Date: June 2012

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/

Associated File(s)*: http://www.omg.org/spec/RoIS/20110501
http://www.omg.org/spec/RoIS/20110502

* 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/.*

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.

Copyright© 2012, Electronics and Telecommunications Research Institute
Copyright© 2012, Japan Robot Association

Copyright© 2012, Object Management Group (OMG)

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

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:

http://www.omg.org/technology/documents/spec_catalog.htm

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.

Overview

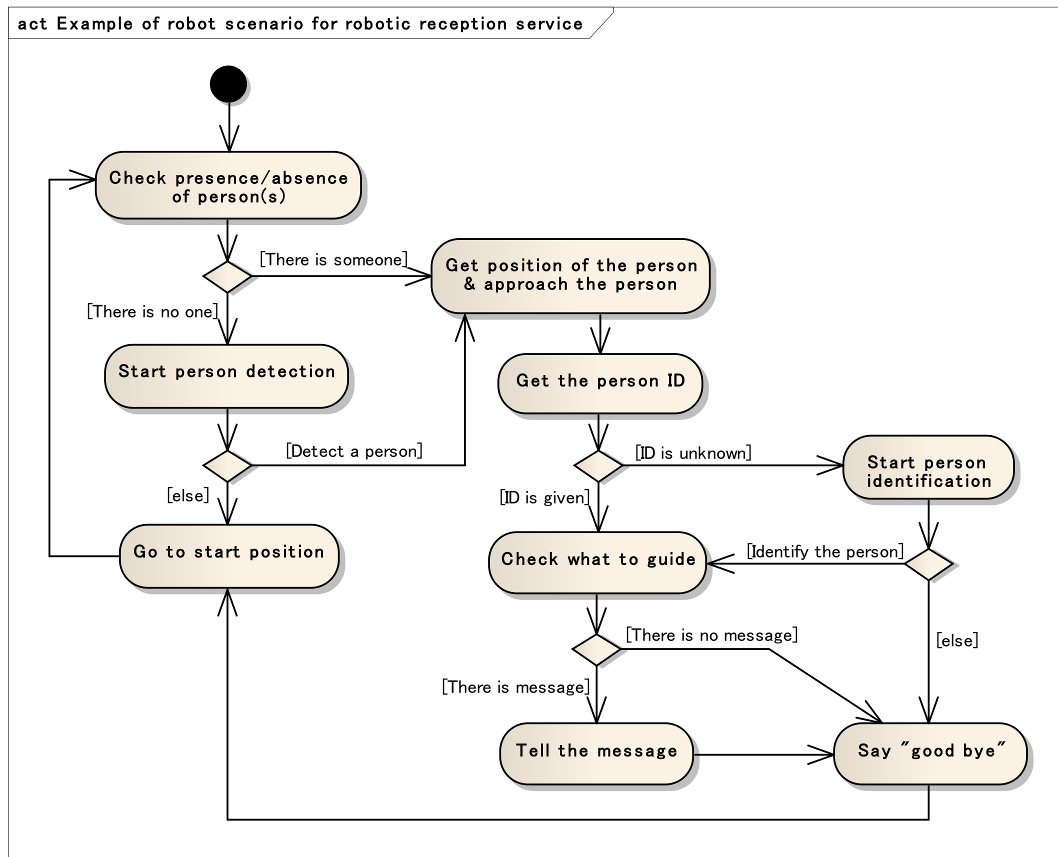

Figure 1: Example of robot scenario for robotic reception service. Events delivered from sensors, actuators or other event sources, such as an internal timer, to a service application trigger each state transition and the application controls the robot according to the scenario.

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.

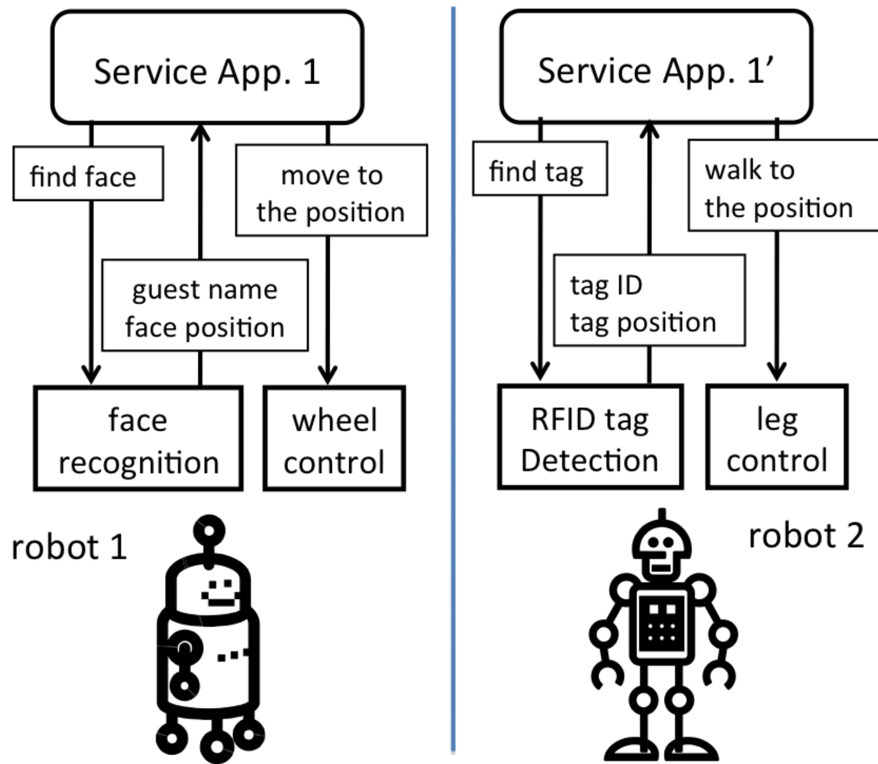

Figure 2: Conventional style of service application programming. Service application programmer must write service application programs for each robot independently because functions provided by each robot are different.

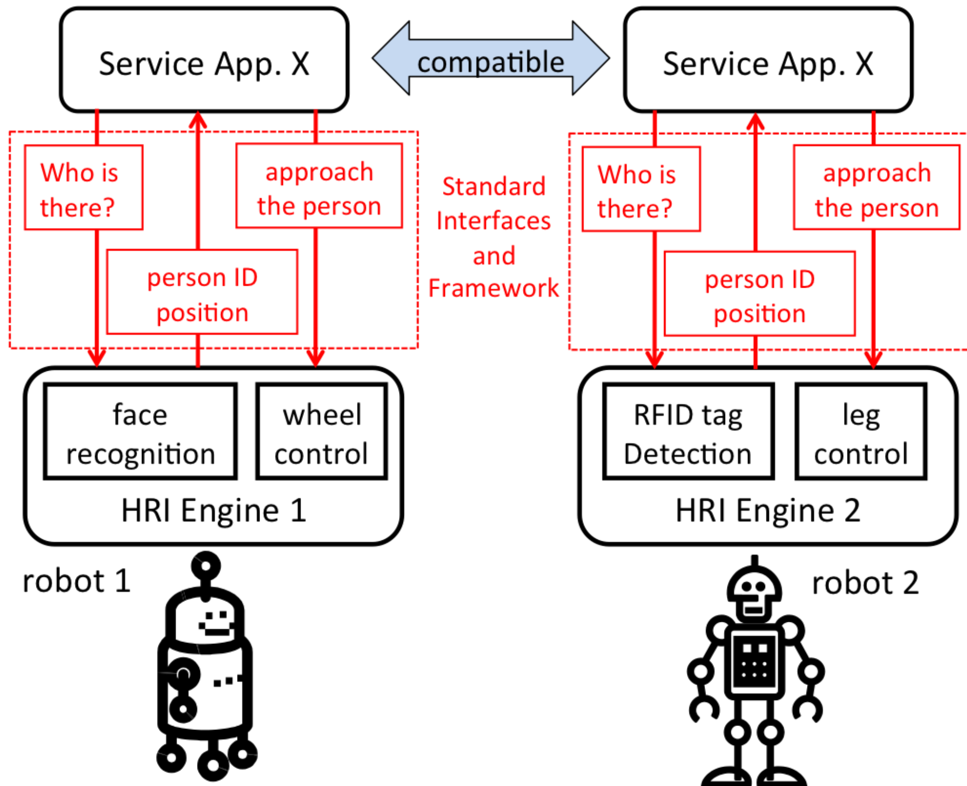

Figure 3: RoIS service application programming style. The same service application program works on different robot platforms with little modification.

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.

[CORBA] Object Management Group, Common Object Request Broker Architecture (CORBA), Version 3.1, 2008

[DDS] Object Management Group, Data Distribution Services (DDS), Version 1.2, 2007

[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

[ISO19115] International Organization for Standardization, Geographic information - Metadata, 2003

[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

[RTC] Object Management Group, Robotic Technology Component (RTC), Version 1.0, 2008

[UML] Object Management Group, OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.3, 2010

[W3C-SRGS] W3C, Speech Recognition Grammar Specification Version 1.0, 2004

[W3C-SSML] W3C, Speech Synthesis Markup Language (SSML) Version 1.0, 2004

3.2 Non-Normative References

[W3C-DT] World Wide Web Consortium, Date and Time Formats, http://www.w3.org/TR/NOTE-datetime, 1998

4 Terms and Definitions

Term	Definition
Basic HRI Component	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.
Detection	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.
HRI	Abbreviated form of “Human-Robot Interaction”
HRI Component	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.
HRI Engine	An object that manages HRI Components. It mediates Human-Robot Interaction functions of the HRI Components to Service Application(s).
Identification	A function that finds target objects and returns a list of identifiers of objects found.
Identifier (ID, in short)	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.
Localization	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.
Service Application	A software which controls HRI Components (via HRI Engine) to implement a robot scenario.
User-defined HRI Component	An HRI Component which provides an HRI function other than those any Basic HRI Components provide.

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:

Table x.x : <Class / Interface Name>

<table>
<tr><td colspan="6">Description : <description></td></tr>
<tr><td colspan="6">Derived From: <parent class></td></tr>
<tr><td colspan="6">Attributes</td></tr>
<tr><td colspan="2"><attribute name></td><td><attribute type></td><td><obligation></td><td><occurrence></td><td><description></td></tr>
<tr><td colspan="2">…</td><td>…</td><td>…</td><td>…</td><td>…</td></tr>
<tr><td colspan="6">Operations</td></tr>
<tr><td colspan="2"><operation name></td><td colspan="4"><description></td></tr>
<tr><td><direction></td><td colspan="2"><parameter name></td><td colspan="2"><parameter type></td><td><description></td></tr>
<tr><td>…</td><td colspan="2">…</td><td colspan="2">…</td><td>…</td></tr>
</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:

Table x.x : <enumeration name>

<constant name>	<description>
…	…

7.1.3 Message

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

Table x.x : <Message Name>

Description : <description>				
Derived From: <parent class>				
Attributes				
<attribute name>	<attribute type>	<obligation>	<occurrence>	<description>
…	…	…	…	…

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 x.x : <HRI Component Name>

Description : <description>				
Command Method				
<method name>	<description>			
argument	<argument parameter name>	<data type>	<obligation>	<description>
Event Method				
<method name>	<description>			
result	<result parameter name>	<data type>	<obligation>	<description>
Query Method				
<method name>	<description>			
result	<result parameter name>	<data type>	<obligation>	<description>

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.

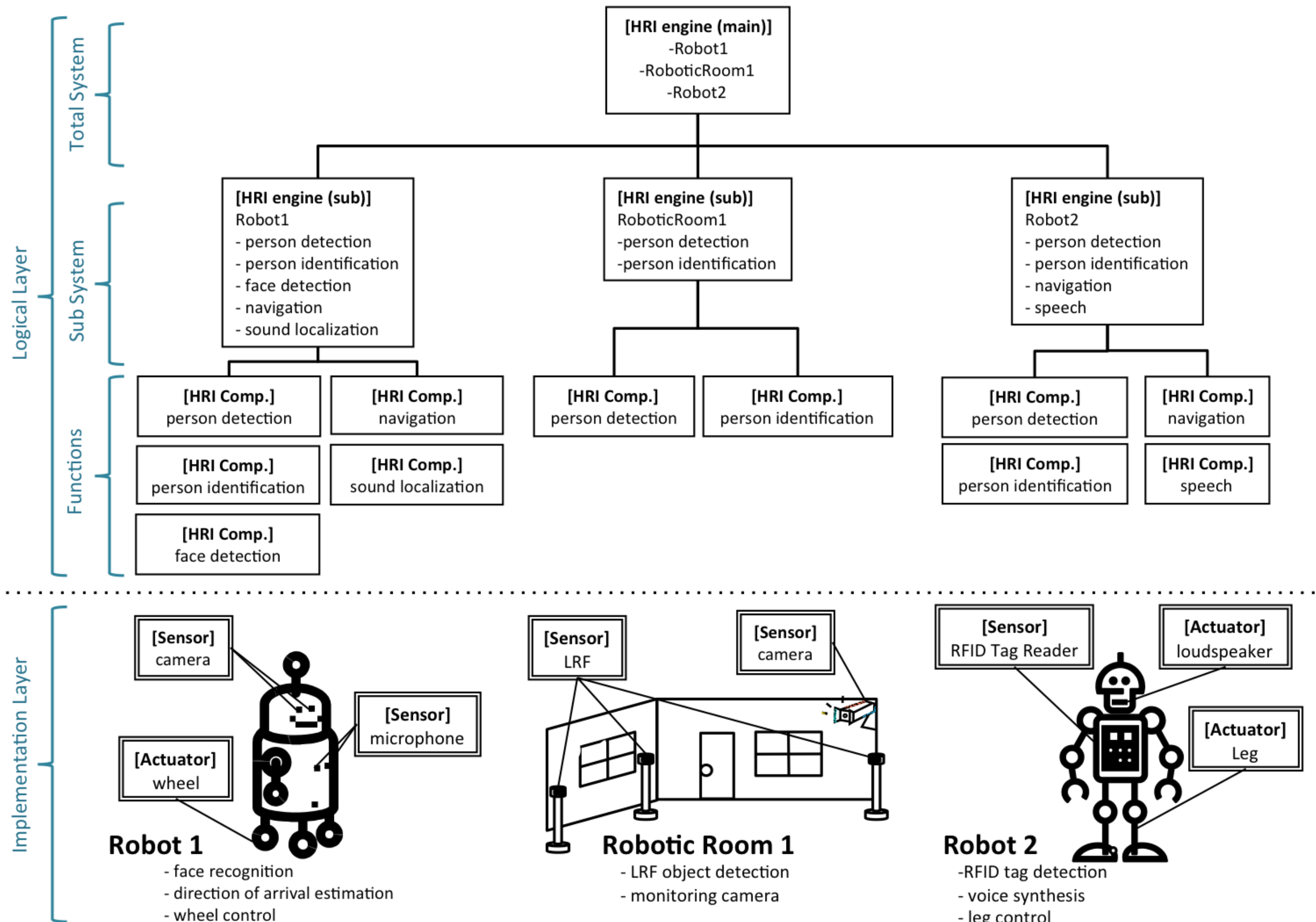

Figure 4: Example of HRI Engine and HRI Components.

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.

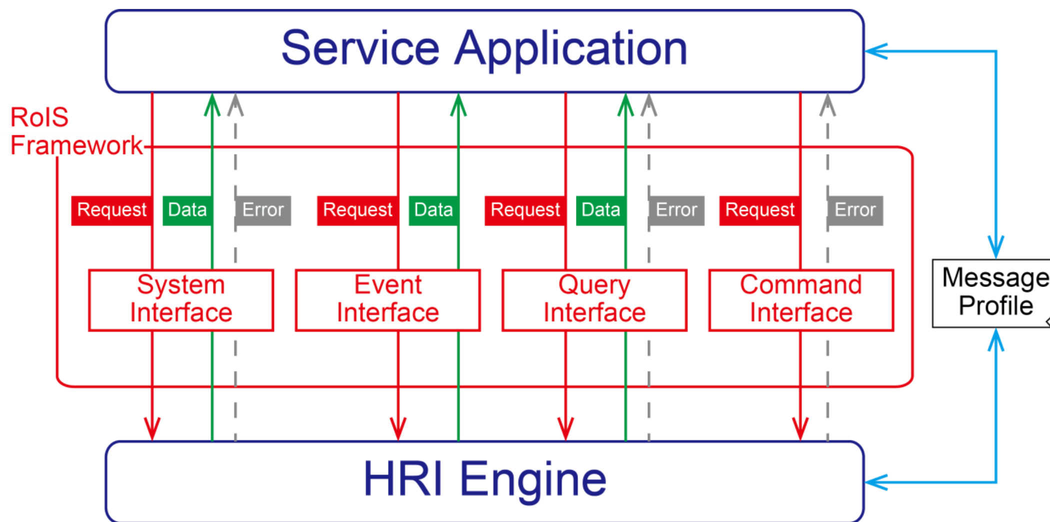

Figure 5 : Schematic diagram of RoIS Framework and its message flows. In the framework, Service Application communicates with HRI Engine by some messages through System, Event, Query and Command Interfaces.

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

OK	Successful return.
ERROR	Generic, unspecified error.
BAD_PARAMETER	Illegal parameter value.
UNSUPPORTED	Unsupported operation.
OUT_OF_RESOURCES	Service ran out of the resources needed to complete the operation.
TIMEOUT	The operation timed out.

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.

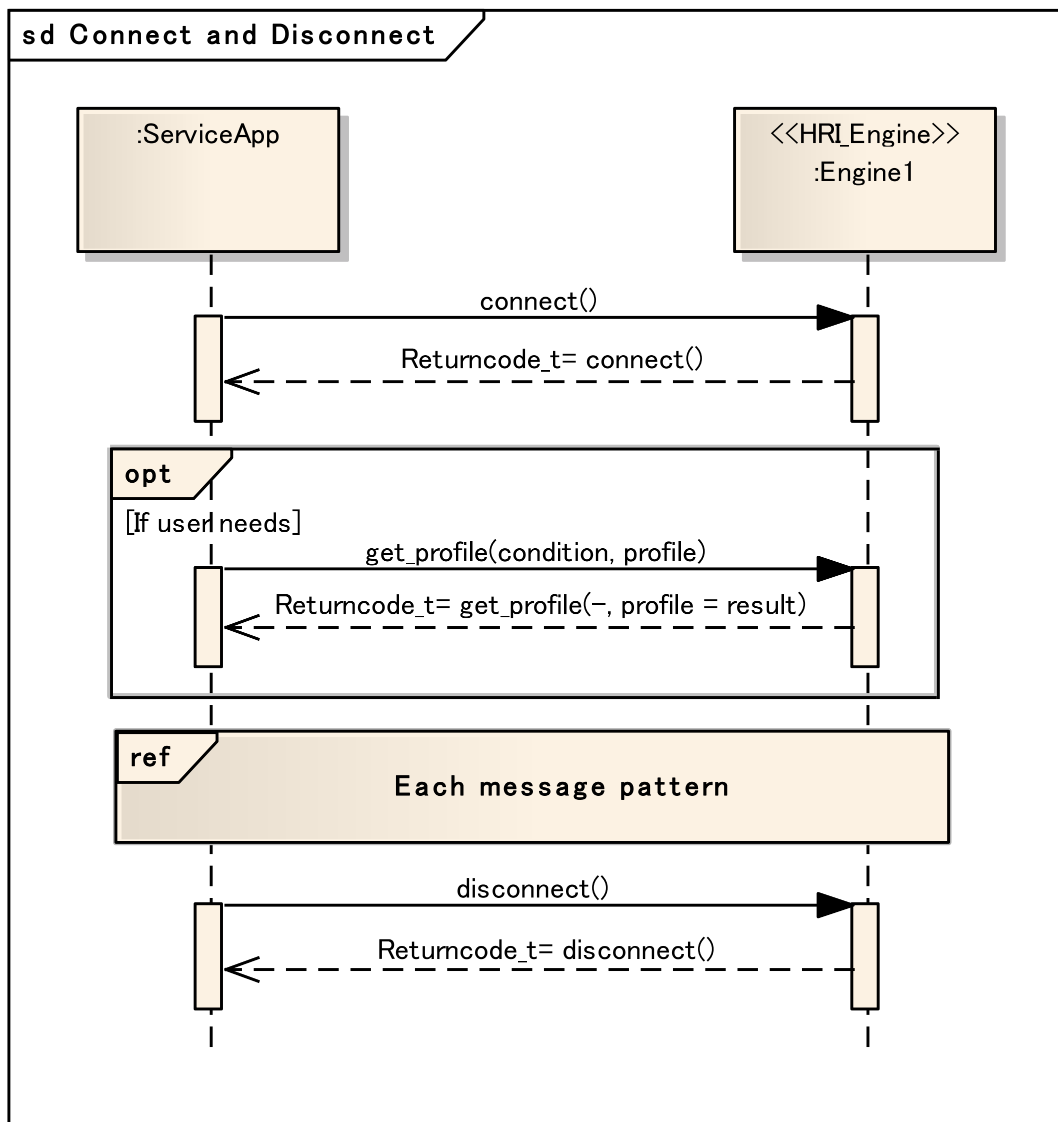

Figure 6: Sequence Diagram of System Interface (Connect / Disconnect)

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.

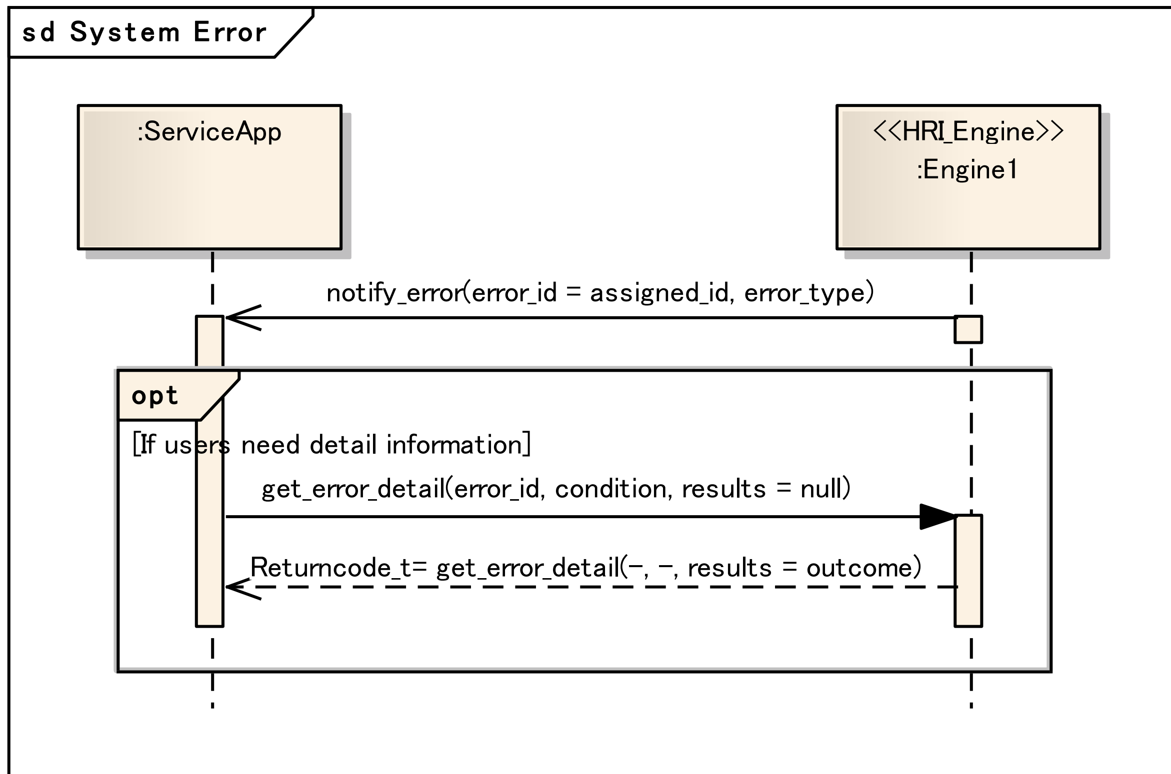

Figure 7: Sequence Diagram of System Interface (System Error)

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.

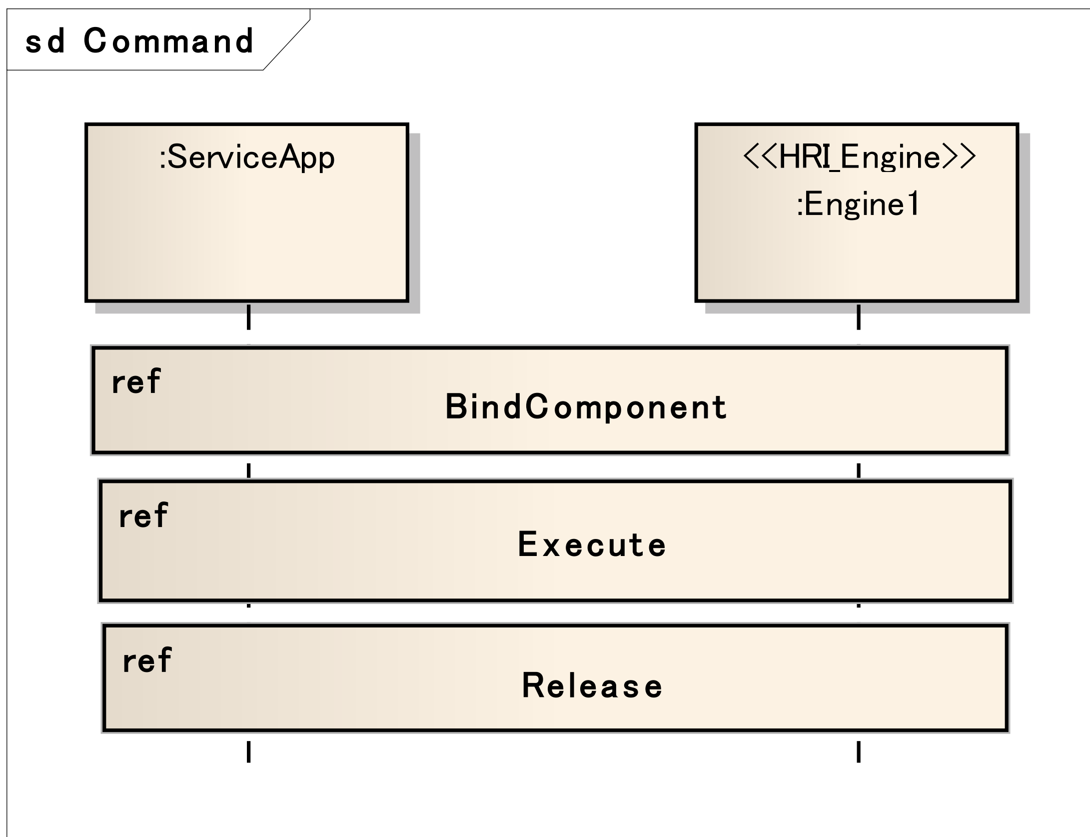

Figure 8: Sequence Diagram of Command Interface

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.

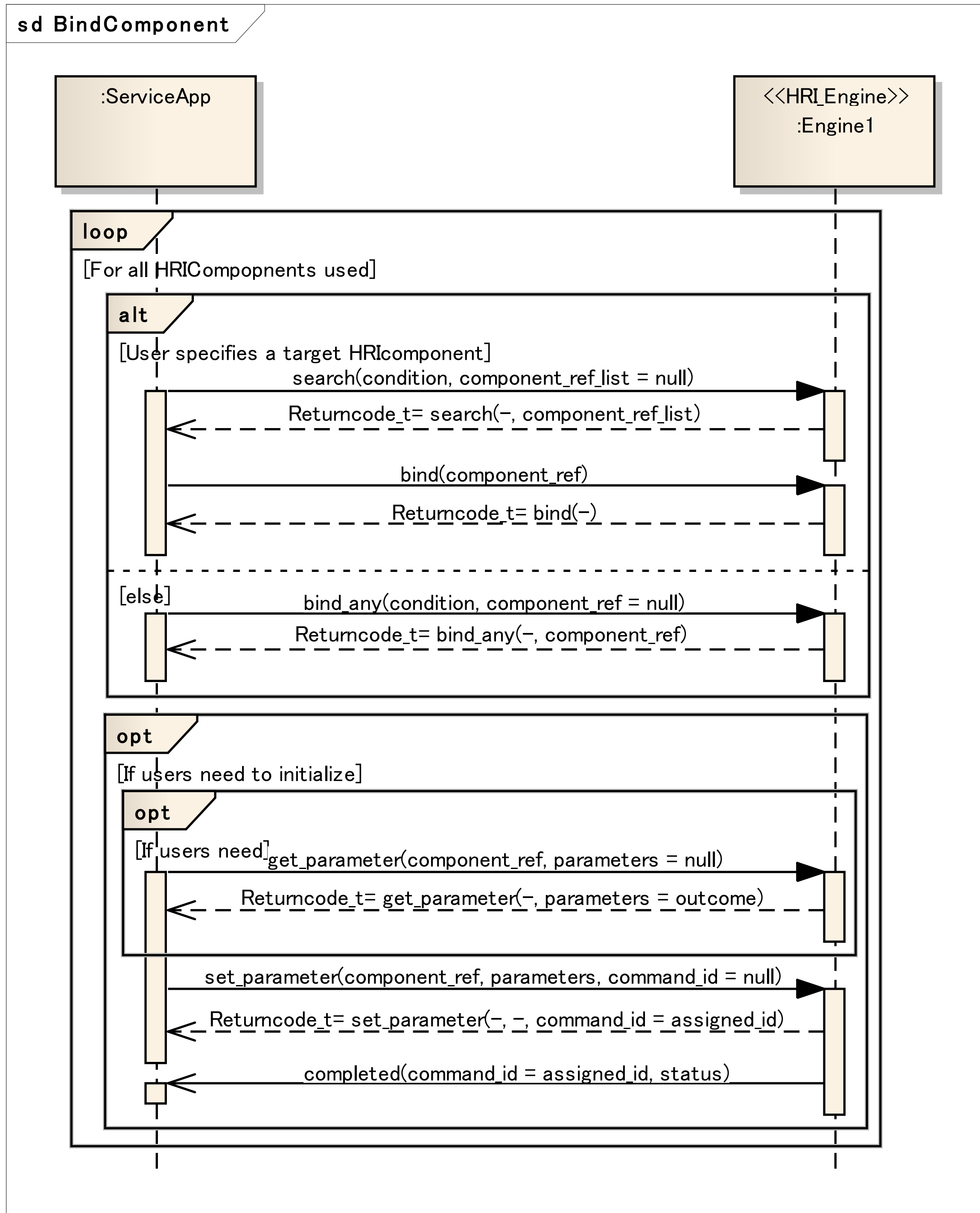

Figure 9: Sequence Diagram of "BindComponent" in Command Interface

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.

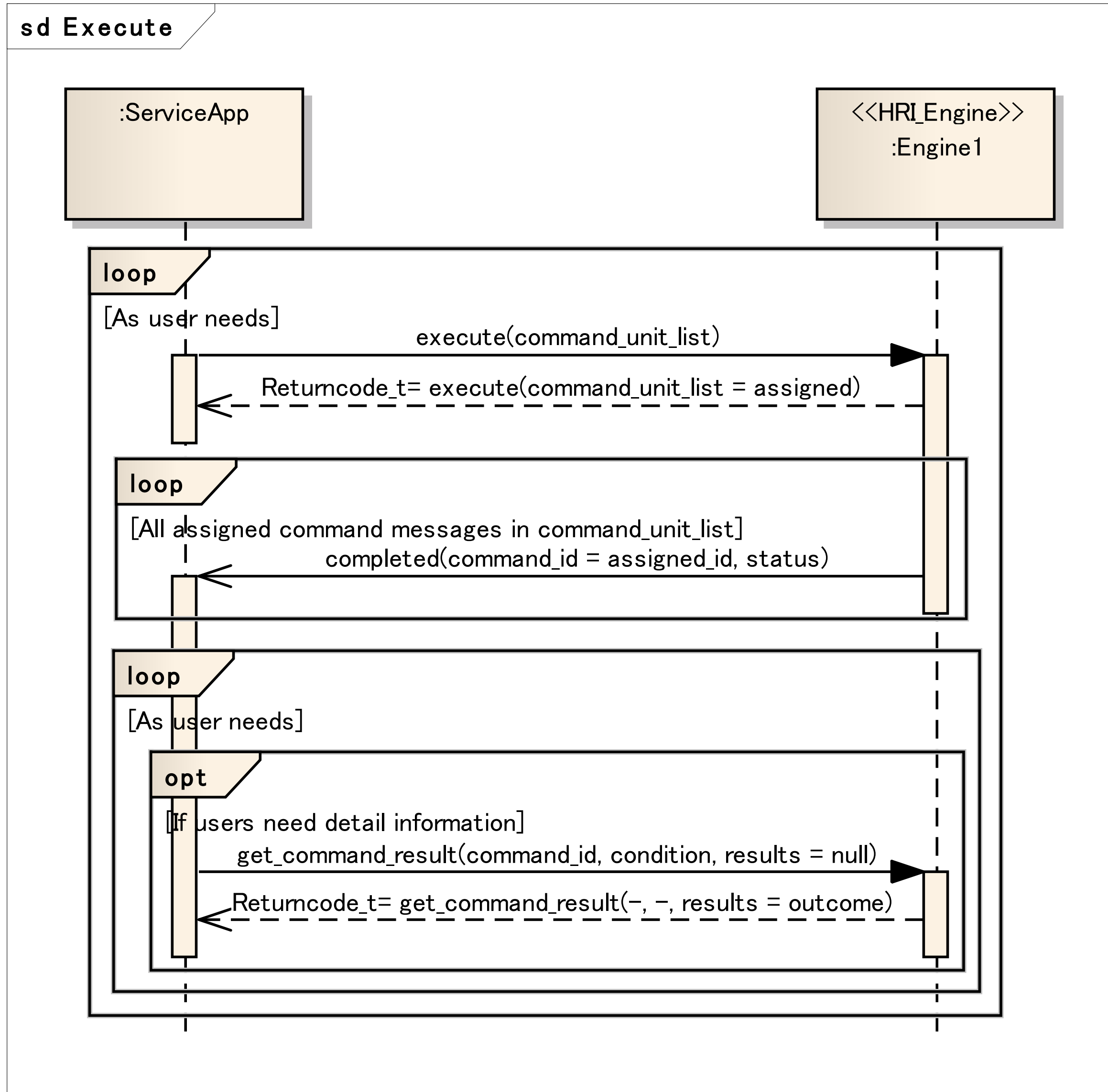

Figure 10: Sequence Diagram of "Execute" in Command Interface

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().

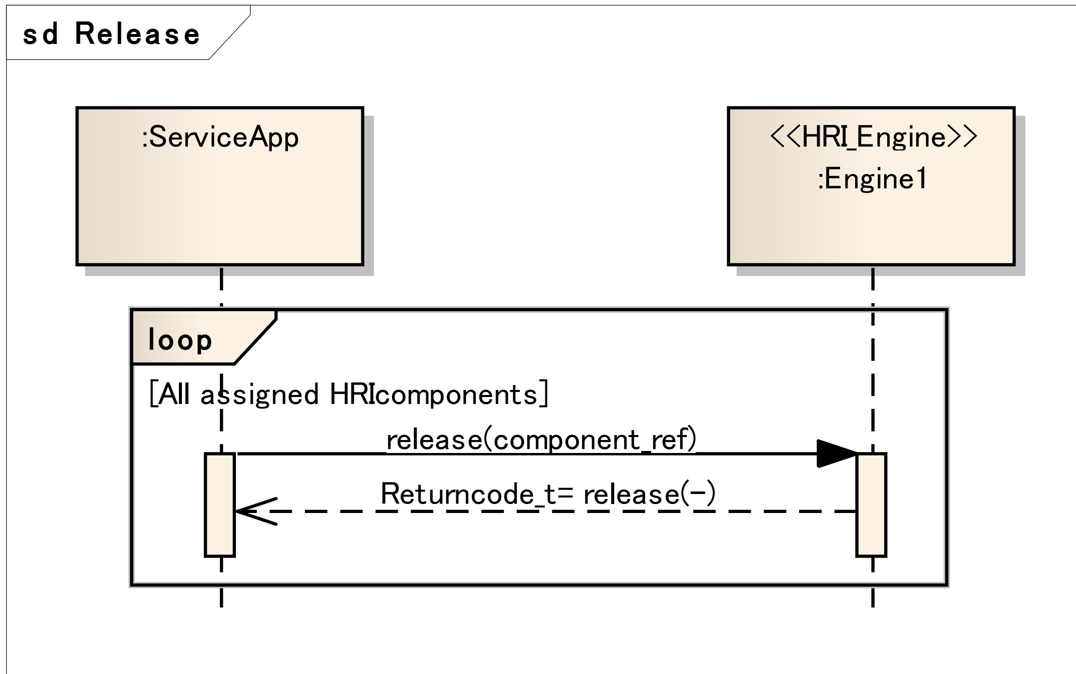

Figure 11: Sequence Diagram of "Release" in Command Interface

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 follows 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.

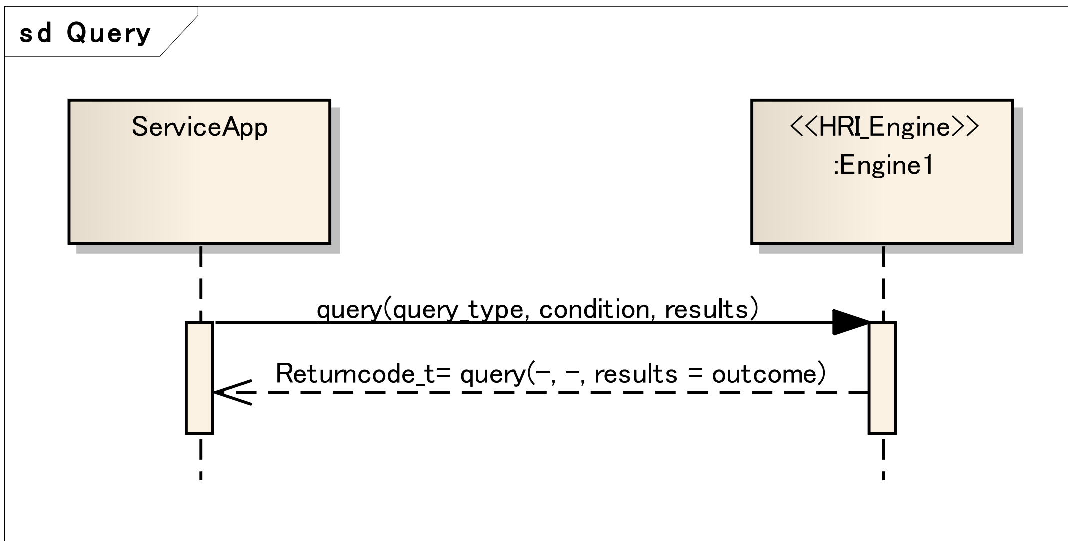

Figure 12: Sequence Diagram of Query Interface

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.

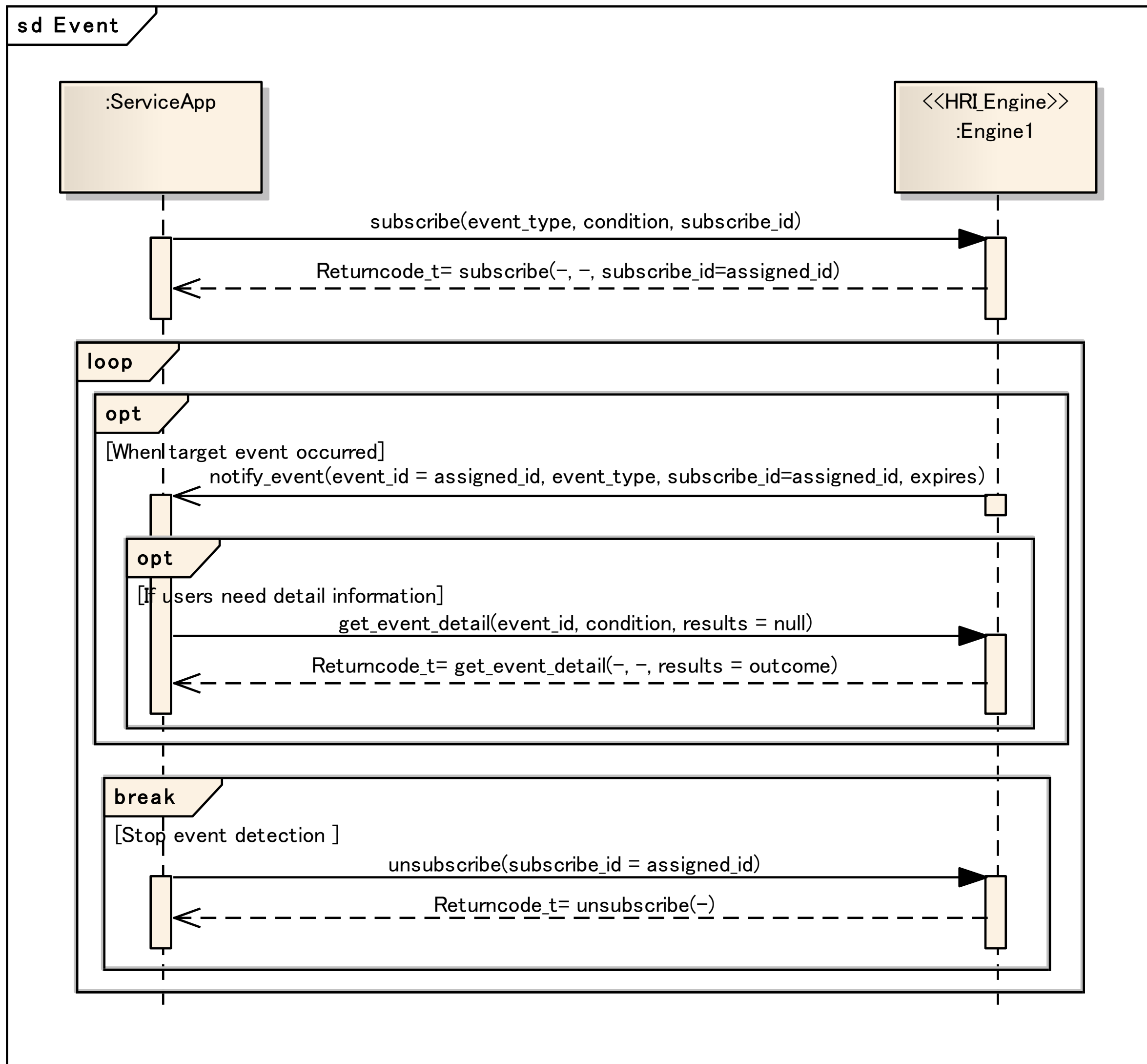

Figure 13: Sequence Diagram of Event Interface.

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 .

7.4.2 Interfaces

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

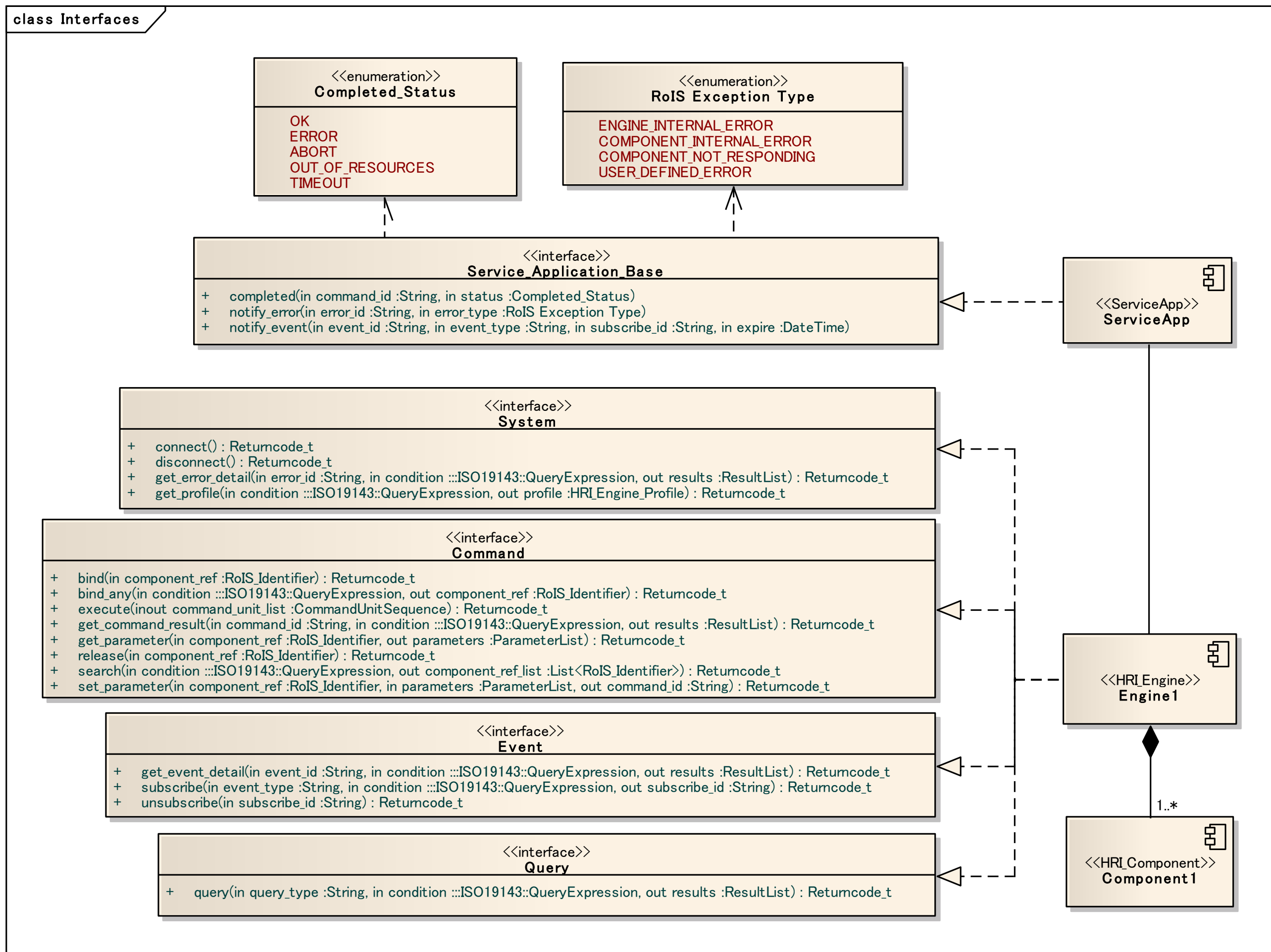

Figure 14 : RoIS Interfaces.

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

Description: The interface required to enable the HRI Engine to receive requests related to system management from the Service Application.	
Derived From: None	
Operations	
connect	Connects to the HRI Engine.
disconnect	Disconnects from the HRI Engine.

get_profile		Obtains the profile.	
in	condition	QueryExpression [ISO19143]	Specifies conditions of the profile to be obtained.
out	profile	HRI_Engine_Profile	Holds the obtained HRI Engine profile.
get_error_detail		Obtains details on an error notification from the HRI Engine.	
in	error_id	String	Specifies the ID identifying the error event assigned at the time of error-event notification.
in	condition	QueryExpression [ISO19143]	Specifies conditions for the error information to be obtained.
out	results	ResultList	Holds error information.

Table 7.3: Command Interface

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.	
in	condition	QueryExpression [ISO19143]	Specifies the conditions for the HRI Component to be searched for.
out	component_ref_list	List< RoIS_Identifier>	Holds a list of IDs for components that match specified conditions.
bind		Binds the specified HRI Component.	
in	component_ref	RoIS_Identifier	Specifies the ID of the HRI Component to be bound.
bind_any		Has the HRI Engine automatically select and bind an HRI Component that matches the conditions for executing a function.	
in	condition	QueryExpression [ISO19143]	Specifies the conditions of the HRI Component to be selected.
out	component_ref	RoIS_Identifier	Holds the ID of the bound HRI Component.
release		Releases the specified HRI Component.	
in	component_ref	RoIS_Identifier	Specifies the ID of the HRI Component to be released.
get_parameter		Obtains parameters of the bound HRI Component.	
in	component_ref	RoIS_Identifier	Specifies the ID of the bound HRI Component.
out	parameters	ParameterList	Holds the obtained parameters.
set_parameter		Sets parameters of the bound HRI Component.	
in	component_ref	RoIS_Identifier	Specifies the ID of the bound HRI Component.
in	parameters	ParameterList	Specifies the parameters to be set.
out	command_id	String	Holds the command ID assigned for this command message.
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.	

in	command_id	String	Specifies the command ID assigned for this command message.
in	condition	QueryExpression [ISO19143]	Specifies the conditions for obtaining command-execution results.
out	results	ResultList	Holds command-execution results.

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			
query	Sends a query message to the HRI Engine and obtains information.		
in	query_type	String	Specifies the type of the query message to be sent.
in	condition	QueryExpression [ISO19143]	Specifies the conditions of the information to be obtained.
out	results	ResultList	Holds the obtained information.

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			
subscribe	Registers an event message for which notifications are to be received.		
in	event_type	String	Specifies the type of the event message to be registered.
in	condition	QueryExpression [ISO19143]	Specifies the conditions of the event message to be registered.
out	subscribe_id	String	Holds the event-registration ID assigned when registering this event message.
unsubscribe	Cancels the registered event message.		
in	subscribe_id	String	Specifies the event-registration ID assigned when registering this event message.
get_event_detail	Obtains detailed information on this event notification.		
in	event_id	String	Specifies the ID of the event notification assigned at the time of this event-message notification.
in	condition	QueryExpression [ISO19143]	Specifies the conditions of the information to be obtained.
out	results	ResultList	Holds detailed information on the event notification.

7.4.2.2 Interfaces for Service Application

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

Table 7.6: Service Application Base Interface

Description: The interface required to enable the Service Application to receive notifications from the HRI Engine.			
Derived From: None			
Operations			
notify_event	Receives event message for which notification has been registered.		
in	event_id	String	Holds the ID of the event notification assigned when sending the event message.
in	event_type	String	Holds the ID of this event message.
in	subscribe_id	String	Holds the event-registration ID assigned when registering this event message.
in	expire	DateTime[W3C-DT]	Time limit for obtaining detailed results by get_event_detail().
notify_error	Receives an error notification from the HRI Engine or the HRI Component.		
in	error_id	String	Holds the ID of the error notification assigned when notifying of this error.
in	error_type	ErrorType	Holds the type of error.
completed	Receives notification that command execution has completed.		
in	command_id	String	Holds the command ID assigned when the command message was sent.
in	status	Completed_Status	Holds the state of command completion.

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

Table 7.7: ExceptionType enumeration

ENGINE_INTERNAL_ERROR	An error internal to the HRI Engine.
COMPONENT_INTERNAL_ERROR	An error internal to the HRI Component.
COMPONENT_NOT_RESPONDING	No response received from the HRI Component.
USER_DEFINED_ERROR	An error defined by the user.
Note: Corresponding situations of these error types shall be defined with respect to each HRI Engine.	

Table 7.8: Completed_Status enumeration

OK	Successful return.
ERROR	Generic, unspecified error.
ABORT	The operation was aborted.
OUT_OF_RESOURCES	Service ran out of the resources needed to complete the operation.
TIMEOUT	The operation timed out.

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.

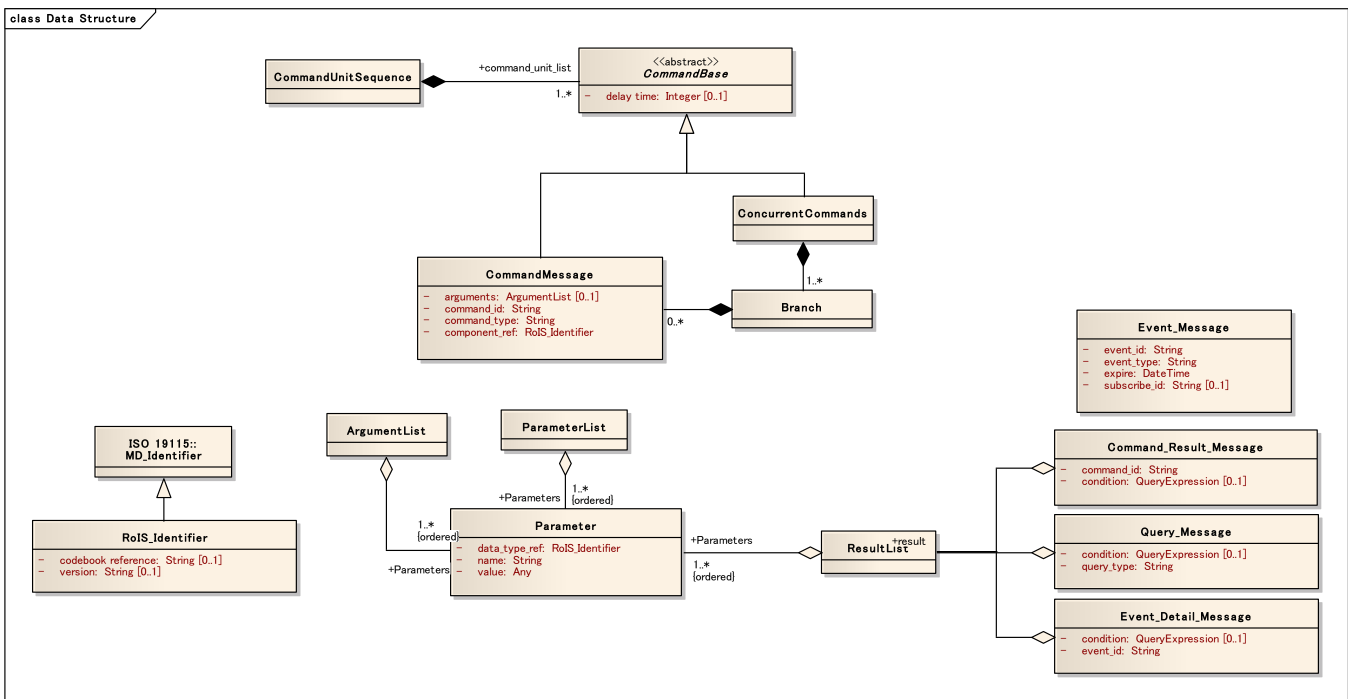

Figure 15: Data Structure of Command Message.

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

Table 7.9: RoIS_Identifier

Description: A data type for describing an ID that identifying an instance and the reference codebook for the ID.				
Derived From : MD_Identifier [ISO19115]				
Attributes				
codebook reference	String	O	1	URI of the codebook used.
version	String	O	1	Version identifier for the codebook.

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

branch list	Branch	M	N	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 messagessequentially.				
Derived From :				
Attributes				
command list	CommandMessage	M	Nord	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

Table 7.18: ParameterList class

Description: A data class for specifying a list of configuration parameters				
Derived From: None				
Attributes				
Parameters	Parameter	M	Nord	Configuration 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

Description: A data class for specifying a command result message				
Derived From: None				
Attributes				
command_id	String	M	1	ID of the command transmission assigned when receiving the command message
condition	QueryExpression [ISO19143]	O	1	Conditions of information to be obtained
results	ResultList	M	1	Results of command execution

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

Description: A data class for specifying a query message				
Derived From: None				
Attributes				
query_type	String	M	1	type of the query message
condition	QueryExpression [ISO19143]	O	1	Conditions of information to be obtained
results	ResultList	M	1	Obtained information

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

Description: A data class for specifying an event message
Derived From: None
Attributes

event_id	String	M	1	ID of the event notification assigned when sending the event message
event_type	String	M	1	type of the event message
subscribe_id	String	M	1	ID of event registration assigned when registering the event message
expire	DateTime[W3C-DT]	O	1	Time limit for obtaining detailed results by get_event_detail().

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

Description: A data class for specifying an event detail message				
Derived From: None				
Attributes				
event_id	String	M	1	ID of the event notification assigned when sending the event message
condition	QueryExpression [ISO19143]	O	1	Conditions of information to be obtained
results	ResultList	M	1	Detailed information on event

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

Description: A data class for specifying an error message				
Derived From: None				
Attributes				
error_id	String	M	1	ID of the error notification assigned when sending the event message
error_type	String	M	1	type of the error message
subscribe_id	String	M	1	ID of event registration assigned when registering the event message
expire	DateTime[W3C-DT]	O	1	Time limit for obtaining detailed results by get_error_detail().

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

Description: A data class for specifying an error detail message
Derived From: None

Attributes				
error_id	String	M	1	ID of the error notification assigned when sending the error message
condition	QueryExpression [ISO19143]	O	1	Conditions of information to be obtained
results	ResultList	M	1	Detailed information on error

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.

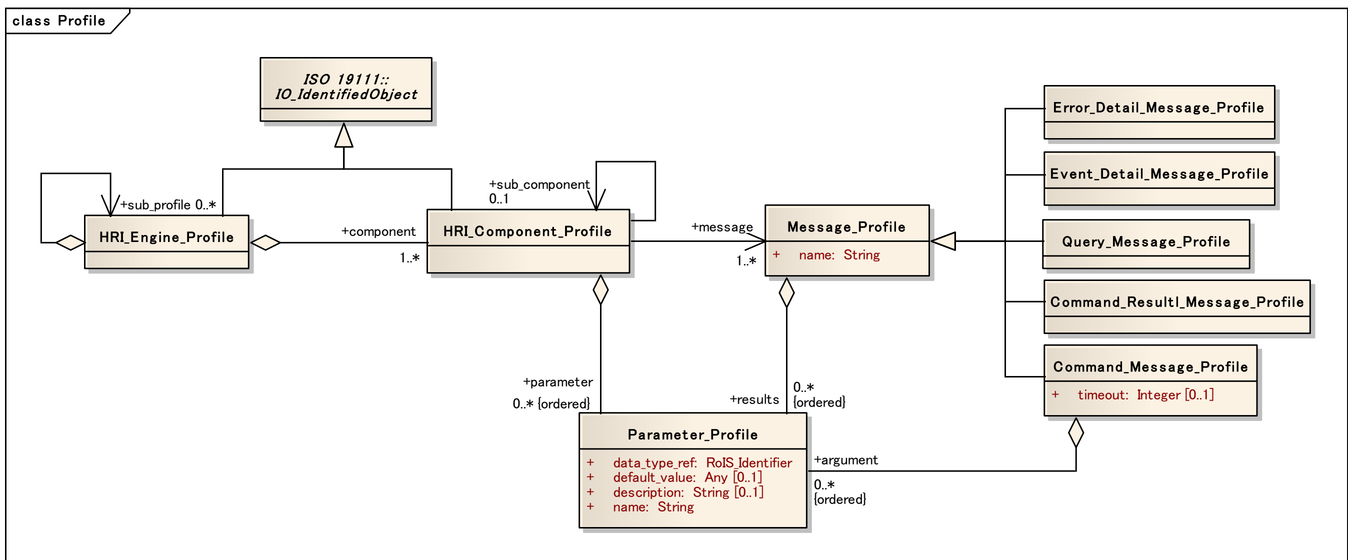

Figure 16: RoIS Profile. RoIS profile mainly consists of 4 types of profiles, i.e. "HRI_Engine_Profile", "HRI_Component_Profile", "Message_Profile", and "Parameter_Profile".

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 7.25: Parameter_Profile

Description: Profile for defining each parameter for HRI Engines.				
Derived From :				
Attributes				
name	String	M	1	Parameter name
data_type_ref	RoIS_Identifier	M	1	Reference ID of data definition
default_value	Any	O	1	Necessary arguments when issuing this message
description	String	O	1	Description

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				
name	String	M	1	Message name
results	Parameter_Profile	O	N ord	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.

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	1	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

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

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

Description: Profile for defining messages for query interface type.
Derived From : Message_Profile

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

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

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

Description: Profile for defining messages for system interface type.
Derived From : Message_Profile

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

Description: Profile for defining lists of messages and parameters for each HRI Component
Derived From : IO_IdentifiedObject [ISO19111]
Attributes

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

Description: Profile for defining lists of logical units and parameters for each HRI Engine and sub HRI Engine.				
Derived From : IO_IdentifiedObject [ISO19111]				
Attributes				
component	HRI_Component_Profile	M	N	Specifies the HRI Component Profile of an HRI Component of this HRI Engine. Multiple items may be defined.
sub_profile	HRI_Engine_Profile	O	N	Specifies the sub HRI Engine Profile included in this HRI Engine. Multiple items may be defined.

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

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

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

Table 7.35: RoIS_Common

Description: common method for all HRI Components.				
Command Method				
start	Start the functionality of the HRI Component.			
stop	Stop the functionality of the HRI Component.			
suspend	Pause the functionality of the HRI Component.			
resume	Resume the functionality of the HRI Component.			
Query Method				
component_status	Obtain status information of the component.			
result	status	Component_Status	M	Status information of this component.

Component status is defined as follows.

Table 7.36: Component_status enumeration

UNINITIALIZED	The component is not initialized.
READY	The component is ready to use.
BUSY	The component is used by other application(s).
WARNING	Warning against the use of the component
ERROR	Generic, unspecified error.

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

7.6.1 System Information

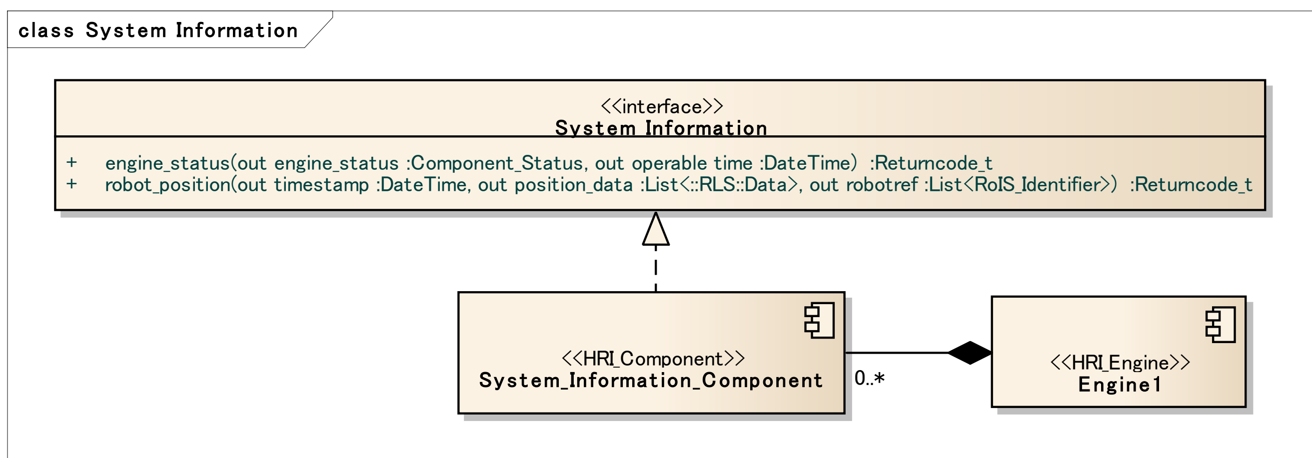

Figure 17: System Information

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				
robot_position	Returns location information.			
result	timestamp	DateTime [W3C-DT]	M	Measurement time.
result	robot_ref	List<RoIS_Identifier>	M	List of the robot IDs.
result	position data	List<Data> [RLS]	M	List of location data. Each entry at least contains ID of the location data. This may also be accompanied with additional information such as position or pose of the robot, sensor or actuator. It may also contain certainty of the localization act.
engine_status	Returns status information of the HRI Engine.			
result	status	Component_Status	M	Status information of this engine.
result	operable time	DateTime [W3C-DT]	O	Operable time of the HRI Engine that includes this basic component.

7.6.2 Person Detection

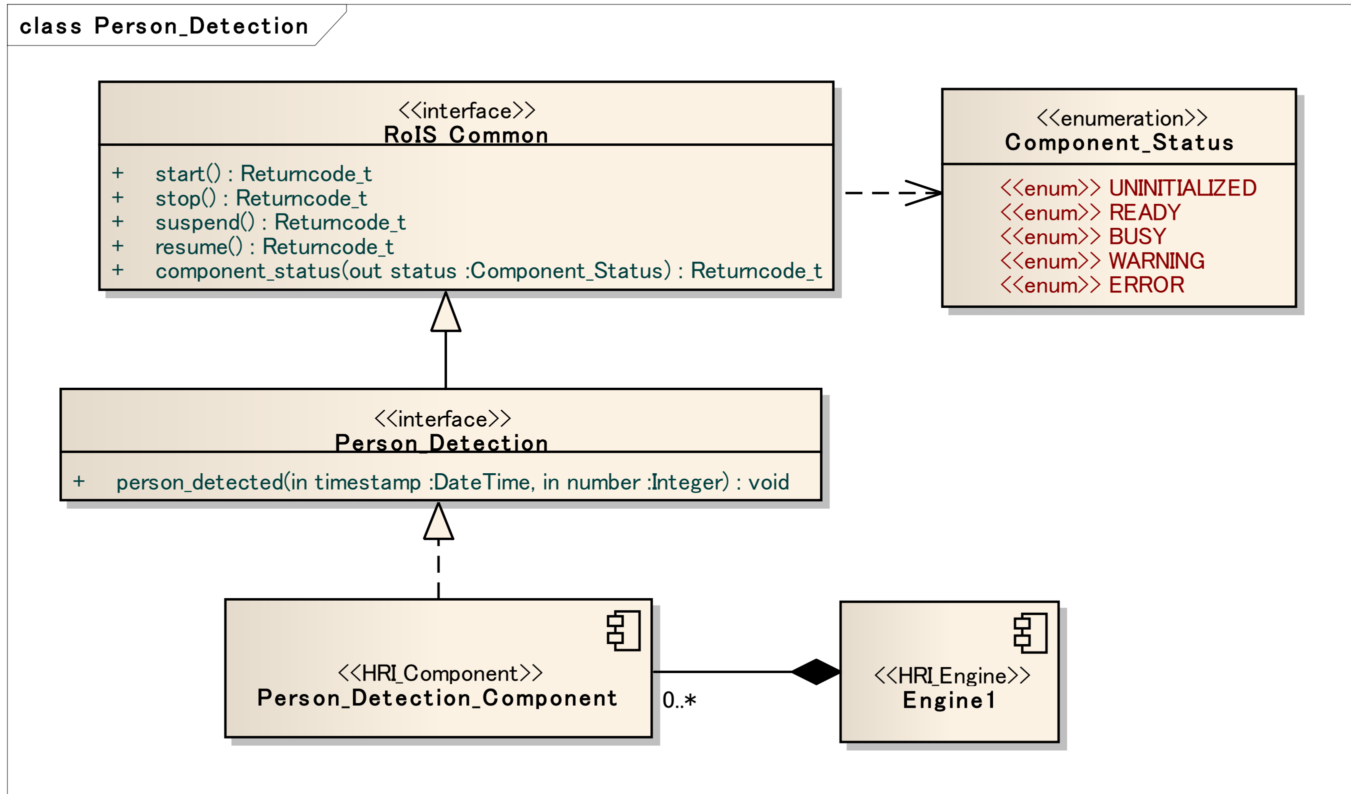

Figure 18 : person detection

Table 7.38: person detection

Description: This is a component for detecting number of persons. This component notifies a number of the detected people when the number has changed. This functionality is essential for typical robotic services; for example, if a Service Application is required to start its service when a person enters the service area, this component is effective to detect the entry of people. Similarly, if the Service Application needs to stop the service when the person moves out of the service area, this component can also detect the exit of people.				
Event Method				
person_detected	Notifies number of people when the number has changed.			
result	timestamp	DateTime [W3C-DT]	M	Measurement time.
result	number	Integer	M	Number of detected persons

7.6.3 Person Localization

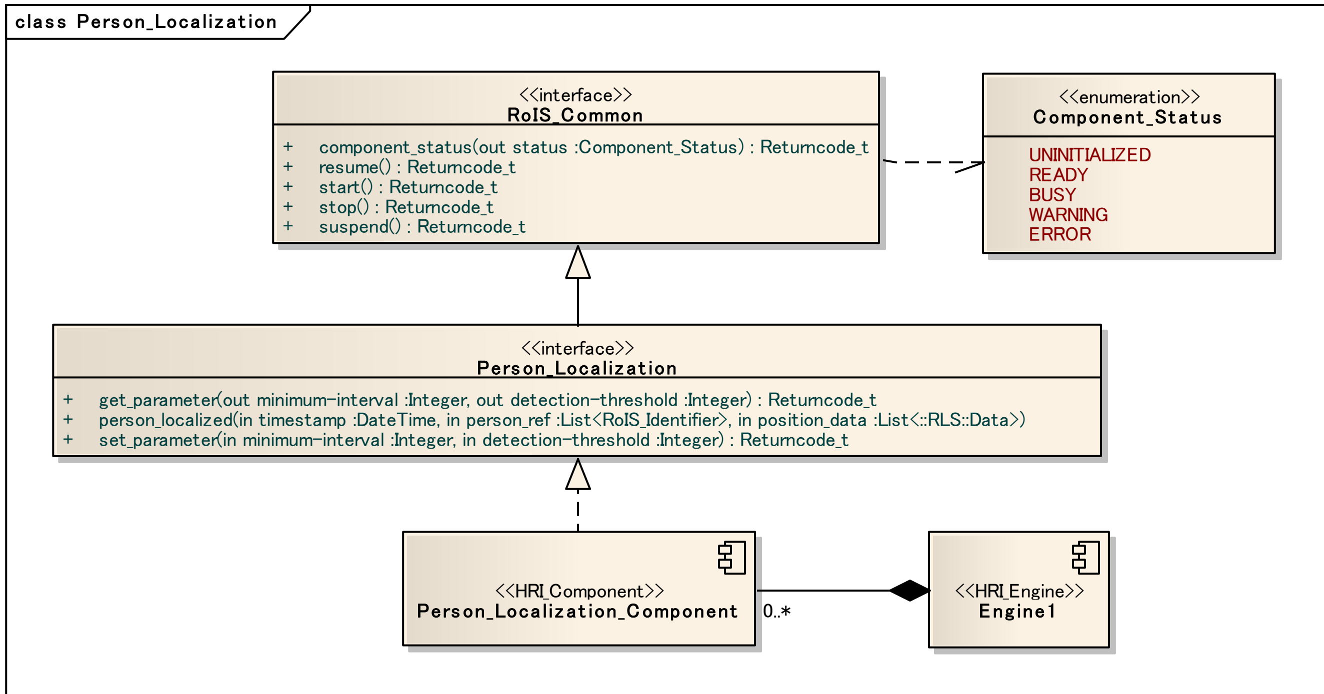

Figure 19 : person localization

Table 7.39: person localization

<table>
<tr><td colspan="5">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.</td></tr>
<tr><td colspan="5">Command Method</td></tr>
<tr><td colspan="2">set_parameter</td><td colspan="3">Specifies person localization parameters.</td></tr>
<tr><td>argument</td><td>detection threshold</td><td>Integer</td><td>O</td><td>This component notifies an event if the distance of movement since previous event notification exceeds the threshold value.</td></tr>
<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>
<tr><td colspan="5">Query Method</td></tr>
<tr><td colspan="2">get_parameter</td><td colspan="3">Obtains person localization parameters.</td></tr>
</table>

result	detection threshold	Integer	O	This component notifies an event if the distance of movement since previous event notification exceeds the threshold value.
result	detection threshold	Integer	O	This component notifies an event if the distance of movement since previous event notification exceeds the threshold value.
Event Method				
person_localized	Notifies position of people when the position has localized.			
result	timestamp	DateTime [W3C-DT]	M	Measurement time.
result	person ref	List<RoIS_Identifier>	M	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.
result	position data	List<Data[RLS]>	M	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.

7.6.4 Person Identification

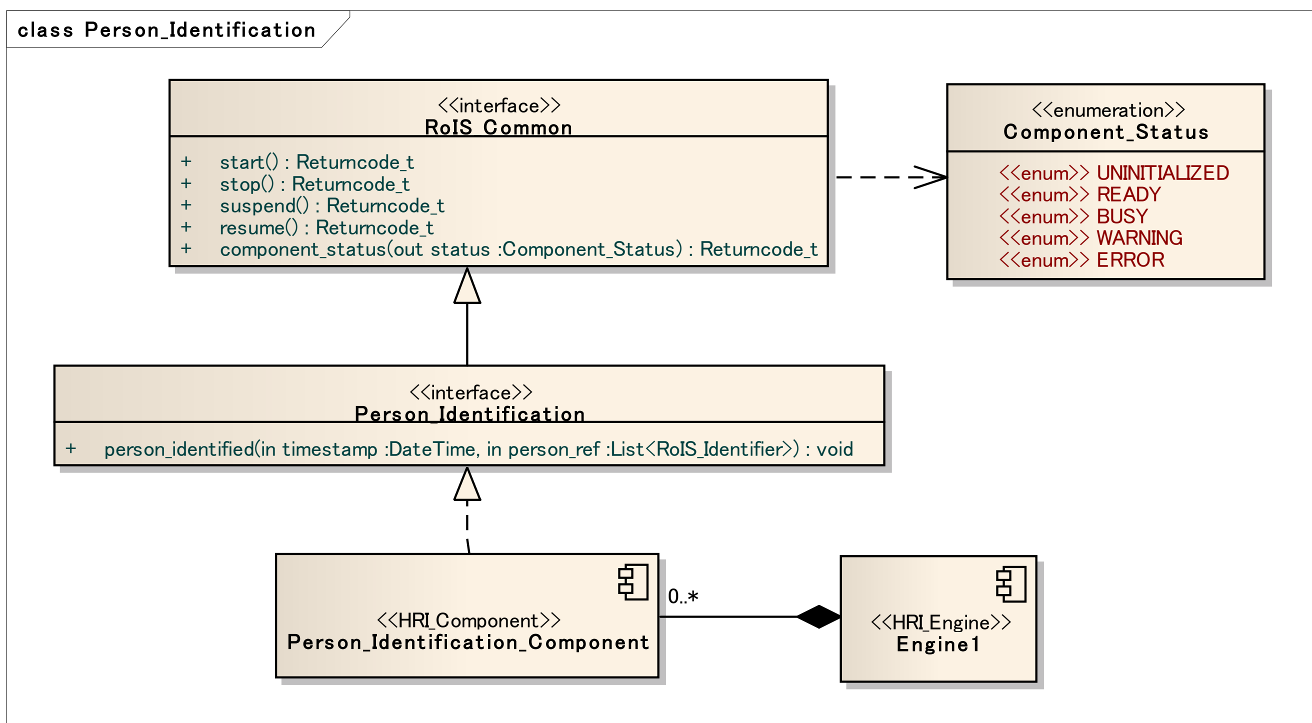

Figure 20: person identification

Table 7.40: person identification

Description: This is a component for identifying person ID. This component notifies ID(s) of the detected people when the ID(s) has been identified. This functionality is essential for performing various robotic services, from simply calling by one's name to performing advanced services based on person profiles or service histories. Numbers of methods and means for identification have been proposed and have been used so far, such as face, iris or gate recognition. This HRI Component provides an abstract mean for utilizing person recognition results.				
Event Method				
person_identified	Notifies ID of people when the ID has identified.			
result	timestamp	DateTime [W3C-DT]	M	Measurement time.
result	person ref	List<RoIS_Identifier>	M	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.

7.6.5 Face Detection

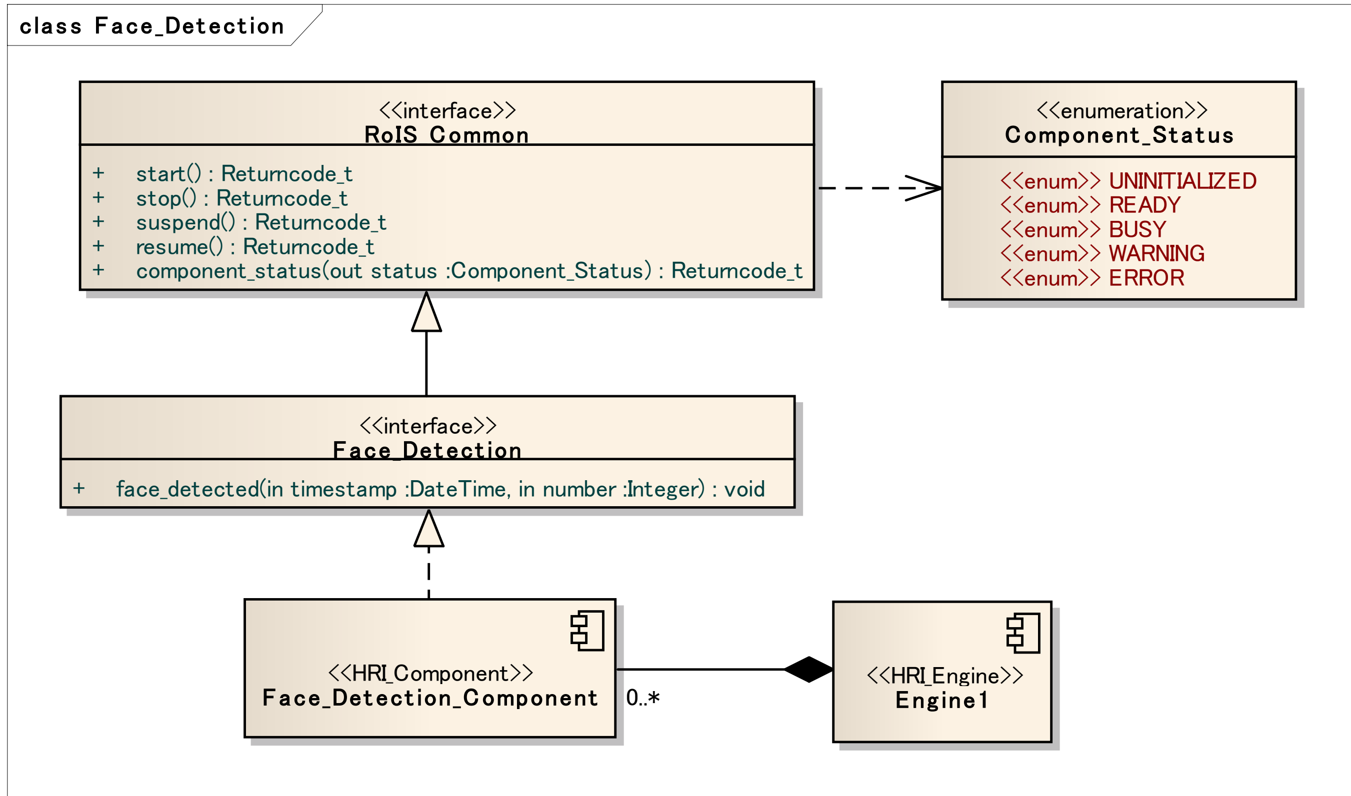

Figure 21 : face detection

Table 7.41: face detection

<table>
<tr><td colspan="5">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.</td></tr>
<tr><td colspan="5">Event Method</td></tr>
<tr><td colspan="2">face_detected</td><td colspan="3">Notifies number of human face when the number has changed.</td></tr>
<tr><td>result</td><td>timestamp</td><td>DateTime [W3C-DT]</td><td>M</td><td>Measurement time.</td></tr>
<tr><td>result</td><td>number</td><td>Integer</td><td>M</td><td>Number of human faces</td></tr>
</table>

7.6.6 Face Localization

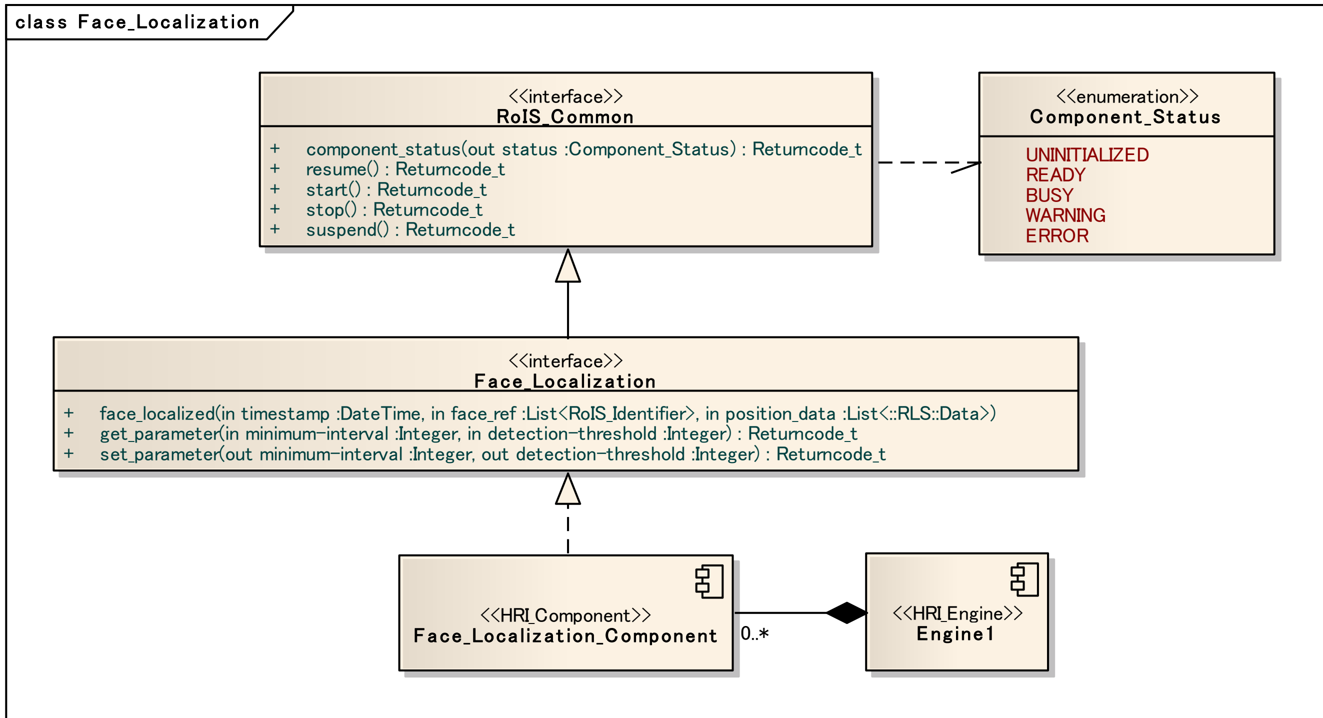

Figure 22: face localization

Table 7.42: 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.				
Command Method				
	set_parameter	Specifies face localization parameters.		
argument	detection threshold	Integer	O	This component notifies an event if the distance of movement since previous event notification exceeds the threshold value.
argument	minimum interval	Integer	O	This component notifies an event if the period since previous event notification exceeds the value of minimal interval.
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	O	This component notifies an event if the period since previous event notification exceeds the value of minimal interval.
Event Method				
face_localized	Notifies position of human face when the position has localized.			
result	timestamp	DateTime [W3C-DT]	M	Measurement time.
result	face ref	List<RoIS_Identifier>	M	List of detected human face IDs. Reference information related to the ID shall be provided with the 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.
result	position data	List<Data[RLS]>	M	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

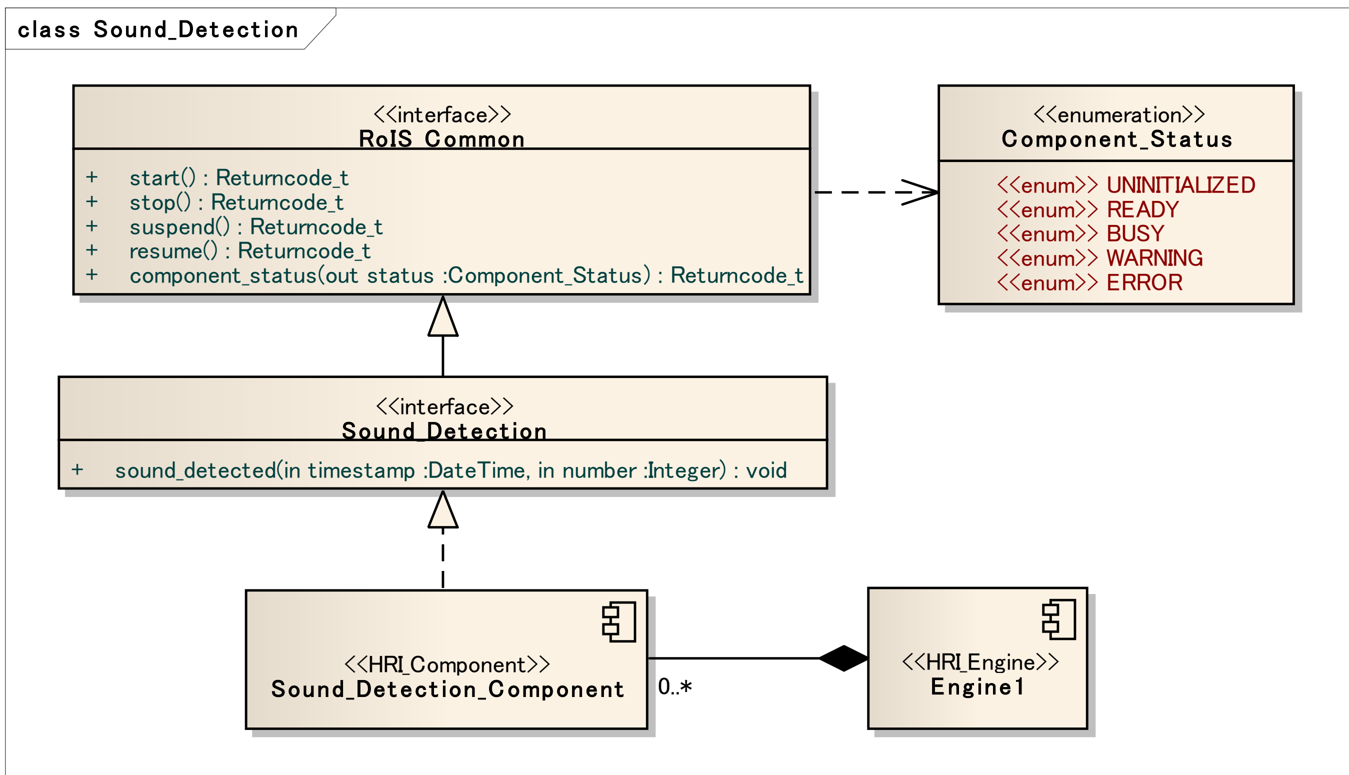

Figure 23: sound detection

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				
	sound_detected	Notifies number of sound sources when the number has changed.		
result	timestamp	DateTime [W3C-DT]	M	Measurement time.
result	number	Integer	M	Number of sound sources. If the component can not detect sound sources separately, this parameter shall be 1 or 0.

7.6.8 Sound Localization

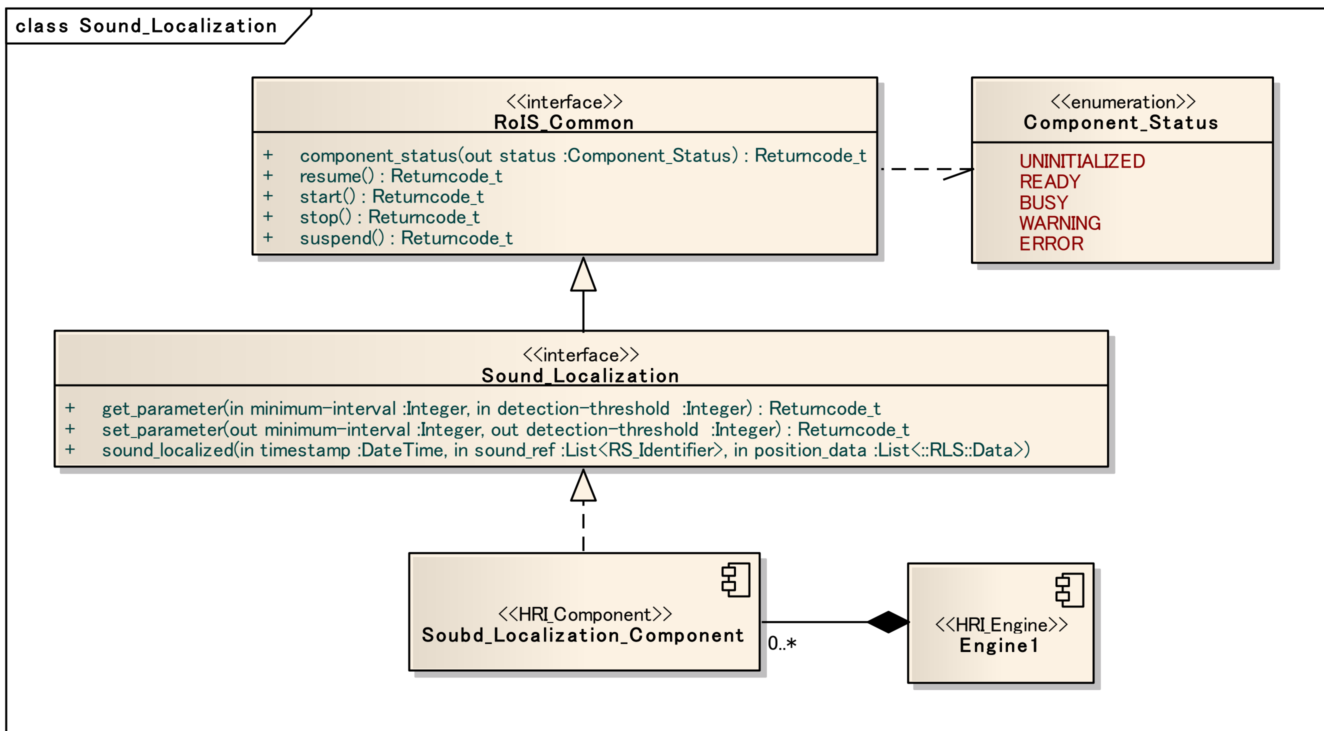

Figure 24: sound localization

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.				
Command Method				
set_parameter		Specifies sound localization parameters.		
argument	detection threshold	Integer	O	This component notifies an event if the distance of movement since previous notification exceeds the threshold value.
argument	minimum interval	Integer	O	This component notifies an event if the period since previous event notification exceeds the threshold value.
Query Method				
get_parameter		Obtains sound localization parameters.		
result	detection threshold	Integer	O	This component notifies an event if the distance of movement since previous notification exceeds the threshold value.
result	minimum interval	Integer	O	This component notifies an event if the period since previous event notification exceeds the value of minimal interval.
Event Method				
sound_localized		Notifies position of sound sources when the position has localized.		
result	timestamp	DateTime [W3C-DT]	M	Measurement time.
result	sound ref	List<RoIS_Identifier>	M	List of detected sound source IDs. Reference information related to the ID shall be provided with the 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.
result	position data	List<Data[RLS]>	M	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.

7.6.9 Speech Recognition

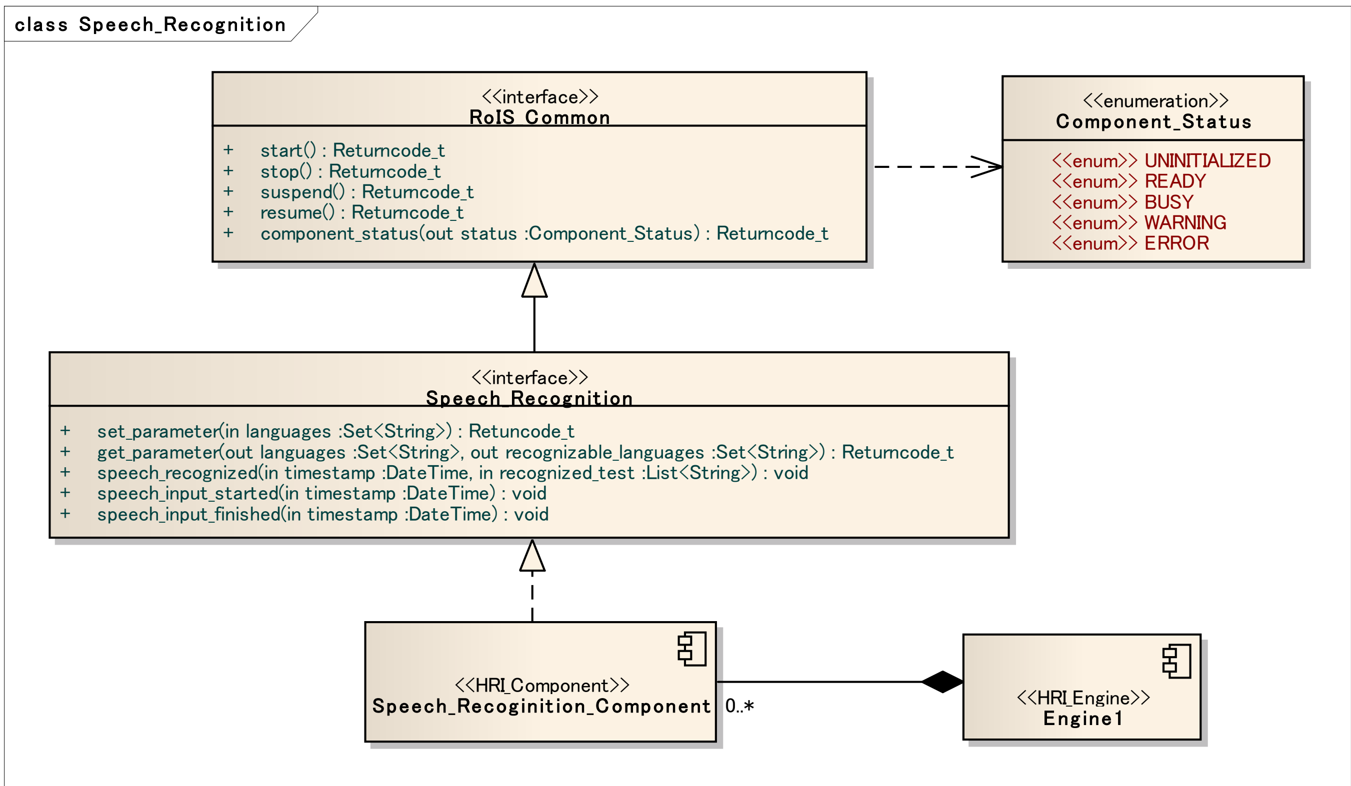

Figure 25: speech recognition

Table 7.45: speech recognition

<table>
<tr><td colspan="5">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 assumes 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.</td></tr>
<tr><td colspan="5">Command Method</td></tr>
<tr><td colspan="2">set_parameter</td><td colspan="3">Specifies speech recognition paramters.</td></tr>
<tr><td>argument</td><td>languages</td><td>Set<String> [ISO639-1]</td><td>M</td><td>Specifies languages the speech recognizer will recognize.</td></tr>
<tr><td>argument</td><td>grammer</td><td>String</td><td>O</td><td>Specifies grammar for the speech recognizer.</td></tr>
<tr><td>argument</td><td>rule</td><td>String</td><td>O</td><td>Specifies active rule in the grammer.</td></tr>
<tr><td colspan="5">Query Method</td></tr>
<tr><td colspan="2">get_parameter</td><td colspan="3">Obtains speech recognition paramters.</td></tr>
</table>

result	recognizable languages	Set<String> [ISO639-1]	M	Obtains languages the speech recognizer can recognize.
result	languages	Set<String> [ISO639-1]	M	Obtains languages the speech recognizer recognizes.
result	grammer	String	O	Obtains grammar for the speech recognizer.
result	rule	String	O	Obtains active rule in the grammer.
Event Method				
speech_recognized	Notifies recognized result when the speech has been recognized.			
result	timestamp	DateTime [W3C-DT]	M	Time when the recognition has completed.
result	recognized text	List<String>	M	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.
speech_input_started	Notifies the recognizer has detected start of speech input.			
result	timestamp	DateTime [W3C-DT]	M	Time when the speech input has started.
speech_input_finished	Notifies the recognizer has detected end of speech input.			
result	timestamp	DateTime [W3C-DT]	M	Time when the speech input has ended.

7.6.10 Gesture Recognition

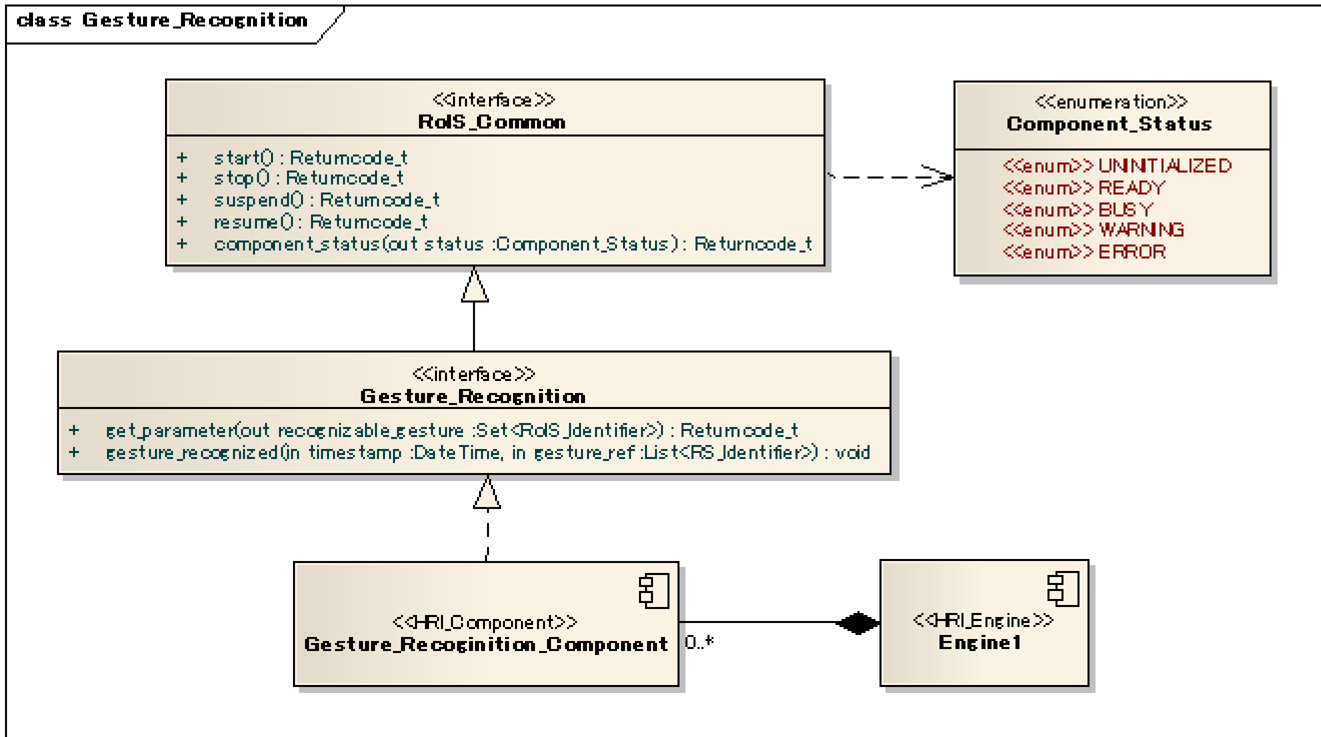

Figure 26: gesture recognition

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				
get_parameter		Obtains speech recognition paramters.		
result	recognizable gestures	Set<RoIS_Identifier>	M	Obtains gestures the gesture recognizer can recognize. The gesture is expressed as ID and the reference for the ID shall be provided with each ID.
Event Method				
gesture_recognized		Notifies recognized result when the gesture has been recognized.		
result	timestamp	DateTime [W3C-DT]	M	Measurement time.
result	gesture ref	List<RoIS_Identifier>	M	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.

7.6.11 Speech Synthesis

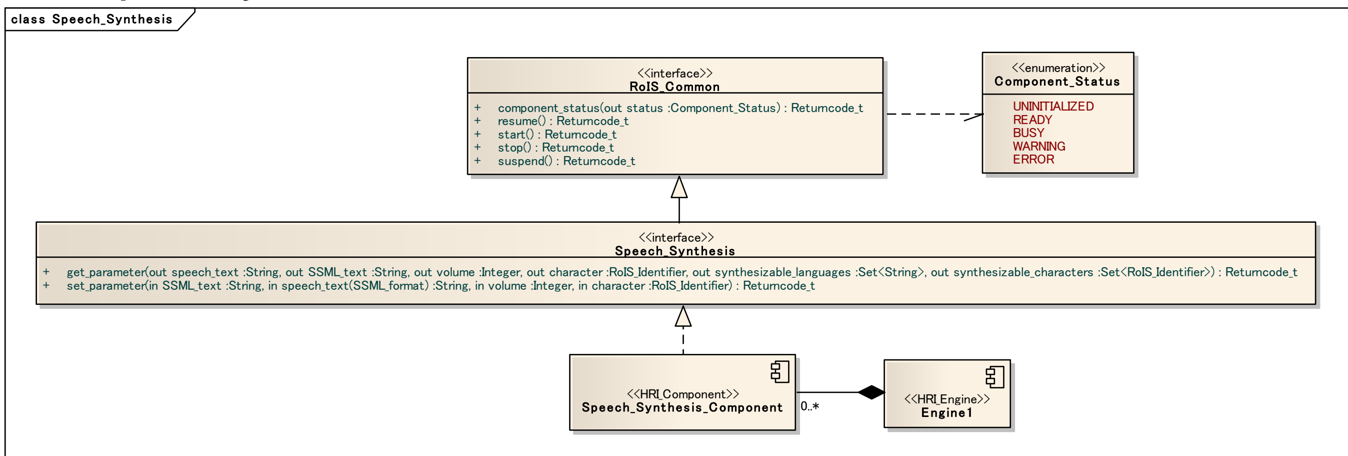

Figure 27: speech synthesis

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 assumes 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				
set_parameter	Specifies speech synthesis paramters.			
argument	speech_text	String	C	Text to synthesize (in plain text format).
argument	SSML text	String [W3C-SSML]	C	Text to synthesize (in W3C-SSML format).
argument	volume	Integer	O	Volume.
argument	language	String[ISO639-1]	O	Langage of the speech.
argument	character	RoIS_Identifier	O	Character of the voice.
Query Method				
get_parameter	Obtains speech synthesis paramters.			
result	speech_text	String	C	Information about specified text (in plain text format).
result	SSML text	String [W3C-SSML]	C	Information about specified text (in W3C-SSML format).
result	volume	Integer	O	Information about specified volume.
result	language	String[ISO639-1]	O	Information about specified language.
result	character	RoIS_Identifier	O	Information about specified character of the voice.
result	synthesizable_languages	Set<String> [ISO639-1]	O	Information about languages that can be synthesized.
result	synthesizable_characters	Set<RoIS_Identifier>	O	Information about characters that can be synthesized.
Condition: These elements shall be selected according to the speech text format.				

7.6.12 Reaction

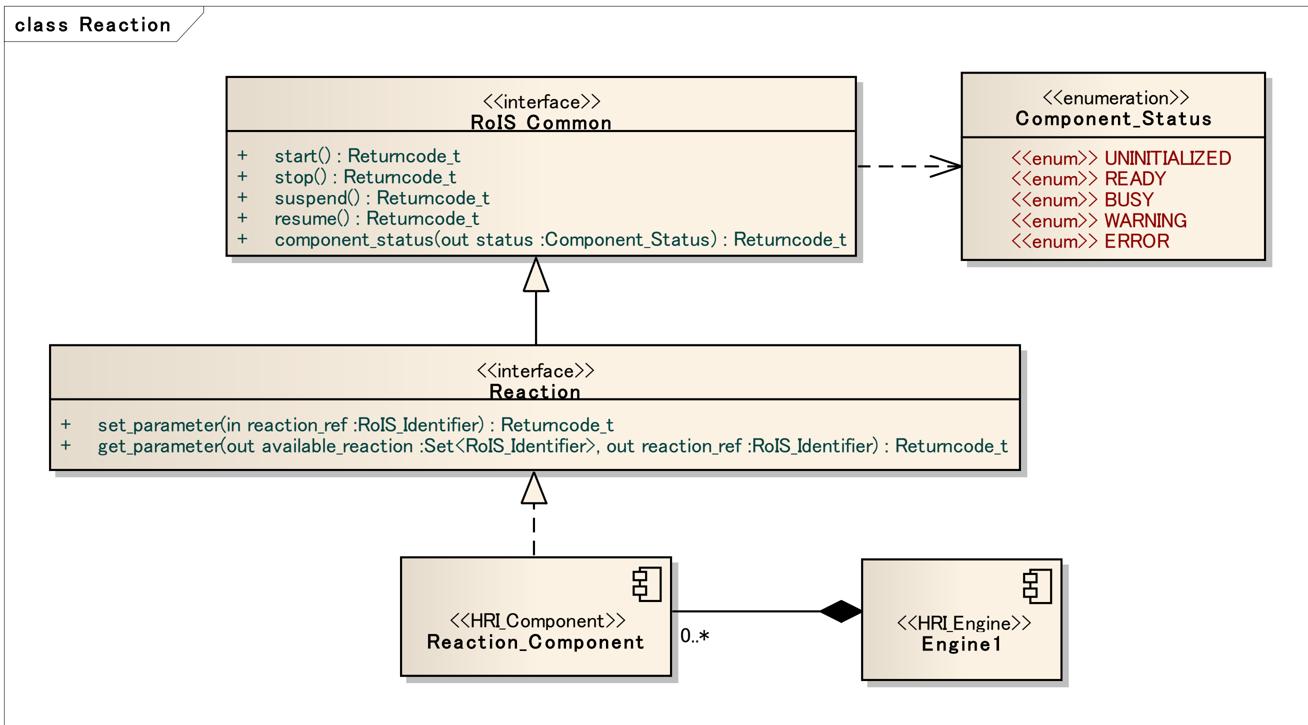

Figure 28: reaction

Table 7.48: reaction

<table>
<tr><td colspan="5">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.</td></tr>
<tr><td colspan="5">Command Method</td></tr>
<tr><td colspan="2">set_parameter</td><td colspan="3">Specifies reaction paramters.</td></tr>
<tr><td>argument</td><td>reaction ref</td><td>RoIS_Identifier</td><td>M</td><td>Reaction type. The type is specifyed by reaction ID.
Reference information related to the ID shall be specified with the each ID.</td></tr>
<tr><td colspan="5">Query Method</td></tr>
<tr><td colspan="2">get_parameter</td><td colspan="3">Obtains reaction paramters.</td></tr>
</table>

result	available_reactions	Set<RoIS_Identifier>	M	Obtains reaction types the robot can execute. The reaction type is expressed as ID reference information.
result	reaction ref	RoIS_Identifier	M	Information about specified reaction type.

7.6.13 Navigation

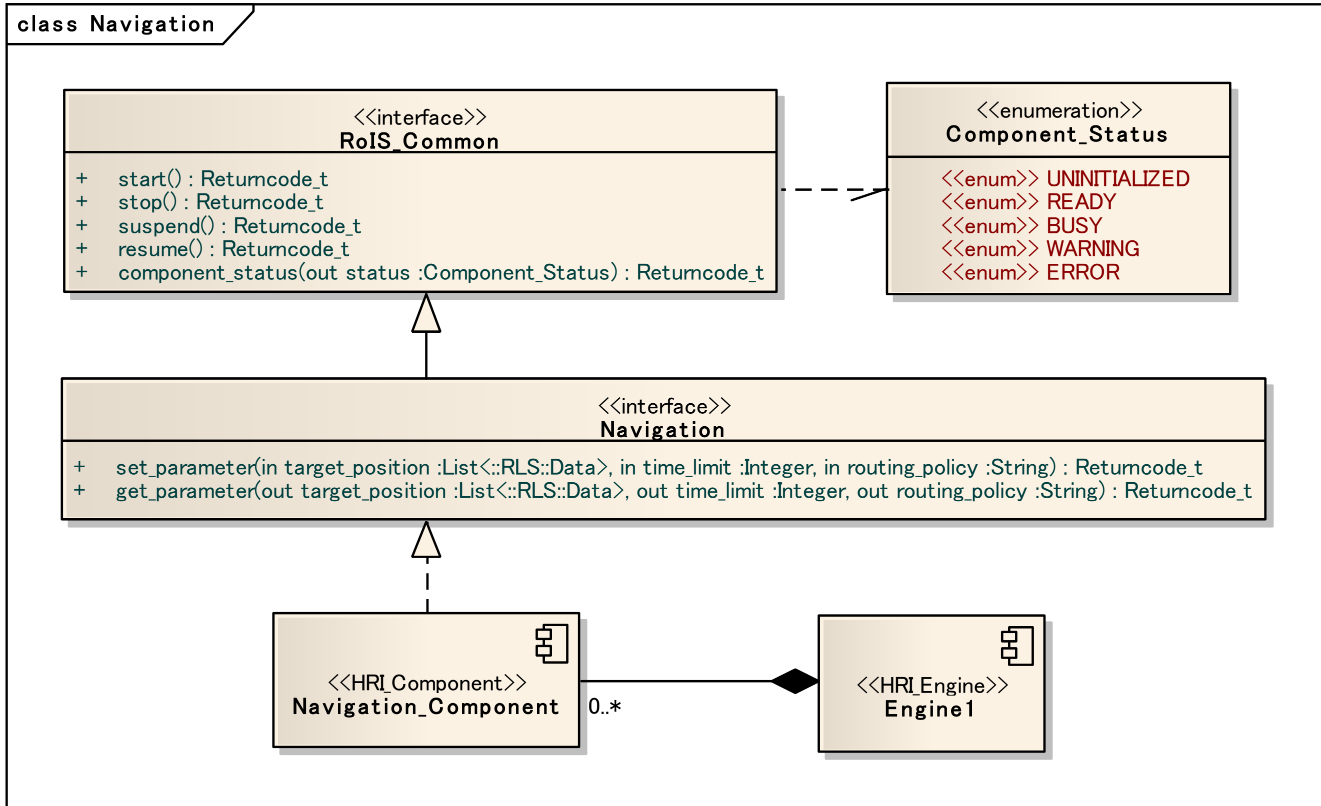

Figure 29: navigation

Table 7.49: 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	
set_parameter	Specifies parameters for navigation.

argument	target_position	List<Data> [RLS]	M	List of target position data. Each data entry may contains ID of the target position. The position data of the target position may be included in this entry, or may be obtained by refering by the ID. This may also be accompanied with additional information such as speed.
argument	time_limit	Integer	O	Time limit for determining whether it is impossible to continue the navigation. The time shall be specified in millisecond.
argument	routing_policy	String	O	Policy for determining the navigation route. For example, there may be the routing policies such as “time priority” or “distance priority”
Query Method				
get_parameter	Obtains parameters for navigation.			
result	target position	List<Data> [RLS]	M	List of specified target position data.
result	time_limit	Integer	O	Time limit for determining whether it is impossible to continue the navigation. The time shall be specified in millisecond.
result	routing_policy	String	O	Policy for determining the navigation route. For example, there may be the routing policies such as “time priority” or “distance priority”

7.6.14 Follow

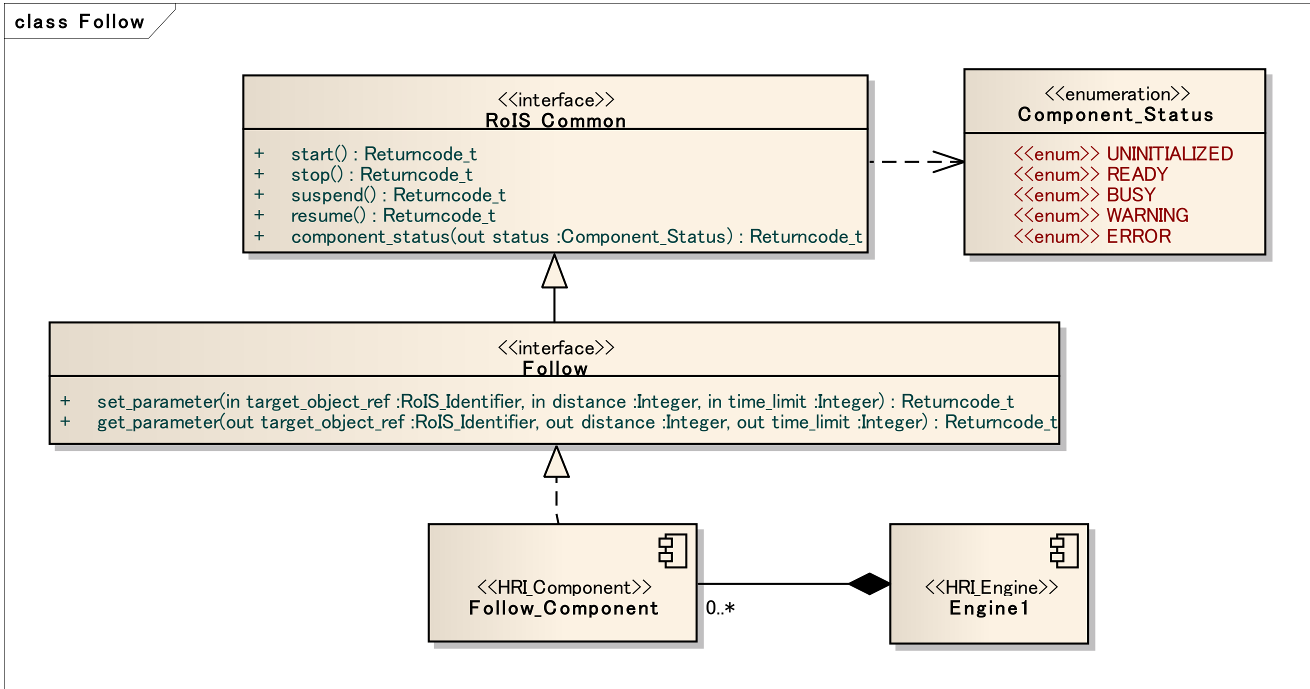

Figure 30: follow

Table 7.50: follow

Description: This is a component for following a specified object. This component acts to follow an object by specifying the ID of the object. An HRI Engine (typically a robot) may include this component when the HRI Engine has the ability to move in the physical world. Follow function is essential for typical robotic services to specify the robot movement for following the target object. This component shall keep following the target until the stop command is requested although the target is not moving.				
Command Method				
set_parameter	Specifies parameters for follow.			
argument	target object ref	RoIS_Identifier	M	Taget object. The object is specified by object IDs. The reference information related to the ID shall be specified with each ID.
argument	distance	Integer	M	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.
argument	time_limit	Integer	O	Time limit for determining whether it is impossible to continue following. If this parameter is not specified, the

				default value may be used. The time shall be specified in milliseconds.
Query Method				
get_parameter	Obtains parameters for follow.			
result	target object ref	RoIS_Identifier	M	Information about the specified target object.
result	distance	Integer	M	Minimum distance between the target and the robot. The distance shall be specified in millimeters.
result	time_limit	Integer	O	Time limit for determining whether it is impossible to continue following. The time shall be specified in milliseconds.

7.6.15 Move

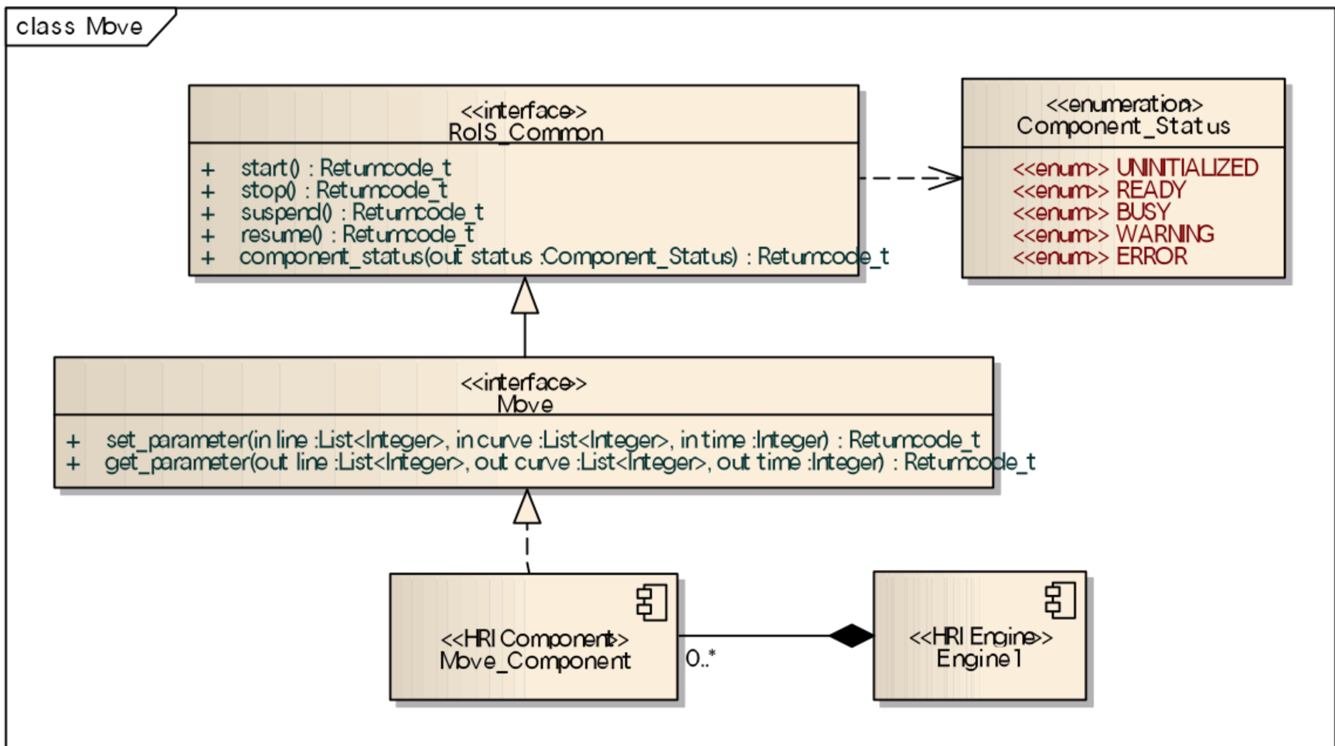

Figure 31: move

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				
set_parameter	Specifies parameters for move.			
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				
get_parameter	Obtains parameters for move.			
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
		);
```

```
};
};

/************************************/
/* RoIS_Common.h */
/************************************/
#include <RoIS_HRI.h>
using namespace RoIS_HRI ;
#include <RoIS_Service.h>
using namespace RoIS_Service ;

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{
};
};

/************************************/
/* RoIS_System_Information.h */
/************************************/
#include <RoIS_Common.h>#include <RLS/Architecture.hpp>
/* http://www.omg.org/spec/RLS/20090601/Architecture.hpp */
namespace 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
	};
};
};
```

```
/*************************************/
/* RoIS_Person_Detection.h    */
/*************************************/
#include <RoIS_Common.h>
namespace Person_Detection
{
class Command : public RoIS_Common::Command{
};
class Query : public RoIS_Common::Query{
};
class Event : public RoIS_Common::Event{
public:
		void person_detected(
				DateTime timestamp,
				Integer number
		);
};
};

/****************************************/
/* RoIS_Person_Localization.h    */
/****************************************/
#include <RoIS_Common.h>
#include <RLS/Architecture.hpp>
/* http://www.omg.org/spec/RLS/20090601/Architecture.hpp */
namespace Person_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 person_localized(
				DateTime timestamp,
				RoIS_IdentifierList person_ref,
				std::vector<RoLo::Architecture::Data> position_data
		);
};
};

/*****************************************/
/* 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_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 <RoIS_Common.h>
#include <RLS/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
		);
};
};

/**********************************/
/* RoIS_Sound_Detection.h    */
/**********************************/
#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
		);
};
};

/************************************/
/* RoIS_Sound_Localization.h    */
/************************************/
#include <RoIS_Common.h>
#include <RLS/Architecture.hpp>
/* 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_localized(
			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(
```

```
			RoIS_Identifier target_object_ref,
			Integer distance,
			Integer time_limit
		);
};
class Query : public RoIS_Common::Query{
public:
		ReturnCode_t get_parameter(
			RoIS_Identifier& target_object_ref,
			Integer& distance,
			Integer& time_limit
		);
};
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{
};
};
```

7.7.2 CORBA PSM

CORBA IDL for this framework is given as follows:

```
/************************************/
/* RoIS_HRI.idl (for HRI Engine) */
/************************************/
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
		);
};
};

/**************************************************/
/* 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
		);
};
};

/************************/
/* RoIS_Common.idl */
/************************/
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{
        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 sequencer<RoLo::Architecture::Data> position_data
```

```
	);
};
};

/****************************************/
/* RoIS_Speech_Recognition.idl    */
/****************************************/
module Speech_Recognition
{
interface Command : RoIS_Common::Command{
	ReturnCode_t set_parameter(
		in sequence<String> languages,
		in String grammer,
		in String rule
	);
};
interface Query : RoIS_Common::Query{
	ReturnCode_t get_parameter(
		out sequence<String> recognizable_languages,
		out sequence<String> languages,
		out String grammer,
		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
	);
};
};

/****************************************/
/* RoIS_Gesture_Recognition.idl    */
/****************************************/
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
	);
};
};
```

```
/************************************/
/* RoIS_Speech_Synthesis.idl    */
/************************************/
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{
};
};

/*************************/
/* RoIS_Reaction.idl    */
/*************************/
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{
};
};

/***************************/
/* RoIS_Navigation.idl    */
/***************************/
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,
				Iin nteger 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 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:sequence>
      <xsd:element name="data_type_ref" type="rois:RoISIdentifierType" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="default_value" type="xsd:string" use="optional"/>
    <xsd:attribute name="description" type="xsd:string" use="optional"/>
  </xsd:complexType>

  <xsd:element name="MessageProfile" type="rois:MessageProfileType"/>
  <xsd:complexType name="MessageProfileType">
    <xsd:sequence>
      <xsd:element name="Results" type="rois:ParameterProfileType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
  </xsd:complexType>

  <xsd:element name="EventMessageProfile" type="rois:EventMessageProfileType" substitutionGroup="rois:MessageProfile"/>
  <xsd:complexType name="EventMessageProfileType">
    <xsd:complexContent>
      <xsd:extension base="rois:MessageProfileType"/>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="QueryMessageProfile" type="rois:QueryMessageProfileType" substitutionGroup="rois:MessageProfile"/>
  <xsd:complexType name="QueryMessageProfileType">
    <xsd:complexContent>
      <xsd:extension base="rois:MessageProfileType"/>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="CommandMessageProfile" type="rois:CommandMessageProfileType" substitutionGroup="rois:MessageProfile"/>
  <xsd:complexType name="CommandMessageProfileType">
    <xsd:complexContent>
      <xsd:extension base="rois:MessageProfileType">
        <xsd:sequence>
          <xsd:element name="Arguments" type="rois:ParameterProfileType" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="timeout" type="xsd:integer" use="optional"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- Data Structure -->
  <xsd:element name="Parameter" type="rois:ParameterType"/>
    <xsd:complexType name="ParameterType">
      <xsd:sequence>
        <xsd:element name="data_type_ref" type="rois:RoISIdentifierType" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="value" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" use="required"/>
    </xsd:complexType>

    <xsd:element name="ArgumentList" type="rois:ArgumentListType"/>
    <xsd:complexType name="ArgumentListType">
      <xsd:sequence>
        <xsd:element name="parameter" type="rois:ParameterType" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
```

```
    </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: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="CommandMessage" type="rois: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:complexContent>
  </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:complexContent>
  </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:ParamerProfile> 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.

```
<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.

```
<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: Results rois:name="operable_time">
```

```
            <rois:data_type_ref rois:code="urn:x-rois:def:DataType:ATR::DateTime"/>
        </rois: Results>
</rois:MessageProfile>
```

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.

```
<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: Results rois:name="number">
                  <rois:data_type_ref rois:code="urn:x-rois:def:DataType:ATR::Integer"/>
            </rois: Results>
      </rois:MessageProfile>
// ===== Parameter =====
      <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>
</rois:HRIComponentProfile>
```

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 messages (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.

```
<rois:HRIEngineProfile>
    <gml:identifier>urn:x-rois:def:HRIEngine:ATR::MainHRI</gml:identifier>
    <gml:name>MainHRI</gml:name>
    <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <rois:HRIComponent>urn:x-rois:def:HRIComponent:ATR::PersonDetection</rois:HRIComponent>
    <rois:HRIComponent>urn:x-rois:def:HRIComponent:ATR::PersonIdentification</rois:HRIComponent>
</rois:HRIEngineProfile>
```

```
<rois:HRIComponentProfile>
    <gml:identifier>urn:x-rois:def:HRIComponent:ATR::PersonDetection</gml:identifier>
    <gml:name>person_detection</gml:name>
    <rois:MessageProfile xsi:type="rois:CommandMessageProfileType" rois:name="start"/>
    .....
</rois:HRIComponentProfile>
```

```
<rois:HRIComponentProfile>
    <gml:identifier>urn:x-rois:def:HRIComponent:ATR::PersonIdentification</gml:identifier>
    <gml:name>person_ identification </gml:name>
    <rois:MessageProfile xsi:type="rois:CommandMessageProfileType" rois:name="start"/>
    .....
</rois:HRIComponentProfile>
```

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.

```
<rois:HRIEngineProfile >
    <gml:identifier>urn:x-rois:def:HRIEngine:ATR::MainHRI</gml:identifier>
    <gml:name>MainHRI</gml:name>
    <rois:SubProfile>
        <gml:identifier>urn:x-rois:def:HRIEngine:ATR::SubHRI01</gml:identifier>
        <gml:name>SubHRI01</gml:name>
        <rois:HRIComponent>urn:x-rois:def:HRIComponent:ATR::PersonDetection</rois:HRIComponent>
        <rois:HRIComponent>urn:x-rois:def:HRIComponent:ATR::PersonIdentification</rois:HRIComponent>
    </rois:SubProfile>
```

```xml
	<rois:SubProfile>
		<gml:identifier>urn:x-rois:def:HRIEngine:ATR::SubHRI02</gml:identifier>
		<gml:name>SubHRI02</gml:name>
		<rois:HRIComponent>urn:x-rois:def:HRIComponent:ATR::PersonDetection</rois:HRIComponent>
		<rois:HRIComponent>urn:x-rois:def:HRIComponent:ATR::PersonIdentification</rois:HRIComponent>
		<rois:HRIComponent>urn:x-rois:def:HRIComponent:ATR::FaceDetection</rois:HRIComponent>
	</rois:SubProfile>
</rois:HRIEngineProfile>
```

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.

```
<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

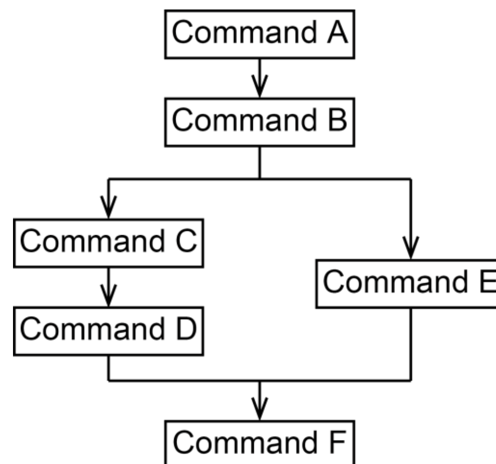

Figure B.1: Structure of CommandUnitSequence example.

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: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: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)

TableC.1: speech_recognition(W3C-SRGS)

Description: Recognize speech input. Here, we assumes 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				
set_parameter		Specifies speech recognition paramters.		
argument	languages	Set<String> [ISO639-1]	O	Specifies languages the speech recognizer will recognize.
argument	position_of_sound	Data [RLS]	O	Specifies direction of sound source the speech recognizer listen to.
argument	grammar	String [W3C-anyURI]	M	Specifies URI of grammar file in W3C-SRGS format.
argument	active_rule	RuleReference [W3C-SRGS]	M	Specifies active rule in the grammar.
Query Method				
get_parameter		Obtains speech recognition paramters.		
result	languages	Set<String> [ISO639-1]	M	Information about languages the recognizer is recognizing.
result	position_of_sound	Data [RLS]	O	Information about direction of sound source the recognizer is listening to.
result	grammar	String [W3C-anyURI]	M	Information about speech recognition grammar.
result	active_rule	RuleReference [W3C-SRGS]	M	Information about active rule in the grammar.
result	recognizable_languages	Set<String> [ISO639-1]	M	Information about languages the recognizer can recognize.
Event Method				
speech_recognized		Notifies speech recognition has completed.		
result	timestamp	DateTime [W3C-DT]	M	Time when the recognition has completed.
result	timestamp_speech_start	DateTime [W3C-DT]	O	Time when the speech input has started.

result	timestamp_speech_end	DateTime [W3C-DT]	O	Time when the speech input has ended.
result	nbest	NbestType	M	Speech recognition result in N-best format.
result	lattice	LatticeType	O	Speech recognition result in lattice format.
result	position_of_sound	Data [RLS]	O	Direction and error distribution of sound source of the recognized speech.
speech_input_started	Notifies the recognizer has detected start of speech input.			
speech_input_finished	Notifies the recognizer has detected end of speech input.			
speech_recognition_started	Notifies the recognizer has started the recognition process.			
speech_recognition_finished	Notifies the recognizer has finished the recognition process.			

Table C.2: NBestType

Description: Data type for speech recognition result in N-best format.				
Derived From: None				
Attributes				
nbest	List<String, String [ISO639-1], Error [RLS]>	M	N ord	Tuple of recognized string, language, certainty.

Table C.3: LatticeType

Description: Data type for speech recognition result in lattice format.				
Derived From: None				
Attributes				
lattice	List<String, String [ISO639-1], RS_Identifier [ISO19115], RS_Identifier [ISO19115], RS_Identifier [ISO19115], Error [RLS]>	M	N ord	Tuple of recognized string, language, id, previous id, next id, certainty.

C.2 Person Gender Identification

Table C.4: person gender identification

Description: This is a component for identifying person gender. This component notifies person gender code of the detected people when the code has been identified. 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.	
Event Method	
person_gender_identified	Notifies gender code of people when the gender has been identified.

result	timestamp	DateTime [W3C-DT]	M	Measurement time.
result	person ref	List<RoIS_Identifier>	M	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.
result	person gender code	List<Integer[ISO5218]>	M	List of detected person gender code.

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				
person_age_recognized	Notifies age of people when the age has been recognized.			
result	timestamp	DateTime [W3C-DT]	M	Measurement time.
result	person ref	List<RoIS_Identifier>	M	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.
result	lower age limit	List<Integer>	M	List of upper limit of recognized age.
result	upper age limit	List<Integer>	M	List of lower limit of recognized age.

Annex D Examples of Data Type (informative)

D.1 Reaction Type

Table D.1: Example of Reaction_Type

Gesture ID	Name	Description
1	nod the head	Move the head downward and return to the original position
2	angle the head	Move the head to the side and return to the original position
3	shake the head	Move the head right and left
4	look right	Turn the head to the right hand side
5	look left	Turn the head to the left hand side
6	look up	Turn the head upward
7	look down	Turn the head downward
8	drop the head	Turn the head obliquely downward
9	bow the head	Turn the head slightly downward
10	shake hands	Shake hands by the right hand and look at the person’s face
11	spread hands slightly	Spread both hands slightly
12	raise hands and spread	Spread both forearms horizontally
13	spread hands	Spread both hands horizontally at shoulders’ height
14	clap hands	Clap hands several times
15	clap hands rhythmically	Clap hands rhythmically
16	point by the right hand	Point to a direction by the right hand, with turning the palm up and stretching the arm
17	point by the left hand	(Same as above, but using the left hand)
18	indicate a monitor display	Turn the head to a monitor display and point to the display by the right hand
19	raise both hands	Move both arms in front of the body and raise them from bottom to top
20	raise both hands from side	Raise both arms from the standing at attention pose to top
21	raise both hands at the shoulder height	Raise both hands from the frontal side to the shoulder height
22	raise a hand straight up (1)	Raise a hand straight up. Wave the hand to catch attention (depends on the implementation)
23	raise a hand straight up (2)	Raise a hand straight up
24	raise the right hand	Raise the right hand
25	raise the left hand	Raise the left hand

26	turn the right palm down	Turn the right palm down slightly with opening the right hand
27	turn the left palm up	(Same as above, but using the left hand)
28	wave the hand	Wave the hand
29	move the fingertip up	Move the thumb-side of the hand in front of the body with the fingertip up and move the hand downward slightly
30	cross arms	Cross arms, making an “X” sign
31	make a circle with arms	Make a circle with arms above the head
32	put both hands on the head	Put both hands on the head
33	put a hand on forehead	Put a hand on forehead
34	salute	Move the right hand to the temple with the arm bent and turning the palm down
35	put a hand to ear	Put a hand to the ear
36	put a hand to mouth	Put a hand to mouth, like shouting. It may use both hands (implementation dependent)
37	make a V sign	Make a “V” sign with a hand
38	strike the chest lightly	Strike the chest lightly with a hand (or a fist)
39	rub the stomach	Move the right hand right and left in front of the stomach
40	put a hand on the waist	Put a hand on the waist with bending the arm
41	put both hands on the waist (1)	Put both hands on the waist with bending arms
42	put both hands on the waist (2)	Put both hands on the waist with bending arms and turning the head slightly up
43	cross arms	Cross both arms in front of the chest
44	swing arms back and forth	Swing both arms back and forth like walking
45	knock	Move a fist back and forth like knocking
46	push by both hands	Raise both hands in front of the chest and move them ahead like pushing
47	indicate a height by a hand	Put a hand at a certain height with turning the palm down
48	bend an arm	Move a hand to the shoulder with bending the arm slowly
49	put an arm on a shoulder	Put an arm on someone’s shoulder
50	glance at a wristwatch	Glance at the left wrist