm which is configurable by a descriptive grammar (W3C-SRGS). Mandatory requirement for the speech recognition component is to return N-best result. For the speech recognition algorithm which can only output one candidate, returning a list filled with 1-best result is recommended. String of recognized text can contain either a word or a sentence.

### Command Method

<table>
<thead>
<tr>
<th>set_parameter</th>
<th>Specifies speech recognition parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>argument</td>
<td></td>
</tr>
<tr>
<td>languages</td>
<td>Set&lt;String&gt; [ISO639-1]</td>
</tr>
<tr>
<td>position_of_sound</td>
<td>Data [RLS]</td>
</tr>
</tbody>
</table>

### Query Method

<table>
<thead>
<tr>
<th>get_parameter</th>
<th>Obtains speech recognition parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td></td>
</tr>
<tr>
<td>languages</td>
<td>Set&lt;String&gt; [ISO639-1]</td>
</tr>
<tr>
<td>position_of_sound</td>
<td>Data [RLS]</td>
</tr>
<tr>
<td>active_rule</td>
<td>RuleReference [W3C-SRGS]</td>
</tr>
<tr>
<td>recognizable_languages</td>
<td>Set&lt;String&gt; [ISO639-1]</td>
</tr>
</tbody>
</table>

### Event Method

<table>
<thead>
<tr>
<th>speech_recognized</th>
<th>Notifies speech recognition has completed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td></td>
</tr>
<tr>
<td>timestamp</td>
<td>DateTime [W3C-DT]</td>
</tr>
<tr>
<td>timestamp_speech_start</td>
<td>DateTime [W3C-DT]</td>
</tr>
<tr>
<td>result</td>
<td>timestamp_speech_end</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>result</td>
<td>nbest</td>
</tr>
<tr>
<td>result</td>
<td>lattice</td>
</tr>
<tr>
<td>result</td>
<td>position_of_sound</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>speech_input_started</td>
<td>Notifies the recognizer has detected start of speech input.</td>
</tr>
<tr>
<td>speech_input_finished</td>
<td>Notifies the recognizer has detected end of speech input.</td>
</tr>
<tr>
<td>speech_recognition_started</td>
<td>Notifies the recognizer has started the recognition process.</td>
</tr>
<tr>
<td>speech_recognition_finished</td>
<td>Notifies the recognizer has finished the recognition process.</td>
</tr>
</tbody>
</table>

Table C.2: NBestType

<table>
<thead>
<tr>
<th>Description</th>
<th>Data type for speech recognition result in N-best format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived From:</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attributes</th>
<th>M</th>
<th>N ord</th>
<th>Tuple of recognized string, language, certainty.</th>
</tr>
</thead>
<tbody>
<tr>
<td>nbest</td>
<td>List&lt;String, String [ISO639-1], Error [RLS]&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table C.3: LatticeType

<table>
<thead>
<tr>
<th>Description</th>
<th>Data type for speech recognition result in lattice format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived From:</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attributes</th>
<th>M</th>
<th>N ord</th>
<th>Tuple of recognized string, language, id, previous id, next id, certainty.</th>
</tr>
</thead>
<tbody>
<tr>
<td>lattice</td>
<td>List&lt;String, String [ISO639-1], RS_Identifier [ISO19115], RS_Identifier [ISO19115], RS_Identifier [ISO19115], Error [RLS]&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

C.2 Person Gender Identification

Table C.4: person gender identification

<table>
<thead>
<tr>
<th>Description</th>
<th>This is a component for identifying person gender. This component notifies person gender code of the detected people when the code has been identified.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Method</td>
<td>This functionality may be effective for performing various robotic services since often the service needs to switch its content on the basis of person gender.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event Method</th>
<th>Notifies gender code of people when the gender has been identified.</th>
</tr>
</thead>
</table>
C.3 Person Age Recognition

**Table C.5: person age recognition**

Description: This is a component for recognizing person age. This component notifies person age of the detected people when the age has been recognized. There may be a range of the recognized age. Therefore the recognized age shall be described by lower age limit and upper age limit.

This functionality may be effective for performing various robotic services since the service often needs to switch its content on the basis of person age.

**Event Method**

<table>
<thead>
<tr>
<th>person_age_recognized</th>
<th>Notifies age of people when the age has been recognized.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>result</th>
<th>timestamp</th>
<th>DateTime [W3C-DT]</th>
<th>M</th>
<th>Measurement time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>person ref</td>
<td>List&lt;RoIS_Identifier&gt;</td>
<td>M</td>
<td>List of detected person IDs. Reference information related to the ID may be provided with the each ID. By refering the reference for the IDs, the Service Application can understand the relationship between the obtained IDs and the other IDs that are obtained from another component.</td>
</tr>
<tr>
<td>result</td>
<td>lower age limit</td>
<td>List&lt;Integer&gt;</td>
<td>M</td>
<td>List of upper limit of recognized age.</td>
</tr>
<tr>
<td>result</td>
<td>upper age limit</td>
<td>List&lt;Integer&gt;</td>
<td>M</td>
<td>List of lower limit of recognized age.</td>
</tr>
</tbody>
</table>
### Annex D Examples of Data Type (informative)

#### D.1 Reaction Type

**Table D.1: Example of Reaction_Type**

<table>
<thead>
<tr>
<th>Gesture ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>nod the head</td>
<td>Move the head downward and return to the original position</td>
</tr>
<tr>
<td>2</td>
<td>angle the head</td>
<td>Move the head to the side and return to the original position</td>
</tr>
<tr>
<td>3</td>
<td>shake the head</td>
<td>Move the head right and left</td>
</tr>
<tr>
<td>4</td>
<td>look right</td>
<td>Turn the head to the right hand side</td>
</tr>
<tr>
<td>5</td>
<td>look left</td>
<td>Turn the head to the left hand side</td>
</tr>
<tr>
<td>6</td>
<td>look up</td>
<td>Turn the head upward</td>
</tr>
<tr>
<td>7</td>
<td>look down</td>
<td>Turn the head downward</td>
</tr>
<tr>
<td>8</td>
<td>drop the head</td>
<td>Turn the head obliquely downward</td>
</tr>
<tr>
<td>9</td>
<td>bow the head</td>
<td>Turn the head slightly downward</td>
</tr>
<tr>
<td>10</td>
<td>shake hands</td>
<td>Shake hands by the right hand and look at the person’s face</td>
</tr>
<tr>
<td>11</td>
<td>spread hands slightly</td>
<td>Spread both hands slightly</td>
</tr>
<tr>
<td>12</td>
<td>raise hands and spread</td>
<td>Spread both forearms horizontally</td>
</tr>
<tr>
<td>13</td>
<td>spread hands</td>
<td>Spread both hands horizontally at shoulders’ height</td>
</tr>
<tr>
<td>14</td>
<td>clap hands</td>
<td>Clap hands several times</td>
</tr>
<tr>
<td>15</td>
<td>clap hands rhythmically</td>
<td>Clap hands rhythmically</td>
</tr>
<tr>
<td>16</td>
<td>point by the right hand</td>
<td>Point to a direction by the right hand, with turning the palm up and stretching the arm</td>
</tr>
<tr>
<td>17</td>
<td>point by the left hand</td>
<td>(Same as above, but using the left hand)</td>
</tr>
<tr>
<td>18</td>
<td>indicate a monitor display</td>
<td>Turn the head to a monitor display and point to the display by the right hand</td>
</tr>
<tr>
<td>19</td>
<td>raise both hands</td>
<td>Move both arms in front of the body and raise them from bottom to top</td>
</tr>
<tr>
<td>20</td>
<td>raise both hands from side</td>
<td>Raise both arms from the standing at attention pose to top</td>
</tr>
<tr>
<td>21</td>
<td>raise both hands at the shoulder height</td>
<td>Raise both hands from the frontal side to the shoulder height</td>
</tr>
<tr>
<td>22</td>
<td>raise a hand straight up (1)</td>
<td>Raise a hand straight up. Wave the hand to catch attention (depends on the implementation)</td>
</tr>
<tr>
<td>23</td>
<td>raise a hand straight up (2)</td>
<td>Raise a hand straight up</td>
</tr>
<tr>
<td>24</td>
<td>raise the right hand</td>
<td>Raise the right hand</td>
</tr>
<tr>
<td>25</td>
<td>raise the left hand</td>
<td>Raise the left hand</td>
</tr>
<tr>
<td>26</td>
<td>turn the right palm down</td>
<td>Turn the right palm down slightly with opening the right hand</td>
</tr>
<tr>
<td>27</td>
<td>turn the left palm up</td>
<td>(Same as above, but using the left hand)</td>
</tr>
<tr>
<td>28</td>
<td>wave the hand</td>
<td>Wave the hand</td>
</tr>
<tr>
<td>29</td>
<td>move the fingertip up</td>
<td>Move the thumb-side of the hand in front of the body with the fingertip up and move the hand downward slightly</td>
</tr>
<tr>
<td>30</td>
<td>cross arms</td>
<td>Cross arms, making an “X” sign</td>
</tr>
<tr>
<td>31</td>
<td>make a circle with arms</td>
<td>Make a circle with arms above the head</td>
</tr>
<tr>
<td>32</td>
<td>put both hands on the head</td>
<td>Put both hands on the head</td>
</tr>
<tr>
<td>33</td>
<td>put a hand on forehead</td>
<td>Put a hand on forehead</td>
</tr>
<tr>
<td>34</td>
<td>salute</td>
<td>Move the right hand to the temple with the arm bent and turning the palm down</td>
</tr>
<tr>
<td>35</td>
<td>put a hand to ear</td>
<td>Put a hand to the ear</td>
</tr>
<tr>
<td>36</td>
<td>put a hand to mouth</td>
<td>Put a hand to mouth, like shouting. It may use both hands (implementation dependent)</td>
</tr>
<tr>
<td>37</td>
<td>make a V sign</td>
<td>Make a “V” sign with a hand</td>
</tr>
<tr>
<td>38</td>
<td>strike the chest lightly</td>
<td>Strike the chest lightly with a hand (or a fist)</td>
</tr>
<tr>
<td>39</td>
<td>rub the stomach</td>
<td>Move the right hand right and left in front of the stomach</td>
</tr>
<tr>
<td>40</td>
<td>put a hand on the waist</td>
<td>Put a hand on the waist with bending the arm</td>
</tr>
<tr>
<td>41</td>
<td>put both hands on the waist (1)</td>
<td>Put both hands on the waist with bending arms</td>
</tr>
<tr>
<td>42</td>
<td>put both hands on the waist (2)</td>
<td>Put both hands on the waist with bending arms and turning the head slightly up</td>
</tr>
<tr>
<td>43</td>
<td>cross arms</td>
<td>Cross both arms in front of the chest</td>
</tr>
<tr>
<td>44</td>
<td>swing arms back and forth</td>
<td>Swing both arms back and forth like walking</td>
</tr>
<tr>
<td>45</td>
<td>knock</td>
<td>Move a fist back and forth like knocking</td>
</tr>
<tr>
<td>46</td>
<td>push by both hands</td>
<td>Raise both hands in front of the chest and move them ahead like pushing</td>
</tr>
<tr>
<td>47</td>
<td>indicate a height by a hand</td>
<td>Put a hand at a certain height with turning the palm down</td>
</tr>
<tr>
<td>48</td>
<td>bend an arm</td>
<td>Move a hand to the shoulder with bending the arm slowly</td>
</tr>
<tr>
<td>49</td>
<td>put an arm on a shoulder</td>
<td>Put an arm on someone’s shoulder</td>
</tr>
<tr>
<td>50</td>
<td>glance at a wristwatch</td>
<td>Glance at the left wrist</td>
</tr>
</tbody>
</table>
USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that:
(1) both the copyright notice identified above and this permission notice appear on any copies of this specification;
(2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and
(3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.
DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c) (1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™, Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™, MOF™, OMG Interface Definition Language (IDL)™, and OMG SysML™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed
using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.
OMG’s Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue (http://www.omg.org/technology/agreement.)
# Table of Contents

1 Scope .......................................................................................................................... 3

2 Conformance ............................................................................................................... 3

3 Normative References ............................................................................................... 3

4 Terms and Definitions ............................................................................................... 4

5 Symbols ...................................................................................................................... 4

6 Additional Information ............................................................................................... 5

   6.1 Changes to Adopted OMG Specifications ......................................................... 5

   6.2 Acknowledgements ............................................................................................. 5

7 Introduction .................................................................................................................. 6

   7.1 Dynamic Deployment and Configuration ......................................................... 6

       7.1.1 Supervisors ............................................................................................... 6

       7.1.2 Target Environment .................................................................................. 8

8 Platform Independent Model ...................................................................................... 9

   8.1 Overview .............................................................................................................. 9

   8.2 Component Data Model ...................................................................................... 10

       8.2.1 ComponentInstanceType ......................................................................... 10

       8.2.2 ComponentKind ....................................................................................... 11

       8.2.3 ExecutionType ......................................................................................... 11

       8.2.4 ActivityType ............................................................................................ 12
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2.3</td>
<td>Primitive Types</td>
<td>29</td>
</tr>
<tr>
<td>9.2.4</td>
<td>Mapping to IDL</td>
<td>29</td>
</tr>
<tr>
<td>9.2.5</td>
<td>DEPL</td>
<td>29</td>
</tr>
<tr>
<td>9.2.6</td>
<td>Notification Service</td>
<td>30</td>
</tr>
</tbody>
</table>

**Annex A: XML Schema and IDL** 31
Preface

Overview of this Specification

The Robotic Technology Component (RTC) Specification is an OMG standard for a component model for robotic systems. The RTC represents hardware and/or software entity that provide functionality and services for robotic systems. For deployment and configuration for component based applications, the Deployment and Configuration of Component-Based Distributed Applications (DEPL) OMG standard is available.

Generally speaking, since system structure and configuration are frequently affected by robot movement and application or scenario state, it is important to be able to represent and realize dynamic component deployment and run-time re-configuration requirements. In order to sustain and increase use of RTC and DEPL standards, it is essential to extend the DEPL standard to effectively support requirements related to dynamic behaviors.

The DDC4RTC specification defines extensions to DEPL specification to realize dynamic deployment and configuration requirements for dynamic applications based on RTC and other component models. The RTC profile defines additional component information, and the RTS Profile defines additional system information as extensions to DEPL specification. The Supervisor FSM defines a description scheme for dynamic behavior which is described as RTS Profile based state transition.

This submission provides a response to the Dynamic Deployment and Configuration for Robotic Technology Component RFP (mars/2010-06-16) and defines a solution to address the limitations identified in the afore-mentioned RFP by extending the DEPL specification with an RTC related data types, dynamic deployment and configuration, and additional services for dynamic behavior.

Intended Audience

The audience for this document is intended to be middleware developers, middleware vendors, tool vendors and application developers. This document presumes familiarity with concepts and terminology from SDO and RTC, as well as object oriented programming and component based software development. The actual data model and interface specification is provided in the Object Management Group's Interface Definition Language (IDL) and XML schema; experience with Java or C++ syntax should be sufficient to allow comprehension.

Organization of this Specification

The specification includes the following chapters:
- Scope: Specifies the scope of DDC4RTC specification covered.
- Conformance: Compliance with DDC4RTC.
- Normative References: References to other adopted specifications.
- Terms and Definitions: Formal definitions that are taken from other documents.
- Symbols: Symbols used by this specification.
OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A Specifications Catalog is available from the OMG website at:


Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM)

Platform Specific Model and Interface Specifications

- CORBAservices
- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications

All of OMG’s formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: pubs@omg.org

Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org
Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

**Helvetica/Arial - 10 pt. Bold:** OMG Interface Definition Language (OMG IDL) and syntax elements.

**Courier - 10 pt. Bold:** Programming language elements.

**Helvetica/Arial - 10 pt:** Exceptions

NOTE: Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.
Submission Specific Information

0.0 Submission Team and Copyright Waiver

The following companies submitted this specification:

- ETRI (Electronics and Telecommunications Research Institute)
- AIST (National Institute of Advanced Industrial Science and Technology, Japan)
  AIST Tsukuba Central 2,
  Tsukuba Ibaraki 305-8568 Japan
  Contact: Noriaki Ando (n-ando@aist.go.jp)

The following companies supported this specification:

- Technologic Arts Incorporated
- Japan Robot Association

Each of the entities listed above: (i) grants to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version, and (ii) grants to each member of the OMG a nonexclusive, royalty-free, paid up, worldwide license to make up to fifty (50) copies of this document for internal review purposes only and not for distribution, and (iii) has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used any OMG specification that may be based hereon or having conformed any computer software to such specification.

0.1 Mandatory Requirements

Proposals shall provide a Platform Independent Model (PIM) expressed in UML and at least one Platform Specific Model (PSM) as CORBA-specific model and XML schema for RTC Dynamic Deployment and Configuration. The models shall meet the following requirements.

Resolution: This specification provides a PIM in UML, and CORBA IDL and XML schema are provided as PSM

Platform independent deployment and configuration model

0.1.1 [6.5.1] Proposals shall specify interfaces to services for dynamic configuration and deployment of RTGs.

- storing, searching and retrieving RTC,
- storing, searching and retrieving RTC-based applications
- RTC registration

Resolution: Basic feature of RTC and RTC-based application storing, searching and retrieving are already defined in [DEPL] specification. DDC4RTC utilizes the specification. RepositoryManager which extends DEPL's RepositoryManager is defined in section 8.3.1. The search condition is given by the query which is described by the ISO/TC211 Geographic Information-filter encoding (ISO reference number: 19143).

RTC instances registration and searching features are defined as DirectoryManager in section 8.5.3.

0.1.2 [6.5.2] Proposal shall specify interfaces to initiate RTC configuration based on external and/or internal events. A capability for event filtering shall be provided.

Resolution: RTC configuration is performed by DEPL's deployment and configuration mechanisms. And event handling/filtering are defined in SupervisorFSM in the section 8.4.1.
0.1.3 [6.5.3] Proposals shall reuse or extend at least the PIM, including terms and definitions, of the deployment architecture as defined by the Deployment and Configuration of Component-based Distributed Applications Specification [DEPL].

Resolution: Proposed specification reuse or extend DEPL specification in the Component Data Model package, the Component Management Model package, the Execution Data Model package and the Execution Management Model.

Platform independent RTC information model

0.1.4 [6.5.4] Proposals shall provide a schema, the RTC Profile, describing RTC characteristics such as basic RTC information, ports information and so on, based on the RTC specification.

Resolution: In the section 8.2, DEPL Component Data Model is extended for RTC data model. Since DEPL's Port and RTC's Port has some differences, new port description data model are defined there.

0.1.5 [6.5.5] Proposals shall provide a schema, the RTC-based System Profile, describing RTC-based systems characteristics such as port connection information, configuration information, deployment conditions and so on.

Resolution: RTC-based system description is described by DEPL's system description with extended component data model for RTC.

0.1.6 [6.5.6] Proposals shall specify query services to discover and interrogate characteristics of RTCs and RTC-based systems.

Resolution: RTC instances registration and searching features are define as DirectoryManager in section 8.5.3. The search condition is given by the query which is described by the ISO/TC211 Geographic Information-filter encoding (ISO reference number : 19143).

0.1.7 [6.5.7] Proposal shall specify query services to discover characteristics and location information of deployed RTCs and RTC-based systems.

Resolution: RTC instances registration and searching features are define as DirectoryManager in section 8.5.3. The search condition is given by the query which is described by the ISO/TC211 Geographic Information-filter encoding (ISO reference number : 19143).

0.2 Discussion Items

0.2.1 [6.7.1] Proposals shall discuss the possibility of applying the proposed model to other existing fields/projects of interest that deploy components such as CCM [CCM], SDRP [SDRP], DEPL [DEPL] and other well-known component models.

This specification extends DEPL. Full integration into next revision of CCM is planned.

0.2.2 [6.7.2] Proposals shall discuss their relation to and dependency on existing communication protocols or middleware standards, such as CORBA [CORBA] or DDS [DDS].

This specification uses CORBA Notification Service.

0.2.3 [6.7.3] Proposals shall discuss efficient methods/procedures to avoid the need for extensive
information discovery activities when interacting with the environment or other robots.

This specification takes advantages of the efficiency of CORBA Notification Service. See also section 7.
1 Scope

This specification defines data models and service interfaces of deployment and configuration for RTC (OMG Robotic Technology Component Specification) based dynamic applications as an extension to DEPL (OMG Deployment and Configuration of Component-based Distributed Applications Specification) specification. It includes PIM (Platform Independent Models) and PSM (Platform Specific Models) for these models.

This specification is designed to support development of applications whose structure changes dynamically at run-time, such as robotic applications. This specification extends the existing Deployment and Configuration Specification, and utilizes the Robotic Technology Component specification as a component mode.

This specification assumes a state machine for managing when to re-deploy in the application life-cycle. Each state would be a deployment and a change in state means executing the deployment plan for that state. The developer would specify the robot's lifetime state machine, setting the deployment requirements in each state and describing the transitions.

This specification defines a management model and an information model for the dynamic deployment and configuration for Robotic Technology Components (RTC).

In particular, the specification provides:

- Ways to search for and deploy RTC into robotic systems at run-time.
- Ways to notify the relevant RTC instances of environment changes.
- Ways to search for appropriate RTC instances and dynamically configure them.

2 Conformance

The DEPL specification, which is a basis of this specification, defines several independent compliance points to enable different vendor implementations or user replacement of implementations. Suggested conformance points are RepositoryManager, TargetManager, NodeManager and ExecutionManager. DDC4RTC follows these conformance points of the DEPL.

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.


[CORP] UML Profile for CORBA 1.0, http://www.omg.org/spec/CORP/1.0/


4 Dynamic Deployment and Configuration for Robotic Technology Component (DDC4RTC) Specification, draft
4 Terms and Definitions

For the purposes of this specification, the following terms and definitions apply. The terms are defined based on the terms and definitions in DEPL, SDO and RTC specifications.

Robot application

A software application that controls a robot's behavior. Examples include a vacuum cleaning robot and a butler robot.

Super Distributed Object (SDO)

A logical representation of a hardware device or a software component that provides well-known functionality and services. One of the key characteristics in super distribution is to incorporate a massive number of objects, each of which performs its own task autonomously or cooperatively with other objects. Examples of SDOs include abstractions of devices such as mobile phones, PDAs, and home appliances, but are not limited to device abstractions. An SDO may abstract software component and act as a peer in a peer-to-peer networking system. For more details see [SDO].

Robotic Technology (RT)

Robotic Technology (RT) is a general term of the technology originating in robotics, and it can be applied not only to standalone robots but also to ubiquitous computing and other more intelligent electrical devices.

Robotic Technology Component (RTC)

A software component that supports the integration of RT systems. RTC provides rich component life cycle to enforce state coherency among components. It also supports fundamental design patterns including collaboration of fine-grained components tightly coupled in time, stimulus response with finite state machines and dynamic composition of components collaborating synchronously or asynchronously. For more details see [RTC].

Robotic Technology (RT) System

Systems based on robotic technology, in general. Systems that are comprised of RTCs is called RT System in this specification.

Dynamic Deployment and Configuration

Changing the configuration, connections, and even utilized components of a deployed component-based system at runtime to meet the changing needs of the system.

5 Symbols

There are no special symbols.
6       Additional Information

6.1   Changes to Adopted OMG Specifications

This specification only extends the OMG DEPL specification without any changes to the original specification.

6.2   Acknowledgements

The following companies submitted this specification:

- ETRI (Electronics and Telecommunications Research Institute)
- AIST (National Institute of Advanced Industrial Science and Technology, Japan)
  AIST Tsukuba Central 2,
  Tsukuba Ibaraki 305-8568 Japan
  Contact: Noriaki Ando (n-ando@aist.go.jp)

The following companies supported this specification:

- Technologic Arts Incorporated
- Japan Robot Association
7 Introduction

The Object Management Group's Robot Technology Component (RTC) specification describes a component model for robotic and intelligent systems, providing a framework in which such systems can be developed using model- and component-based engineering technologies. At its core is the LightweightRTC, the definition of a basic component of robot technology. It includes sophisticated introspection facilities, configuration facilities, and separates execution control from functional specification by way of Execution Contexts.

Use of any real system involves copying the components involved to the computing nodes where they will execute, configuring them according to the system's specification, and executing them. This process is known as deployment. It has been standardised by the OMG in its Deployment and Configuration of Component-based Distributed Applications specification (also known as DEPL). DEPL defines the various data models and execution models for deploying CORBA-based component systems. It is a very flexible specification. However, it is also static, being aimed at systems that do not change once deployed.

A robotic system is often a complex, distributed system, made up of the robot devices themselves as well as devices in the environment around the robots, such as sensors. Some of the devices involved are fixed while some can move, both physically and in terms of network topology. This dynamism, which may be triggered by both internal and external events, leads to changes in internal state and the component topology necessary to achieve the robotic system's goals. The deployed component-based system must therefore change its configuration, connections, and even deployed components at runtime to meet the changing needs of the system.

The DEPL specification is a static deployment standard. However, it can be extended to meet the needs of dynamic redeployment and re-configuration. This specification extends the OMG DEPL specification to realise functionality for dynamic system structure changes triggered by both internal and external events.

7.1 Dynamic Deployment and Configuration

Dynamic deployment and configuration (DDC) refers to the monitoring of an RT System (a system comprised of RTCs) at run time, and changing its configuration (the components in use, the connections between the components, and the configuration parameters applied to the components) in response to detected events. The dynamism is discrete, in that when an event is detected, it triggers a transition from one state to another, where each state corresponds to a fixed component topology determined in advance by the developer. As part of deploying new components to a running system, the new components may need to be retrieved from a component repository on a network.

The executing system is completely specified in advance. This includes the specification of the system's various states, each of which corresponds to a component topology, and the transitions between these states, including the events that trigger them. It is the transition from one state to another that performs DDC. Events that may trigger DDC include the passing of time, changes in the lifecycle state of RTCs such as from Active to Error, data from the RTCs themselves, and hardware changes such as the removal or addition of a sensor.

An important concept in this specification is that RT Systems may, utilising the RTC specification's composite component facility, be contained within other RT Systems. This forms a tree structure of RT Systems. This same concept is used in the DEPL specification. A contained RT System is seen as a black-box component by the containing RT System, meaning that it has no knowledge of the internal structure, or even that it has an internal structure. Such isolation is necessary to make RT System reuse possible.

7.1.1 Supervisors

The core technology responsible for managing the state machine at run time, and therefore responsible for performing DDC, is the Supervisor. There is exactly one Supervisor per RT System. When one RT System is embedded in another via composition, it naturally becomes supervised (as a complete unit) by that RT System's Supervisor (this by definition means that each Supervisor, except the top-level Supervisor, has exactly one parent Supervisor). As a result, the
Supervisors also form a tree structure. This is known as a Supervision Tree, and it is a vital concept in ensuring that every Supervisor is itself supervised. It is this tree that is responsible for localizing DDC, while allowing it to propagate up the tree when it cannot be handled locally.

As part of DDC, Supervisors are responsible for managing errors that occur in their RT System. The approach used in this specification is to first attempt local error handling, such as by restarting the offending component. If this fails, the error propagates up to the next Supervisor in the tree, which tries handling the error local to itself. If the error continues to fail to be handled, the Supervisor will declare a failure of its RT System, terminating all components under its control and, as its final act, notifying its parent Supervisor of the error. This propagation continues up the tree until the top-level Supervisor is reached. If an error cannot be handled at this level, the application as a whole fails. For example, an RT-System providing a locomotion service may contain separate RT-Systems for path planning and motor control. A failure in one does not necessarily lead to a failure in the other, and can potentially be handled locally by the supervisor managing the two RT-Systems. If, on the other hand, it cannot be handled locally, the error propagates up the tree, terminating supervisors as it goes, until one is found that can handle the error.

A Supervisor is responsible for monitoring the condition of all components under its control, including composite components that may themselves contain Supervisors. The Supervisor must also monitor other pre-determined conditions, such as events from the environment. It does this via an event service interface based on the CORBA Notification Service specification.

Internally, the Supervisor functions as a Finite State Machine. Each state specifies an RT System. It begins in a known start state. This corresponds to the initial deployment plan. Events received via the notification service and RTC monitoring trigger transitions in this FSM. Each transition is a re-deployment, affected by applying the RT System of the new state. (Transitions back to the same state result in no changes and so no re-deployment.)

Supervisors may also terminate, destroying their RT-Systems, for other reasons. If an event occurs for which the Supervisor has no suitable deployment, it will terminate with a failure. If its parent Supervisor terminates, it will terminate as well.

How the RT-System conditions are checked is not described by this specification. The Supervisor receives events over an
interface; how they are generated is implementation-dependent. A possible method is to have RTCs in the RT-System that monitor the outputs of various components, and transmit events for the Supervisor to listen to. For example, one could monitor the inputs and outputs of motor controllers in an arm and send a “stalled” event when it detects no movement corresponding to the commands, leading to the Supervisor to attempt recovery. A subsequent “stalled” event may then lead to the Supervisor giving up and terminating.

In such an event-based system, the speed at which events arrive and are responded to must be given consideration. This is particularly important for repeated error events. If a Supervisor attempts error recovery, but the error immediately recurs, it may alternate rapidly between the error and recovery states. This is known as “bouncing.” Supervisors avoid continuous bouncing by specifying limits on state transition rates. Exceeding these limits is an error that causes the Supervisor to immediately fail.

An aspect of dynamic redeployment that is particularly important in robotics is the quantity of state information that an RTC may accumulate. Redeploying to alter a system for a new environment, for example swapping out one localisation system for another, must allow for maintaining state. Supervisors are therefore able to retrieve state from RTCs and pass it on to their successors. The developer must specify successors when creating deployments. This allows live upgrading of robot software and transitioning between related RT-Systems without losing state information. RTCs that can pass information on are placed in “lineages” by the developer. There is exactly one component in the lineage in every deployment for the RT-System. The Supervisor can shift the state from one component to another as it executes different deployments.

### 7.1.2 Target Environment

The target environment extends that of the DEPL specification. A robot exists in an environment that may contain additional devices it can utilize, such as sensing devices (for example, cameras) monitoring rooms of a house. In particular, if the robot is mobile, it can move through the environment, which changes what devices are available for its use.

It is not possible to anticipate the exact sequence of events an RT System will move through during operation. The developer is therefore responsible for specifying the allowed states of the RT System and the allowed transitions between those states. Once deployed in the target environment, the DDC facilities, particularly the Supervisor, are responsible for shifting the RT System between various states as appropriate to its immediate needs. This provides the flexibility and adaptability necessary.
8 Platform Independent Model

This section presents the normative specification for the platform independent deployment and configuration model.

8.1 Overview

This specification uses UML diagrams to show classes and their relationships. All classes are part of the DDC4RTC (Dynamic Deployment and Configuration for RTC) package, which contains the Component Data Model sub-package, Component Management Model sub-package, Execution Data Model sub-package and Execution Management Model sub-package. If, in a UML diagram, a class’s attribute and operation compartments are suppressed, then this class is elaborated elsewhere. In this case, the diagram might also not show all of the class’ associations. However, if a class is shown to have only an attribute or an operation compartment, then this signifies that the not-shown compartment is empty. I.e., if a class is shown with an attribute but no operation compartment, then the class does not have any operations.

Figure 8.1: Dynamic Deployment and Configuration for RTC Model Package Structure

Each of the DDC4RTC sub-packages extend the same name sub-package from [DEPL] specification as shown in Figure 8.1. Some classes of DDC4RTC inherit, use, associate and aggregate DEPL classes. In this case, the referenced class includes the appropriate namespace information.
8.2 Component Data Model

The component data model (Figure 8.2) describes profiles of component related information necessary for deployment and assembly of RT Systems. It is based on the DEPL specification.

![Component Data Model Diagram]

Figure 8.2: Component Data Model

8.2.1 ComponentInstanceType

Description

The ComponentInstanceType enumeration defines the component instance type of the RTCs.

Attributes

- STATIC: ComponentInstanceType
  - An RTC with this ComponentInstanceType is a component that is instantiated statically. Static instantiation may be used for reasons such as association with specific hardware. The component shall continue to exist throughout the life cycle of the system.

- UNIQUE: ComponentInstanceType
  - An RTC with this ComponentInstanceType is unique. It is not possible to exchange internal data with other components of the same type at runtime. Generating an instance of this type of RTC may be done dynamically.

- COMMUTATIVE: ComponentInstanceType
  - An RTC with this ComponentInstanceType is commutative. It is
possible to exchange its internal data and state with other components of the same type. Generating an instance of this type of RTC may be done dynamically.

8.2.2 ComponentKind

Description

The ComponentKind enumeration defines the combination of execution semantics defined in the RTC specification's execution semantics. This kind, defined in the RTC specification in “5.3 Execution Semantics,” is a combination of DataFlow, FiniteStateMachine and MultiMode kinds.

Attributes

DataFlow : ComponentKind
An RTC with this type is a dataflow-based component. It is analogous to the DataFlow type in the ComponentAction ExecutionSemantics of [RTC].

FiniteStateMachine : ComponentKind
An RTC with this type is a FiniteStateMachine-based component. It is analogous to the FiniteStateMachine in the ComponentAction ExecutionSemantics of [RTC].

DataFlowFiniteStateMachine : ComponentKind
An RTC with this composite type is a FiniteStateMachineDataflow component. It is analogous to a combination of the FiniteStateMachine and DataFlow types in the ComponentAction ExecutionSemantics [RTC].

FiniteStateMachineMultiMode : ComponentKind
An RTC with this composite type is a MultiModeFiniteStateMachine component. This is analogous to a combination of the MultiMode and FiniteStateMachine types in the ComponentAction ExecutionSemantics [RTC].

DataFlowMultiMode : ComponentKind
An RTC with this composite type is a MultiModeDataflow component. This is analogous to a combination of the MultiMode and DataFlow types in the ComponentAction ExecutionSemantics [RTC].

DataFlowFiniteStateMachineMultiMode : ComponentKind
An RTC with this composite type is a combination of the MultiModeFiniteStateMachine Dataflow type in the ComponentAction ExecutionSemantics [RTC].

8.2.3 ExecutionType

Description

A type of RTC activity.

Attributes

PERIODIC: ExecutionType
The component execution type is PERIODIC with bounded execution time.

12 Dynamic Deployment and Configuration for Robotic Technology Component (DDC4RTC) Specification, draft
8.2.4 ActivityType

Description

A type of RTC activity.

Attributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERIODIC: ExecutionType</td>
<td>The component execution type is PERIODIC.</td>
</tr>
<tr>
<td>SPORADIC: ExecutionType</td>
<td>The component execution type is SPORADIC.</td>
</tr>
<tr>
<td>EVENT_DRIVEN: ExecutionType</td>
<td>The component execution type is EVENT_DRIVEN.</td>
</tr>
<tr>
<td>UNKNOWN: ExecutionType</td>
<td>The component execution type is UNKNOWN.</td>
</tr>
</tbody>
</table>

8.2.5 RTComponentActionDescription
The RTComponentActionDescription holds the information about the callback target of ComponentAction which RTC implements.

**Attributes**

- **on_initialize**: ComponentAction [0..1]  
  Callback for the on_initialize state.

- **on_finalize**: ComponentAction [0..1]  
  Callback for the on_finalize state.

- **on_startup**: ComponentAction [0..1]  
  Callback for the on_startup state.
on_shutdown: ComponentAction [0..1]  Callback for the on_shutdown state.
on_activated: ComponentAction [0..1]  Callback for the on_activated state.
on_deactivated: ComponentAction [0..1] Callback for the on_deactivated state.
on_reset: ComponentAction [0..1]  Callback for the on_reset state.
on_execute: ComponentAction [0..1]  Callback for the on_execute state.
on_state_update: ComponentAction [0..1] Callback for the on_state_update state.
on_rate_changed: ComponentAction [0..1] Callback for the on_rate_changed state.
on_action: ComponentAction [0..1]  Callback for the on_action state.
on_mode_changed: ComponentAction [0..1] Callback for the on_mode_changed state.
on_startup: ComponentAction [0..1]  Callback for the on_startup state.
on_shutdown: ComponentAction [0..1]  Callback for the on_shutdown state.

Semantics

The need to record callbacks will vary depending on the ComponentAction type. At a minimum, the on_initialize, on_finalize, on_startup, on_shutdown, on_activated, on_deactivated, and on_reset states are necessary. For DataFlow components, on_execute, on_state_update, and on_rate_changed are also necessary. For FSM components, on_action is required. MultiMode components must have on_mode_changed. RTCs with a composite type must have the necessary callbacks for each included type.

8.2.6 ComponentAction

Description

This class describes component’s action call back implementation status.

Attributes

implemented: Boolean  A flag indicating if the action is implemented.
boundedExecution: Boolean  A flag indicating if the RTC execution time is bounded.
maxExecutionTime: double [0..1]  The estimated maximum execution time of the ComponentAction of the RTC.
minExecutionTime: double [0..1]  The estimated minimum execution time of the ComponentAction of the RTC.
meanExecutionTime: double [0..1]  The estimated mean execution time of the ComponentAction of the RTC.
8.2.7 RTCImplementationDescription

Description

This class inherits ComponentImplementationDescription in DEPL specification and has additional profile information of an RTC.

Attributes

- `componentInstanceType: ComponentInstanceType` The type of RTC instance.
- `componentKind: ComponentKind` The kind of the RTC. This kind is defined in the RTC specification in 5.3 Execution Semantics, and is combination of DataFlow, FiniteStateMachine and MultiMode.
- `category: String` The category of this RTC.
- `executionType: ExecutionKind` The execution type of this RTC. This is defined in the RTC specification in section 5.2.2.7 ExecutionKind.
- `maxInstances: int` Maximum number of instances of this RTC.
- `vendor: String` The vendor name of this RTC.
- `creationDate: DateTime` The creation date and time of this RTC.
- `updateDate: DateTime` The creation date and time of this RTC.

Associations

- `actions: ComponentActionDescription` Component’s action implementation description.

8.2.8 RTComponentPortDescription
Description

This class describes an RTC's port information, including its owned port interfaces. An RTC's port can have provided interfaces and required interfaces. The RTC's port is defined as a service of PortService interface which manages connections between ports and its owned provided/required interfaces.

Based on the Port of RTC specification, the following values are statically set:
- specificType=RTC::PortService
- supportedType=RTC::PortService
- provider=true
- exclusiveUser=false
- maxConnection would be one
- exclusiveUser is not used in DDC4RTC

Figure 8.4: RTComponentPortDescription

Port and Component in DEPL

Port and Component in RTC

Port itself is a service (RTC::PortService)

Figure 8.5: Difference between Port in DEPL component and Port in RTC

Dynamic Deployment and Configuration for Robotic Technology Component (DDC4RTC) Specification, draft 17
In the DEPL specification, a port is a kind of interface attached dependently to a component. On the other hand, RTC’s port is based on the port owned by EncapsulatedClassifier defined in [UML2S] (for details see [RTC] section 5.2.) The Port is a service interface itself, and it is also place holder for provided and required interfaces. Introducing RTCComponentPortDescription, this specification supports RTC’s port description in the component description.

**Attributes**
No attributes.

**Associations**

<table>
<thead>
<tr>
<th>association</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serviceport: ServicePortDescription</td>
<td>Descriptions of service-oriented ports that are owned by the target component.</td>
</tr>
<tr>
<td>dataport: DataPortDescription</td>
<td>Descriptions of data-centric ports that are owned by the target component.</td>
</tr>
</tbody>
</table>

**Semantics**

RTCPortDescription inherits DEPL::ComponentPortDescription, and some of members are fixed for RTC’s PortService interface. The specificType and supportedType are fixed to RTC::PortService, and since it always provides a service, the provider member is always true.

### 8.2.9 RTCPortInterfaceDescription

**Figure 8.6: RTCPortInterfaceDescription**

**Description**

An RTC port has zero or more interfaces. These interfaces can be a provided interfaces which provide service functions in the component, and a required interfaces that require a provided interface in the other component.

**Attributes**

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: String</td>
<td>Human readable interface name.</td>
</tr>
</tbody>
</table>
### PortInterfacePolarity

Direction of the interface. This value can be PROVIDED or REQUIRED.

### instanceType: InstanceType

The type of instance of the interface. This value can be INDIVIDUAL and SHARED. In case of INDIVIDUAL type, instances are created for each connection, and a required interface uses independent service instance. In case of SHARED type, only one instance is shared among required interfaces.

### instanceName: String

The name of interface instance. instanceName should have unique name in a Port.

### typeDescription: String

Human readable description of the interface.

### specificType: String

The most specific type name of the interface.

### SupportedType: String [0..*]

A list of supported interface types.

#### Associations

No associations.

**Semantics**

Interfaces belong to RTC Ports (RTCPortDescription) and its connections are managed by RTC Ports.

#### 8.2.10 PortInterfaceInstanceType

**Description**

PortInterfaceInstanceType describes the instance type of an interface in a port. INDIVIDUAL means that instances of the port interface are created for individual connections, and SHARED means that a specific instance of the port is shared by multiple connections.

**Attributes**

- **INDIVIDUAL: PortInterfaceInstanceType**
  
  Instances of the port interface are created for each individual connection.

- **SHARED: PortInterfaceInstanceType**
  
  A specific instance of the port is shared by all connections.

**Associations**

No associations.

#### 8.2.11 PortInterfacePair
Description

An RTC port has zero or more interfaces. These interfaces can be a provided interfaces which provide service functions in the component, and a required interfaces that require a provided interface in the other component.

Attributes

- **provided**: String
  
  Human readable interface name.

- **required**: String
  
  Direction of the interface. This value can be PROVIDED or REQUIRED.

Associations

No associations.

Semantics

Interfaces belong to RTC Ports (RTCPortDescription) and its connections are managed by RTC Ports.

8.2.12 RTCSubcomponentPortEndPoint

Description

This class extend SubComponentPortEndPoint in DEPL specification.

Associations

- **interfacePairs**: PortInterfacePair[0..*]  
  
  A list of PortInterfacePairs.
Semantics

This class also replaces ComponentExternalPortEndpoint and ExternalReferenceEndpoint in the DEPL Common package.

8.3 Component Management Model

8.3.1 Repository Manager

![Repository Manager Class Diagram]

**Figure 8.8: Repository Manager**

**Description**

The RepositoryManager provides the interfaces for storing, searching, and retrieving RTCs, and the data model for the component profile description. The RepositoryManager also provides the interfaces for storing, searching, and retrieving RTC-based systems and the data model for the RTC-based system profile description.

**Operations**

```java
search(query : String) : PackageConfiguration[]
```

This function searches a set of packages which meet the given condition and returns a sequence of PackageDescription entities [DEPL]. The condition is given by the query which is described by the ISO/TC211 Geographic Information-filter encoding (ISO reference number: 19143).

8.4 Execution Data Model

8.4.1 SupervisorFSM

Dynamic Deployment and Configuration for Robotic Technology Component (DDC4RTC) Specification, draft 21
SupervisorFSM is an application supervisor that manages RTC-based systems according to predefined state machines. SupervisorFSM model is based on UML State Machine (Behavior State Machine). UML 2.4 State Machine package—and those packages on which they depend, directly or indirectly—are considered normative. A non-normative simplified version of that model is shown in Figure 8.9.

![Simplified UML Behavior State Machine Diagram](image)

Figure 8.9: Simplified depiction of Behavior State Machines from [UML]

**State Machine**

From [UML]:

A state machine owns one or more regions, which in turn own vertices and transitions. The behaviored classifier context owning a state machine defines which signal and call triggers are defined for the state machine, and which attributes and operations are available in activities of the state machine. Signal triggers and call triggers for the state machine are defined according to the receptions and operations of this classifier.

As a kind of behavior, a state machine may have an associated behavioral feature (specification) and be the method of this behavioral feature. In this case the state machine specifies the behavior of this behavioral feature. The parameters of the state machine in this case match the parameters of the behavioral feature and provide the means for accessing (within the state machine) the behavioral feature parameters.

A state machine without a context classifier may use triggers that are independent of receptions or operations of a classifier, i.e., either just signal triggers or call triggers based upon operation template parameters of the (parameterized) statemachine.

**State**

From [UML]: State in Behavioral State machines

A state models a situation during which some (usually implicit) invariant condition holds. The invariant may represent a static situation such as an object waiting for some external event to occur. However, it can also model dynamic conditions such as the process of performing some behavior (i.e., the model element under consideration enters the state...
8.4.2 SupervisorFSMDescription

Figure 8.10: SupervisorFSMDescription

Generalization: “BehaviorStateMachine::StateMachine”

Description

The SupervisorFSMDescription defines behavior of an RTC-based system application based on the finite state machine.

Associations

state: FSMState[0..*] A list of FSMState that associated to the state machine, SupervisorFSMDescription.

8.4.3 FSMState
Generalization: “BehaviorStateMachines::State”

Description

The State defines a state in which the SupervisorFSM may be. It corresponds to a set of one or more deployments that must be executed on entry into the state.

Attributes

- **rtsystems**: ComponentAssemblyDescription [0..1] The set of non-overlapping deployments to execute on entry into this state.
- **rate**: Float The maximum rate to enter this state. For example, if the SupervisorFSM is bouncing between two states, such as an error state and its recovery state, this rate value determines how fast it can bounce before the SupervisorFSM declares a non-recoverable error and terminates.

Semantics

The SupervisorFSM transitions between states, executing each state’s deployments as it enters. It may be limited in the rate at which it can repeatedly enter a state, in order to prevent bouncing between an error condition and recovery failure.

8.5 Execution Management Model

The Event Management Model of DDC4RTC provides certain functionality such as notifying environmental changes to RTC based applications or filtering such events based on previously registered condition. The model uses the OMG Notification Service Specification. The ApplicationSupervisor of DDC4RTC inherits the StructuredPushConsumer defined in the Notification Service Specification.

8.5.1 ApplicationSupervisor
An applicationSupervisor is a unique entity within an RT System (excluding composed RT Systems, which have their own ApplicationSupervisors). It is responsible for managing the lifetimes of the RTCs that make up the RT System. The ApplicationSupervisor inherits from the DEPL ApplicationManager, for component management interfaces, and the NOT StructuredPushConsumer, for event reception.

The ApplicationSupervisor maintains several collections of objects:

- A collection of the RTCs participating in the RT System at the present time.
- A store of RTCProfiles describing the RTCs that may potentially participate in the RT System, including those that are currently participating.
- A store of system description deployments and configuration that may be executed by the SupervisorFSM.

The contents of these collections is specified by the RT System packager in the RT System's specification (see the Dynamic Deployment Data Model). They are created during the first stage of deployment.

**Operations**

- `init(parent: ApplicationSupervisor, fsmdescription: SupervisorFSMDescription) : void`
  - Initialize ApplicationSupervisor with parent ApplicationSupervisor and SupervisorFSMDescription.
- `shutdown() : void`
  - Shut down the ApplicationSupervisor. This includes shutting down the RTCs it is supervising.

**Generalization: “NC::StructuredPushConsumer”**

**Semantics**

The behaviour of the ApplicationSupervisor is that of a Finite State Machine. Each state in the FSM constitutes a set of one or more non-overlapping deployments, where each deployment is specified by an ComponentAssemblyDescription, referencing RTCImplementationDescription stored by the SupervisorFSMDescription.

Dynamic Deployment and Configuration for Robotic Technology Component (DDC4RTC) Specification, draft 25
Upon creation, the ApplicationSupervisor will be empty. It must have a SupervisorProfile loaded into it using the loadSpecification operation. It can then be started by calling the start operation. On startup, the ApplicationSupervisor enters the initial state specified in the SupervisorProfile, executing the deployment plan(s) it specifies. Deployment follows the procedure laid out in the DEPL specification, with the following additional constraints: Components present in the new deployment and already executing in the prior deployment must not be interrupted; they must continue to run as normal. However, they should not receive execution time during the period between beginning deployment and ending deployment. Components must have the opportunity to pass state data on to replacement components.

Components are started in the exact order they are specified in the ComponentAssemblyDescription. Note that a deployment may include connections between components inside the ApplicationSupervisor's application to components in other applications. Once the initial state's deployment is complete, the ApplicationSupervisor enters a waiting state. It awaits notification of an event via the event operation. Reception of an event causes it to evaluate the current state's transitions. If a transition is valid, the ApplicationSupervisor shall transition to that state, executing the new deployment it specifies. If the transition leads to a final state, the ApplicationSupervisor shall execute its shutdown procedure. This involves removing all connections to and within the RT System and shutting down the components in the reverse order to that in which they started. As a final act before shutting down itself, the ApplicationSupervisor will notify its parent (if any) that it has shut down and the reason for shutting down. This allows the parent ApplicationSupervisor to take an appropriate action, such as restarting the ApplicationSupervisor in case of an error, replacing the shut down RT system with an alternative RT System, or propagating the shutdown reason further up the Supervision Tree (particularly in the case of an error). Notification to the parent is performed via the parent's event operation.

The ApplicationSupervisor is required to handle all events. If an event not specified in the SupervisorFiniteStateMachineDescription is received, the ApplicationSupervisor terminate with an error. For the set of pre-defined events shown below, it should respond as described. For all other events, it should respond as described by its SupervisorFiniteStateMachineDescription.

![Figure 8.13: An example behavior and communication between an ApplicationSupervisor and its RTCs.](image-url)
• **CHILD_SHUTDOWN_ERROR**: A child of the ApplicationSupervisor has shut down after an error. The ApplicationSupervisor must respond by restarting the child component according to its SupervisorFiniteStateMachineDescription. This may optionally include shutting down all children started after the failed component (according to the startup order), or all children in the RT System. Which option is used is defined in the ComponentAssemblyDescription. Any other children shut down must be restarted with the failed child in the appropriate order as defined in the ComponentAssemblyDescription. When other children are shut down, they must be informed that it is due to a failed component.

• **CHILD_SHUTDOWN_OK**: A child of the ApplicationSupervisor has shut down after completing execution. Response, if any, is determined by the SupervisorFiniteStateMachineDescription.

The ApplicationSupervisor must track the number of times an error occurs. If the error occurs at a rate greater than the maximum defined in the SupervisorFiniteStateMachineDescription, the ApplicationSupervisor must terminate with an error, including terminating all of its children. For example, an error repeatedly occurring at 5Hz in an ApplicationSupervisor with a configured maximum of 1Hz will trigger termination.

### 8.5.2 Relation to the DEPL ApplicationManager

The ApplicationSupervisor inherits from the ApplicationManager. It reuses the interface for starting and destroying applications, but with more specific semantics (described above).

### 8.5.3 DirectoryManager

![DirectoryManager class diagram]

**Description**

The DirectoryManager provides the interfaces for RTC instance discovery and the data model which describes the RTC instance.

**Operations**

- `register(desc: RTCInstanceDeploymentDescription): void`
  
  This function registers the information of an RTC instance to the directory under the given information of the RTC instance. It throws `ALREADY_REGISTERED` when the RTC instance is already registered, `INVALID_ARGUMENT` when the given information of the RTC instance is not correct, and `UNKNOWN_ERROR` when there is some error occurred.

- `unregister(ref : String) : void`
  
  This function deletes the information of an RTC instance from the directory. It throws `NOT_REGISTERED` when the RTC instance is not registered, `INVALID_ARGUMENT` when the reference is not correct, and `UNKNOWN_ERROR` when an error occurs.
This function searches a set of RTC instances which meet the given condition and returns a sequence of RTCInstanceDeploymentDescription. The condition is given by the query which is described by the ISO/TC211 Graphic Information-filter encoding (ISO reference number: 19143). It throws INVALID_ARGUMENT when the query is not correct, and UNKNOWN_ERROR when there is some error occurred.

**Associations**

managedInfos:
RTCInstanceDeploymentDescription[0..*]

The DirectoryManager manages a set of RTCInstanceDeploymentDescription.

**Semantics**

No semantics.

---

9 Platform Specific Models
In order to maximize interoperability, this document describes one PSM that should be considered normative in section 9.2. The PSM draws on a common set of IDL definitions.

## 9.1 UML-to-IDL Transformation

The PSM requires IDL definitions for the interfaces, data types, and other model elements from the PIM. They also require IDL representations of the model elements from [UML] on which the PIM depends: RepositoryManager, TargetManager, NodeManager, ExecutionManager, ApplicationSupervisor, DirectoryManager. Representing all of the UML in IDL is beyond the scope of this specification. This specification takes a more parsimonious approach.

- IDL definitions for the elements from this specification are provided explicitly in Section 9.2.
- Mapping rules from a subset of UML to IDL are provided in this section. Only those parts of UML that are necessary to describe the PIM are described here. Mappings of all other UML constructs are implementation-defined.

### 9.1.1 Basic Types and Literals

The standard UML String and Boolean types are also used by this specification.

- **Boolean:** boolean
- **String:** string
- **int:** long

The literal specifications defined in the PIM—as well as those referenced from [UML]—shall be represented as IDL literal values.

### 9.1.2 Classes and Interfaces

UML classes and interfaces shall be represented as IDL interfaces of the same name.

- Each operation or attribute on the UML classifier shall be represented by a corresponding operation or attribute in IDL.
- The general classifiers of UML classes and interfaces shall be represented as inheritance between the corresponding IDL interfaces.

### 9.1.3 Enumerations

Enumerations in the PIM shall be represented as IDL enumerations of the same name. Each attribute shall correspond to a constant within that enumeration.

### 9.1.4 Packages

A UML package described in the section 8.1 shall be represented by an IDL module of the same name.

## 9.2 CORBA PSM

In this PSM, DDC4RTC is mapped to CORBA interfaces extending the relevant IDL interfaces described in Section 9.1.

### 9.2.1 Generic Transformation Rules

Dynamic Deployment and Configuration for Robotic Technology Component (DDC4RTC) Specification, draft 29
The mapping to IDL is accomplished using the rules set forth in the UML Profile for CORBA. To enable the usage of an index, the composition of the target element in its container is qualified with the “ordered” constraint. Wherever the multiplicity of an attribute, parameter or return value is not exactly one (but 0..1, 1..* or *), a new class is introduced to represent a sequence of the type of the attribute, parameter or return value. The sequence class has the «CORBASequence» stereotype, and its name is the English plural of the name of the type. The sequence class has a 128 composition association with the element class that is navigable from the sequence to the element. The composition is qualified with the index of the sequence. The attribute, parameter or return value is then replaced with an attribute, parameter or return value, respectively, with the same name as before, but with the type being the newly introduced sequence class and the exactly one (1..1) multiplicity. A similar rule is applied to all navigable association or composition ends whose multiplicity is not exactly one (but 0..1, 1..* or *): a new class is introduced to represent a sequence of the class at the navigable end; this sequence class is defined as describe above. The original association or composition end is then replaced with a navigable association or composition end, with the same role name as before, at the new sequence class, with a multiplicity of exactly one (1..1). According to the rules in the UML Profile for CORBA, these associations and compositions will then map to a structure member in IDL, its type being a named sequence of the referenced type. Exempted from the two rules above are attributes, parameters, return values or navigable association or composition ends where the type is String, unsigned long or Endpoint. Instead of defining new sequence types, the existing types in the CORBA package are being used; see below. Note that in combination, these rules map non-composite associations between classes with a common owner and a multiplicity other than 1 to sequence of “unsigned long” type. Another exception from the rule above are attributes of type String with the 0..1 (zero or one) multiplicity. In this case, the multiplicity is updated to 1..1 (exactly one). If the value is missing in an XML representation of the model, the empty string is used as default value.

### 9.2.2 Sequence of String

A type representing a sequence of strings already exists in the CORBA package and can be re-used. Wherever the String type is used with a multiplicity other than exactly one, it is mapped to the StringSeq class from the CORBA package as shown above. It then maps to the CORBA::StringSeq type in IDL (from the orb.idl file).

### 9.2.3 Primitive Types

The UML data types String, Integer and Boolean are mapped to the classes string, long and boolean in the CORBAPROfile package, respectively. They will then map to the string, long and boolean types in IDL, respectively.

### 9.2.4 Mapping to IDL

After applying the transformations defined in this section, IDL is generated by applying the rules set forth in the UML Profile for CORBA specification [UPC].

### 9.2.5 DEPL

The ComponentDataModel, ComponentManagementModel, ExecutionDataModel, ExecutionManagementModel relies on the DEPL specification. Implementations that support that packages shall use that specification’s CORBA PSM.

### 9.2.6 Notification Service

The StructuredPushConsumer relies on the Notification Service specification [NOT]. Implementations that support that packages shall use that specification’s CORBA PSM.
Annex A: XML Schema and IDL

(normative)

A.0 CORBA IDL

#include <DEPL.idl>
#include <COS/NotificationService.idl>

#pragma prefix "omg.org"

#define PackageConfiguration string
#define RTCInstanceDeploymentDescription string

module DDC4RTC
{
    typedef sequence<PackageConfiguration> PackageConfigurationList;

    interface RepositoryManager
        : DEPL::RepositoryManager
        {
            PackageConfigurationList search(in string query);
        }

    interface ApplicationSupervisor
        : DEPL::ApplicationManager, NotificationService::StructuredPushConsumer
        {
            void init(in ApplicationSupervisor parent,
                        in SupervisorFSMDescription fsmdescription);
            void shutdown();
        }

typedef sequence<RTCInstanceDeploymentDescription> RTCInstanceDeploymentDescriptionList;

    interface DirectoryManager
        {
            void register(in RTCInstanceDeploymentDescription desc);
            void unregister(in string ref);
            RTCInstanceDeploymentDescriptionList search(in string query);
        }
};

A.1 Component Data Model XML Schema

<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:Deployment="http://www.omg.org/Deployment"
            xmlns:DEPL="D:\00\OMG\DEPL\Spec_Defined_Deployment.xsd"
            xmlns:PortInterfacePolarity="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http://www.omg.org/Deployment"
            xmlns:SimpleType="http://www.omg.org/Deployment"
            xmlns:Restriction="http://www.omg.org/Deployment"
            xmlns:Include="http: Dynamic Deployment and Configuration for Robotic Technology Component (DDC4RTC) Specification, draft
<xs:element name="on_finalize" type="ComponentAction"
  minOccurs="0" maxOccurs="1"/>
<xs:element name="on_mode_changed"
  type="ComponentAction" minOccurs="0" maxOccurs="1"/>
<xs:element name="on_startup" type="ComponentAction"
  minOccurs="0" maxOccurs="1"/>
<xs:element name="on_shutdown"
  type="ComponentAction" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
<xs:element name="RTCComponentPortDescription"
  type="RTCComponentPortDescription"/>
<xs:complexType name="RTCComponentPortDescription">
<xs:complexContent>
<xs:extension base="ComponentPortDescription">
<xs:sequence>
<xs:element name="maxConnection" type="xs:int"
  minOccurs="1" maxOccurs="1"/>
<xs:element name="properties"
  type="Deployment:Property" minOccurs="1" maxOccurs="1"/>
<xs:element name="interfaces"
  type="RTCPortInterfaceDescription" minOccurs="0"
  maxOccurs="unbounded"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:element name="RTCPortInterfaceDescription"
  type="RTCPortInterfaceDescription"/>
<xs:complexType name="RTCPortInterfaceDescription">
<xs:sequence>
<xs:element name="name" type="xs:string"
  minOccurs="1" maxOccurs="1"/>
<xs:element name="polarity" type="xs:string"
  minOccurs="1" maxOccurs="1"/>
<xs:element name="instanceType"
  type="PortInterfaceInstanceType" minOccurs="1" maxOccurs="1"/>
<xs:element name="instanceName" type="xs:string"
  minOccurs="1" maxOccurs="1"/>
<xs:element name="typeDescription"
  type="xs:string" minOccurs="1" maxOccurs="1"/>
<xs:element name="specificType" type="xs:string"
  minOccurs="1" maxOccurs="1"/>
<xs:element name="supportedType"
  type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:simpleType name="PortInterfaceInstanceType">
<xs:restriction base="xs:string">
<xs:enumeration value="INDIVIDUAL"/>
<xs:enumeration value="SHARED"/>
</xs:restriction>
</xs:simpleType>
<xs:element name="RTCSubcomponentPortEndPoint"
  type="RTCSubcomponentPortEndPoint"/>
<xs:complexType name="RTCSubcomponentPortEndPoint">
<xs:complexContent>
</xs:complexType>
<xs:element name="SubcomponentPortEndPoint" type="ComponentPortDescription"/>
<xs:complexType name="SubcomponentPortEndPoint">
<xs:sequence>
<xs:element name="maxConnection" type="xs:int"
  minOccurs="1" maxOccurs="1"/>
<xs:element name="properties"
  type="Deployment:Property" minOccurs="1" maxOccurs="1"/>
<xs:element name="interfaces"
  type="RTCPortInterfaceDescription" minOccurs="0"
  maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:element name="SubcomponentPortEndPoint" type="ComponentPortDescription"/>
<xs:complexType name="SubcomponentPortEndPoint">
<xs:sequence>
<xs:element name="maxConnection" type="xs:int"
  minOccurs="1" maxOccurs="1"/>
<xs:element name="properties"
  type="Deployment:Property" minOccurs="1" maxOccurs="1"/>
<xs:element name="interfaces"
  type="RTCPortInterfaceDescription" minOccurs="0"
  maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
A.2 Execution Data Model Schema

```xml
<xs:element name="FSMState" type="FSMState"/>
<xs:complexType name="FSMState">
  <xs:sequence>
    <xs:element name="id" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="rtsystem" type="Deployment:ComponentAssemblyDescription" minOccurs="1" maxOccurs="1"/>
    <xs:element name="transitions" type="Transition" minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="SuperVisorFSMDescription" type="SuperVisorFSMDescription"/>
<xs:complexType name="SuperVisorFSMDescription">
  <xs:sequence>
    <xs:element name="events" type="FSMState" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="finalState" type="FinalState" minOccurs="1" maxOccurs="1"/>
    <xs:element name="initialState" type="InitialState" minOccurs="1" maxOccurs="1"/>
    <xs:element name="Event" type="Event" minOccurs="1" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="Event" type="Event"/>
<xs:complexType name="Event">
  <xs:sequence>
    <xs:element name="label" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="TransitionEvent" type="TransitionEvent" minOccurs="1" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="Transition" type="Transition"/>
<xs:complexType name="Transition">
  <xs:sequence>
    <xs:element name="label" type="xs:string" minOccurs="1" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
```
<xs:element name="targetState" type="xs:string" minOccurs="1" maxOccurs="1"/>
<xs:element name="rate" type="xs:float" minOccurs="1" maxOccurs="1"/>
<xs:element name="events" type="TransitionEvent" minOccurs="1" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element name="InitialState" type="InitialState"/>
<xs:complexType name="InitialState">
<xs:complexContent>
<xs:extension base="FSMState">
<xs:sequence/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
</xs:element name="FinalState" type="FinalState"/>
<xs:complexType name="FinalState">
<xs:complexContent>
<xs:extension base="FSMState">
<xs:sequence/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
</xs:element name="TransitionEvent" type="TransitionEvent"/>
<xs:complexType name="TransitionEvent">
<xs:sequence>
<xs:element name="isRequired" type="xs:boolean" minOccurs="1" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
</xs:schema>