Date: August 2025

OBJECT MANAGEMENT GROUP®

Robotic Interaction Service (RoIS) Framework

Version 2.0-beta2

OMG Document Number: dtc/2025-09-22

Normative reference: https://www.omg.org/spec/RoIS/2.0/

Machine readable file(s): https://www.omg.org/RoIS/20250801

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.

Table of Contents

Preface

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 Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (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. All OMG Specifications are available from the OMG website at:

https://www.omg.org/spec

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
9C Medway Road, PMB 274
Milford, MA 01757
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 https://www.iso.org

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.

The new 3-layered structure of RoIS 2.0 should be described here.

- Definition of the functional components using RoSO and RoIS ontologies.
- The lower surface of the messaging layer is to be removed from the messaging section and PSMs.
- Add a functional component that provides streaming functions for cybernetic avatar services.

2 Conformance

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

- Implementations shall provide interfaces described in "Section 8.2 RoIS Interface."
- Data structure of messages treated by implementations shall support the profile described in "Section 8.3 RoIS Profiles."
- Implementations shall support the common messages described in "Section 8.4 Common Messages." This does not mean that the module shall include every common message described herein. However, every module should support the common messages when the module uses the basic components listed in "8.4 Common Messages."
- Implementations shall handle component profiles described in XML files (XML-Profile PSM machine readable files) and messages defined in the component profile.

3 Normative References

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

[Commons] Object Management Group, Commons Ontology Library, Version 1.1, 2024. Available at https://www.omg.org/spec/Commons/ .

[CORBA] Object Management Group, Common Object Request Broker Architecture (CORBA), Version 3.4, 2021. Available at https://www.omg.org/spec/CORBA/ .

[DDS] Object Management Group, Data Distribution Services (DDS), Version 1.4, 2015. Available at https://www.omg.org/spec/DDS/ .

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

[ISO8601] International Organization for Standardization, Data elements and interchange formats –Information interchange- Representation of dates and times.

[ISO14882] International Organization for Standardization, Programming Language C++, 2020.

[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.

[OWL 2] OWL 2 Web Ontology Language Quick Reference Guide (Second Edition), W3C Recommendation 11 December 2012. Available at http://www.w3.org/TR/2012/RECowl2-quick-reference-20121211/.

[RDF] RDF 1.1 Concepts and Abstract Syntax. Richard Cyganiak, David Wood and Markus Lanthaler, Editors. W3C Recommendation, 25 February 2014. Available at http://www.w3.org/TR/rdf11-concepts/.

[RDF Schema] RDF Schema 1.1. Dan Brickley and R.V. Guha, Editors. W3C Recommendation, 25 February 2014. Available at http://www.w3.org/TR/rdf-schema/.

[RDF Turtle] RDF 1.1 Turtle. Eric Prud'hommeaux, Gavin Carothers, Editors. W3C Recommendation 25 February 2014. Available at http://www.w3.org/TR/turtle/ .

[RLS] Object Management Group, Robotic Localization Service (RLS), Version 1.1, 2012. Available at https://www.omg.org/spec/RLS/ .

[RTC] Object Management Group, Robotic Technology Component (RTC), Version 1.1, 2012. Available at https://www.omg.org/spec/RTC/ .

[UML] Object Management Group, OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.5.1, 2017. Available at https://www.omg.org/spec/UML/ .

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

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

4 Terms and Definitions

For the purposes of this specification, the following terms and definitions apply.

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 Additional Information

6.1 Acknowledgements

The following companies submitted this specification:

- Japan Robot Association (JARA)
- Korea Association of Robot Industry (KAR)

The following additional organizations contributed to this specification:

- National Institute of Advanced Industrial Science and Technology (AIST)
- Shibaura Institute of Technology
- Meijo University

The following additional companies and organizations are supporters of this specification:

- Advance Telecommunications Research Institute International (ATR)
- Electronics and Telecommunications Research Institute (ETRI)
- Future Robot Co., Ltd.
- Hitachi, Ltd.
- Ministry of Internal Affairs and Communications, Japan
- New Energy and Industrial Technology Development Organization (NEDO)
- Nippon Telegraph and Telephone
- University of Seoul

- University of Tokyo
- University of Tsukuba

6.2 Intellectual Property Right

The submission team, including all submitters and contributors listed above (i) grants to the Object Management Group, Inc. (OMG) a perpetual, nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version, and to allow other persons to do so; and (ii) agrees that no person shall be deemed to have infringed our copyright interest in the Submission by using OMG specifications or conforming any software to OMG specifications.

As specified in the RFP, the IPR mode for this submission is the RF on Limited Terms per Appendix A of the OMG IPR Policy [IPR].

6.3 Reuse of the Ontologies

The Robotic Interaction Service (RoIS) Framework 2.0 uses and extends the Robotic Service Ontology (RoSO). RoSO uses and extends a number of the ontologies specified in the companion Commons Ontology Library specification. The Commons Ontology Library contains small but fundamental building block ontologies that are essential to RoSO. RoSO also uses ontologies included in the Languages, Countries, and Codes (LCC) specification for the identification of languages and geographic regions associated with vocabulary elements.

6.4 Notations

The notation used to represent description logic expressions (i.e., the expressions in the Parent columns in class tables containing ontology details) is consistent with the notation defined in the Description Logic Handbook [DL Handbook]. The notation used in this specification, representing a subset of OWL 2, is described in Table 6.1, below.

Table 6.1: Description Logic Expressions Notation

Construct	*Description*	*Notation*
Boolean Connectives and Enumeration		
intersection	The intersection of two classes consists of exactly those individuals which are instances of both classes.	C ∩ D
union	The union of two classes contains every individual which is contained in at least one of these classes.	C ∪ D
enumeration	An enumeration defines a class by enumerating all its instances.	oneOf (i_1, i_2, i_3, … i_n)
Property Restrictions		
universal quantification	Universal quantification is used to specify a class of individuals for which all related individuals must be instances of a given class (*i.e.*, allValuesFrom in OWL).	∀R.C, where R is the relation (property) and C is the class that constrains all values for related individuals
existential quantification	Existential quantification is used to specify a class as the set of all individuals that are connected via a particular property to at least one individual which is an instance of a certain class (*i.e.*, someValuesFrom in OWL).	∃R.C, where R is the relation (property) and C is the class that constrains some values of related individuals

individual value	Individual value restrictions are used to specify classes of individuals that are related to one particular individual (*i.e.*, hasValue in OWL).	∀R.I, where R is the relation (property) and I is the individual
exact cardinality	Cardinality (number) restrictions specify classes by restricting the cardinality on the sets of fillers for roles (relationships, or properties in OWL). Exact cardinality restrictions restrict the cardinality of possible fillers to exactly the number specified.	= n R (for unqualified restrictions) = n R.C (for qualified restrictions, i.e., including onClass or on DataRange)
maximum cardinality	Maximum cardinality restrictions restrict the cardinality of possible fillers to at most the number specified (inclusive).	≤ n R (for unqualified restrictions) ≤ n R.C (for qualified restrictions)
minimum cardinality	Minimum cardinality restrictions restrict the cardinality of possible fillers to at least the number specified (inclusive).	≥ n R (for unqualified restrictions) ≥ n R.C (for qualified restrictions)
Class Axioms		
equivalent classes	Two classes are considered equivalent if they contain exactly the same individuals.	≡ C
disjoint classes	Disjointness means that membership in one class specifically excludes membership in another.	¬ C
Property Axioms		
complex role inclusions	Role inclusions allow [object] properties to be chained together in a sequence that is a subproperty of a higher-level property.	R ○ R

7 RoIS Framework

7.1 Overview of RoIS Framework

In this RoIS [Robotic Interaction Service Framework] specification, we define "Robotic Service" as "the service provided by service robot system [hereafter "service-robot"] which includes multiple robots and/or distributed robotic components" to human in the shared space. Many service-robot applications prepare robot scenarios like the one shown in Figure 7.1. 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 7.1 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."

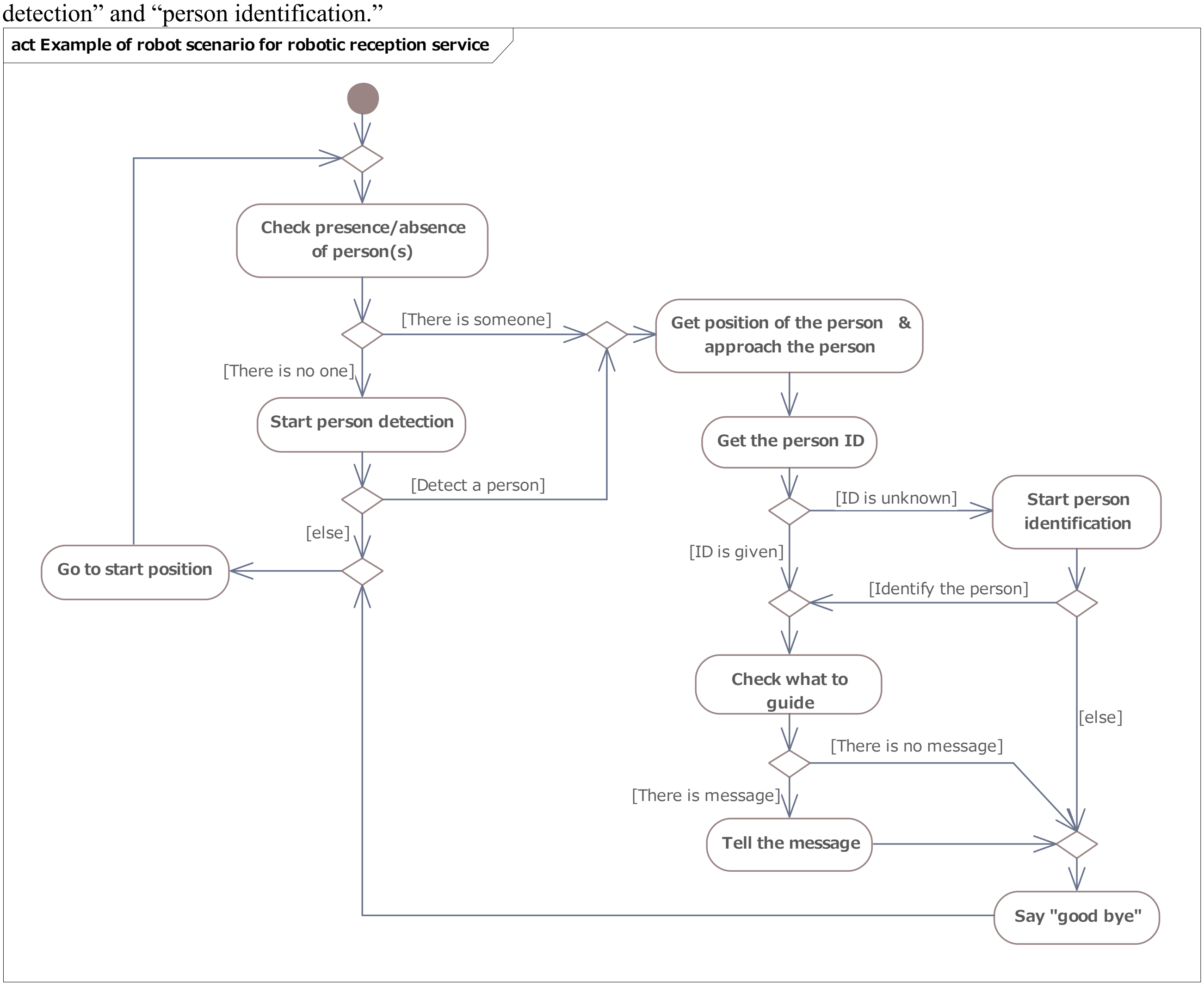


Figure 7.1: Example of robot scenario for robotic reception service. Events delivered from sensors, actuators or other event sources, such as an internal timer, to a service application trigger each state transition and the application controls the robot according to the scenario.

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 7.2 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.

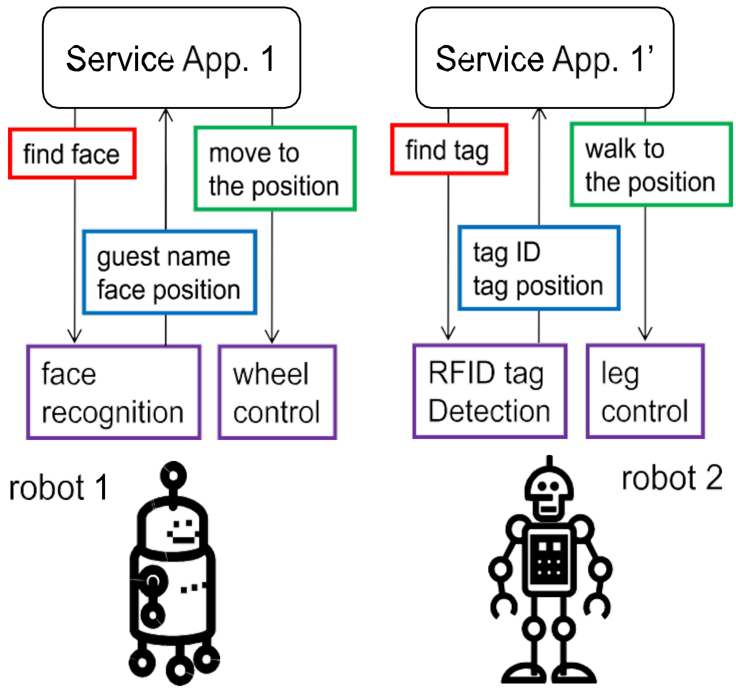


Figure 7.2: Conventional style of service application programming. Service application programmer must write service application programs for eachrobot independently because functions provided by each robot are different.

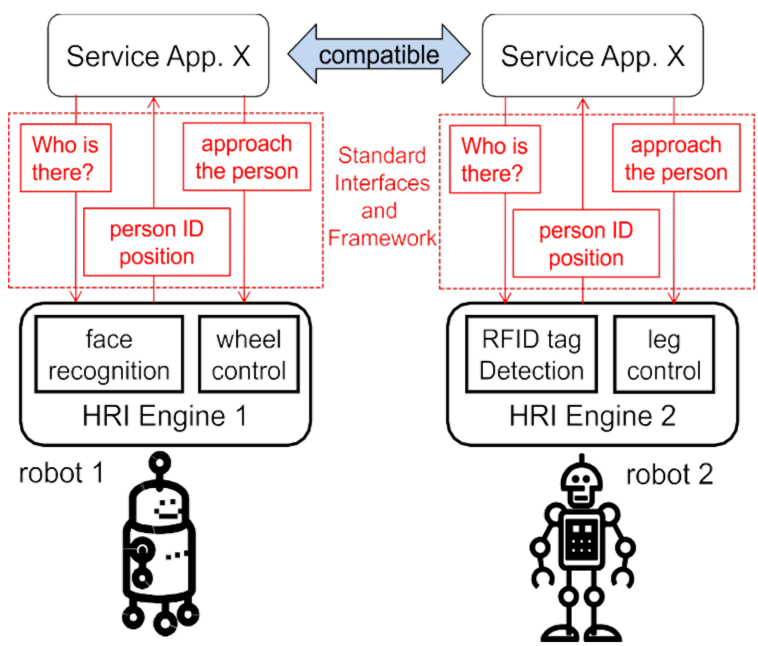


Figure 7.3: RoIS service application programming style. The same service application program works on different robot platforms with little modification.

7.2 Structure of RoIS Framework

The cases on HRI service as a part of RoIS service are described above. However, service robot is a robot which provides "robotic services" interacting with humans in the shared environment; here the "service" includes physical actions or information which work on the surroundings. To describe such service processes, or to exchange description of services, provided by service robots, it can have much wider interaction with the surroundings. The Robotic Interaction Service (RoIS) Framework abstracts the hardware in the service robot (sensors and actuators) and the functions provided by the robots and /or components, and provides a uniform interface between service robots and applications.

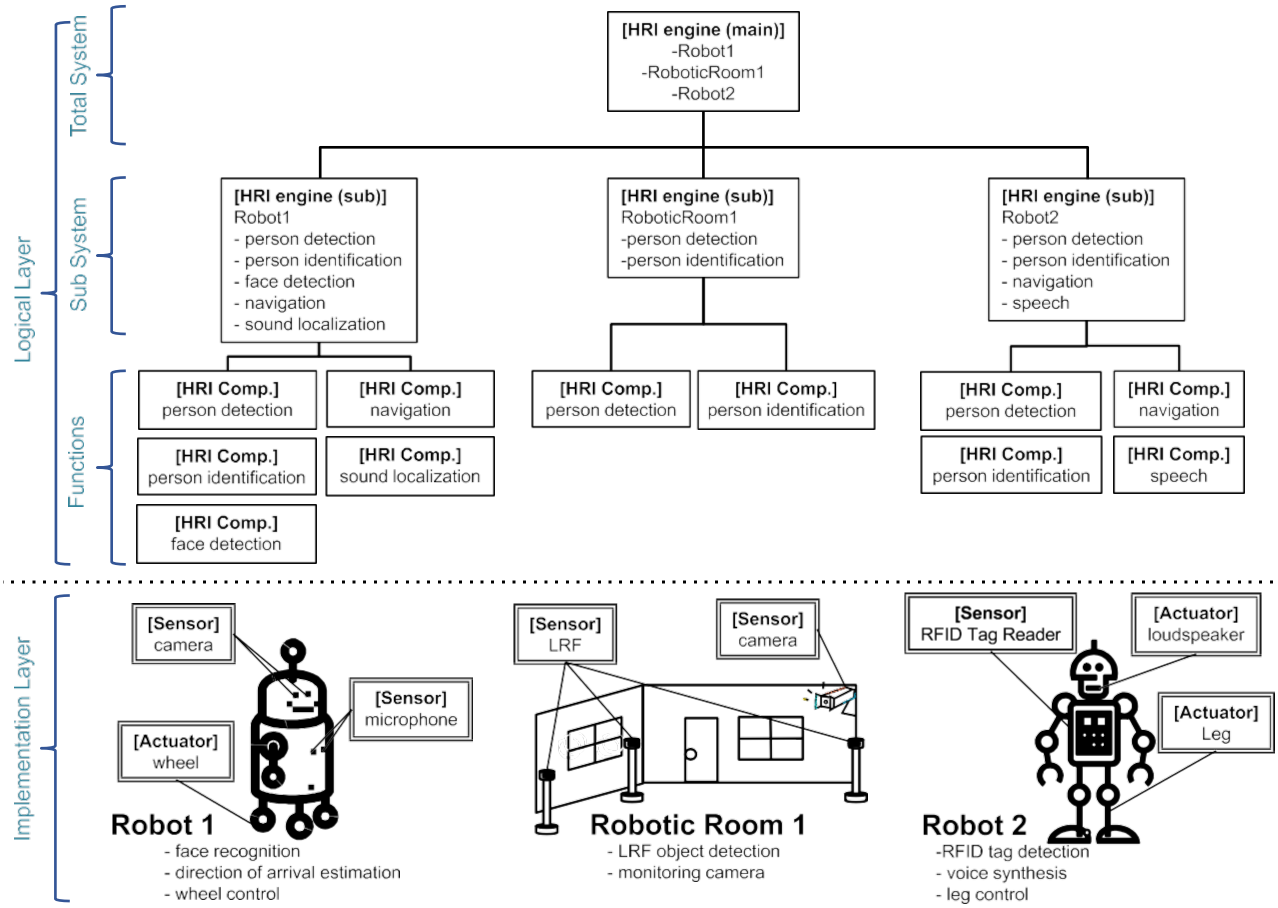


Figure 7.4: Example of HRI Engine and HRI Components

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

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

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

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

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

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

7.3 RoIS Communication Framework

7.3.1 RoIS Messaging Framework

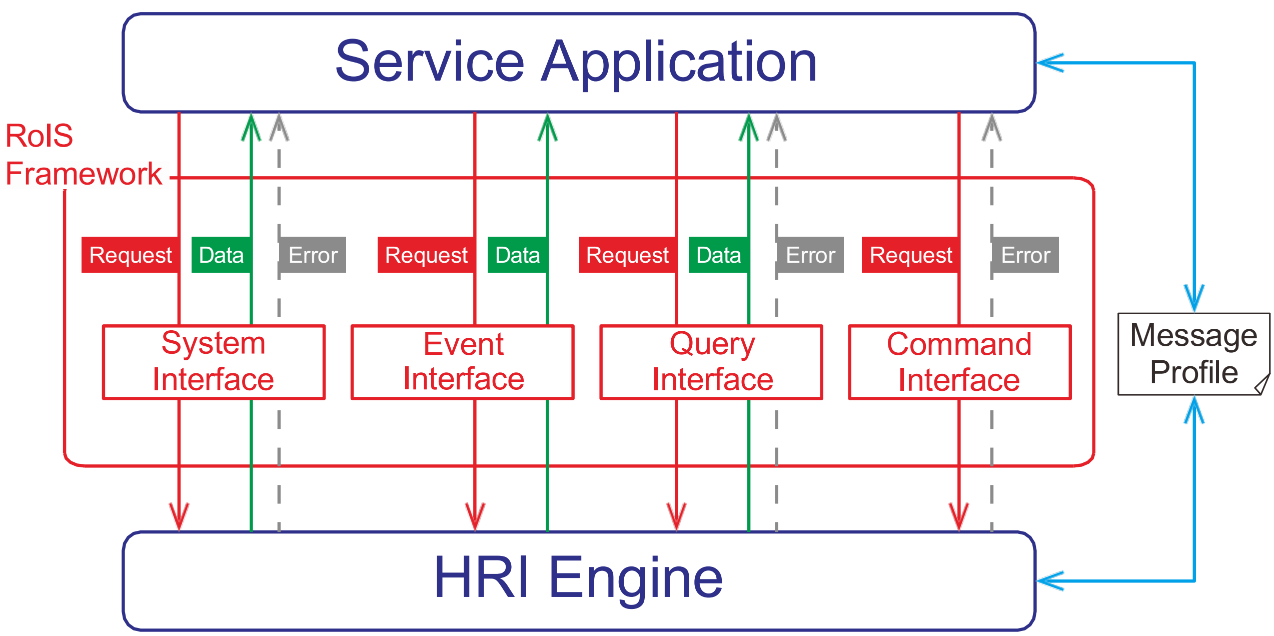


Figure 7.5: Schematic diagram of RoIS Framework and its message flows. In the framework, Service Application communicates with HRI Engine by some messages through System, Event, Query and Command Interfaces.

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 7.5).

- **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 8.3 RoIS Profiles.

7.3.2 Use of Other Component Models

RoIS messaging framework defines messages exchanged in the RoIS protocol but does not define detailed transfer method of those messages. C++ PSM and CORBA PSM are defined, but they only define method signatures of components' functions to be exposed by components and messaging functions to be called from those components.

RoIS messages can be exchanged upon any messaging protocols or remote procedure calls such as OMG CORBA, OMG RTC or ROS (or ROS2 upon OMG DDS). This specification does not intend to define implementation details of message exchange but does allow making use of any transport protocols. Interoperability among nodes is to be discussed only within the same transport protocol. Annex F.2.3 provides an example implementation of message transport for cybernetic avatar services developed upon web socket.

7.3.3 Use of Streaming Channels

Robotic services that provided with semi-autonomous service robots require participation of human operators. As semi-autonomous service robots including avatar robots are supposed to be operated remotely by human operators, those robots and/or execution environment are expected to transmit information of the surroundings to the operators. Video and audio streams are expected to be used in addition to components with sensing functions. Though the RoIS messaging framework and the definitions of functional components' messages do not cover the details of streaming formats, this specification defines how to manage and control such streams as components' functions in 8.4.18 Audio Streaming and 8.4.19 Video Streaming and provides an example implementation of such components and streaming platform in Annex F.2.3.

7.4 RoIS Functional Components

7.4.1 RoIS HRI Profiles

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.

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.

Details of each profile are described in 8.3 RoIS Profiles.

7.4.2 Ontology for RoIS Functional Components

Several ontologies have been developed to describe things and events in the world and make it possible to process reasoning on those things and events. In the field of robotic services, if the specification and requirements of robotic functional components are described with ontology, service developers and service execution environment reasoning for service execution.

RoSO provides ontologies for robotic services including basic vocabularies to describe Human-Robot or Human-Agent Interaction (HRI/HAI) to be defined upon abstract ontologies (not concretely specified but depicted as "High-level Ontology" in Figure 1 and 2). The vocabularies constitute robotic services, vocabularies to describe functions and constraints of robotic functional components for deployment in robotic services; and vocabularies to describe functions and requirements of higher-level robotic services.
Those vocabularies are, partly from a physical viewpoint, classified under Agents, Services, Functions, and Environments. The ontology also incorporates common ontologies from OMG Commons Library.

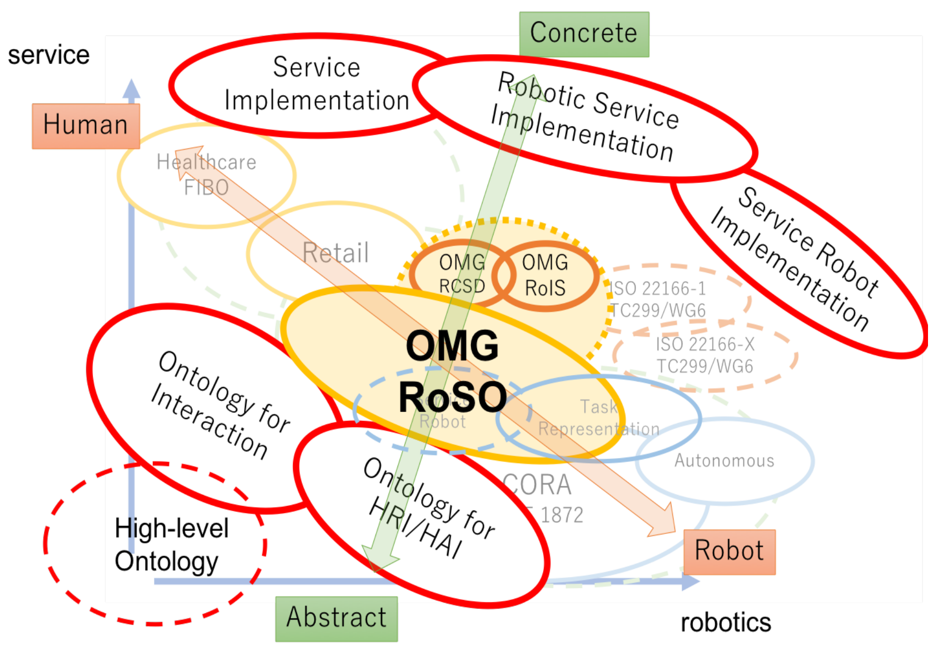


Figure 7.6 How RoSO supports and is supported by other standards. (Figure 2 in RoSO 1.0 beta)

As RoSO defines fundamental ontology for robotic services, vocabularies specific for component functions and their parameters are defined as an extension for RoSO in Section 8.4.2.

8 Platform Independent Model

8.1 Format and Conventions

8.1.1 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 8.2.1 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

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.

8.1.2 Enumeration

Enumerations are documented as follows:

Table x.x: <enumeration name>

<constant name>	<description>
…	…

8.1.3 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>
…	…	…	…	…

8.1.4 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 8.2 RoIS Interface.

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 8.17 and Table 8.16, respectively.

8.2 RoIS Interface

8.2.1 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 8.1: ReturnCode_t enumeration

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

8.2.2 Interaction

8.2.2.1 System Interface

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

8.2.2.1.1 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 8.1.

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.

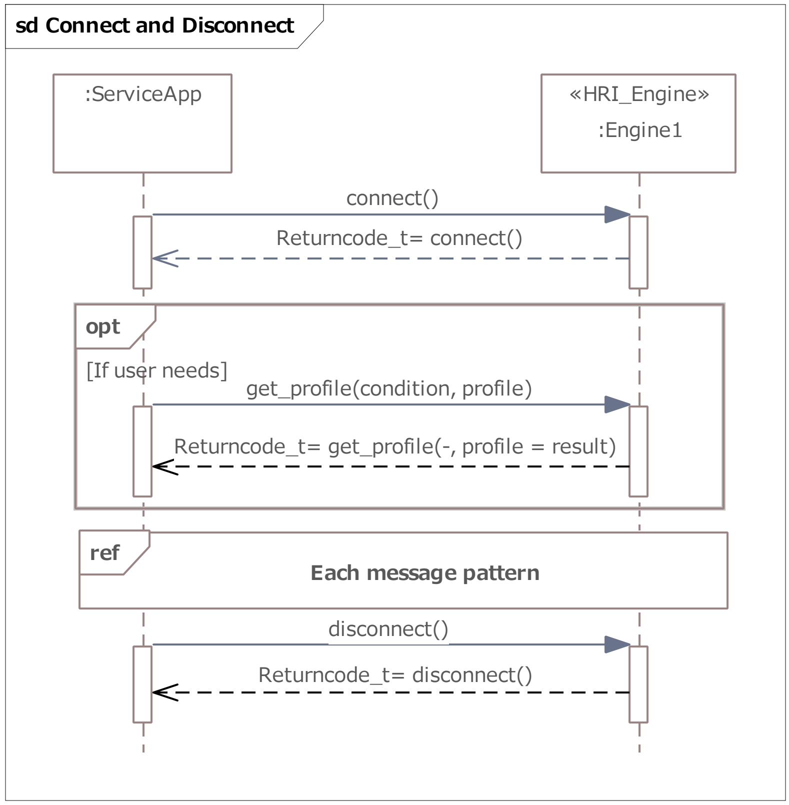


Figure 8.1: Sequence Diagram of System Interface (Connect / Disconnect)

8.2.2.1.2 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 8.2.

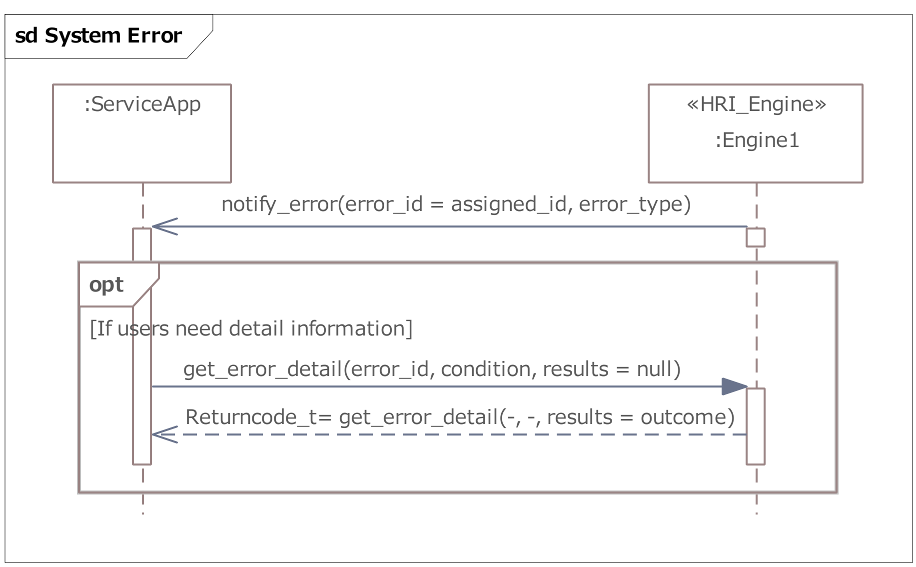


Figure 8.2 Sequence Diagram of System Interface (System Error)

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().

8.2.2.2 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 8.3.

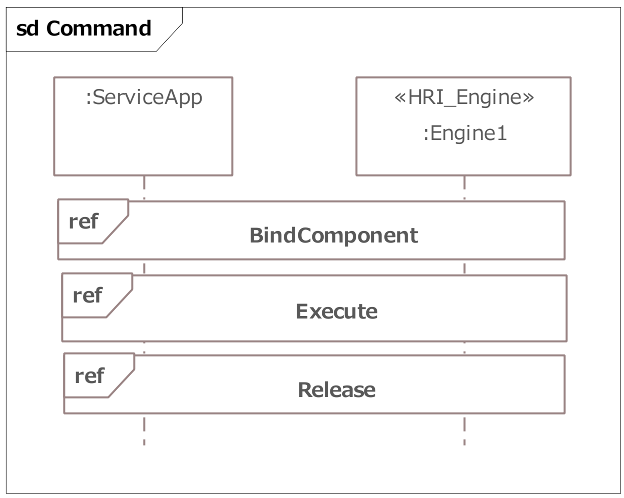


Figure 8.3: Sequence Diagram of Command Interface

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.

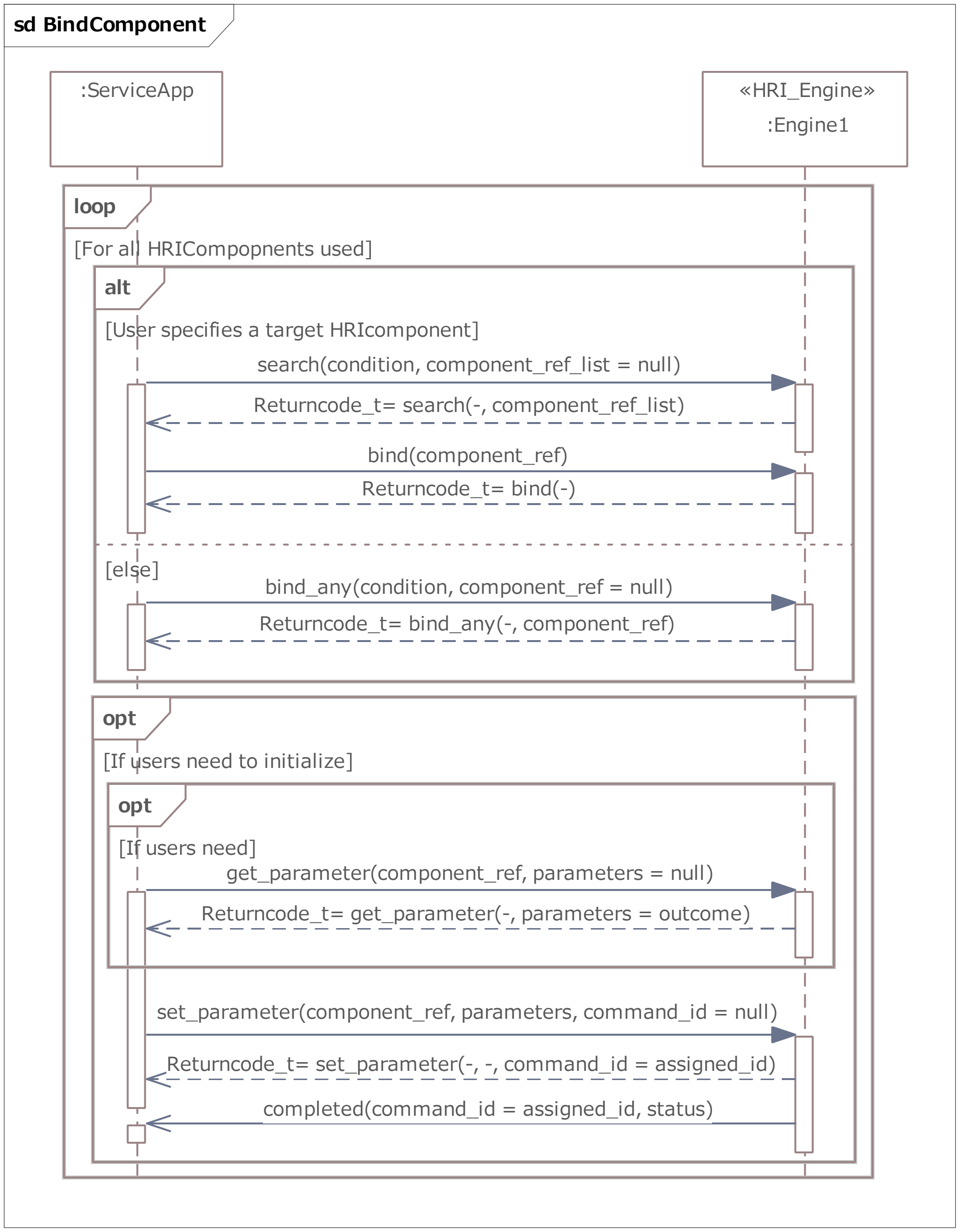


Figure 8.4: Sequence Diagram of "BindComponent" in Command Interface

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_refs"), 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.

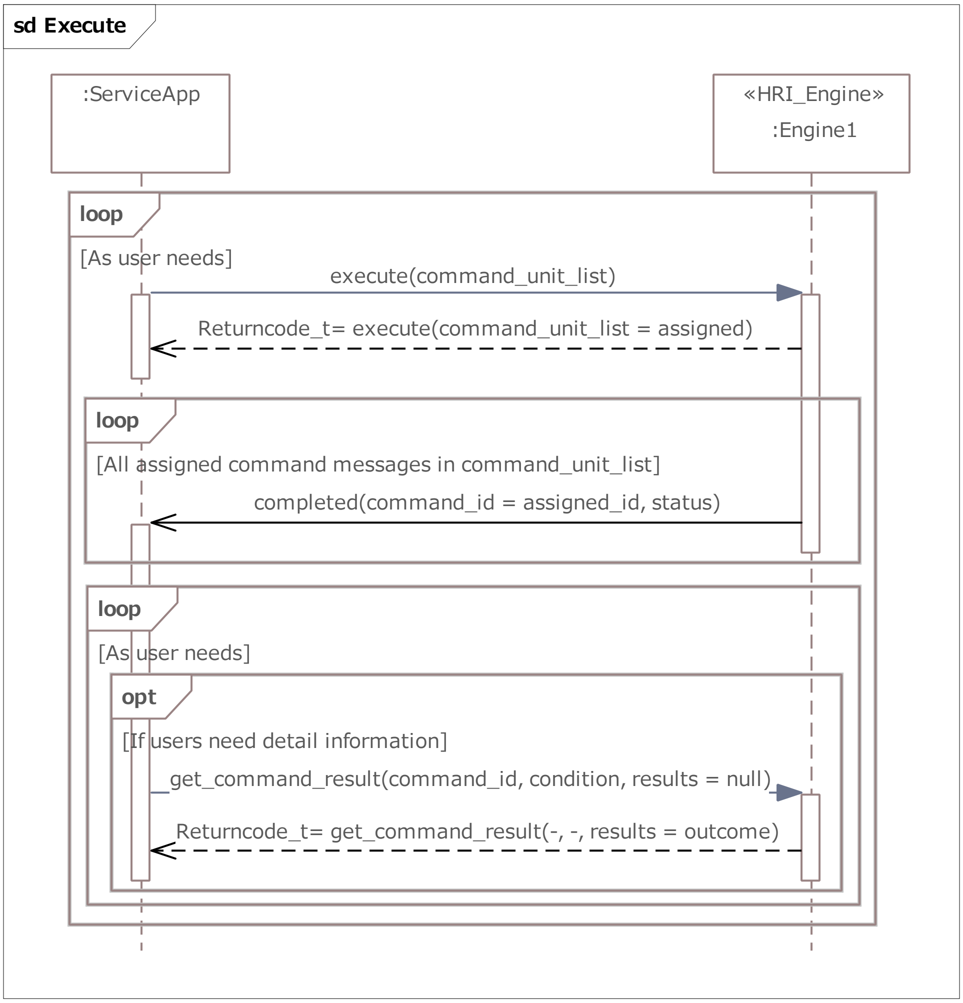


Figure 8.5: Sequence Diagram of "Execute" in Command Interface

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 8.2.4.1 Command Message.

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().

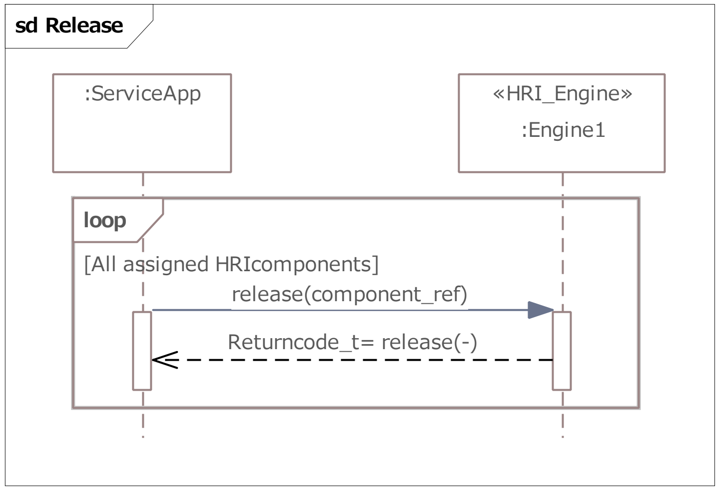


Figure 8.6: Sequence Diagram of "Release" in Command Interface

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 follow the execution status of each command message that it issues.

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

8.2.2.3 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 8.7.

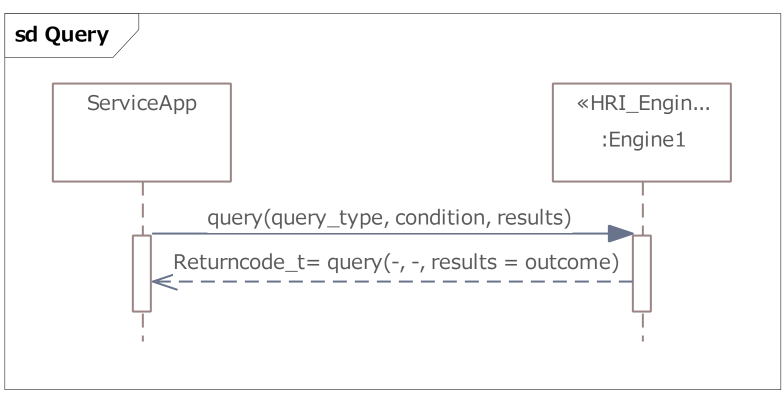


Figure 8.7: Sequence Diagram of Query Interface

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.

8.2.2.4 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 8.8.

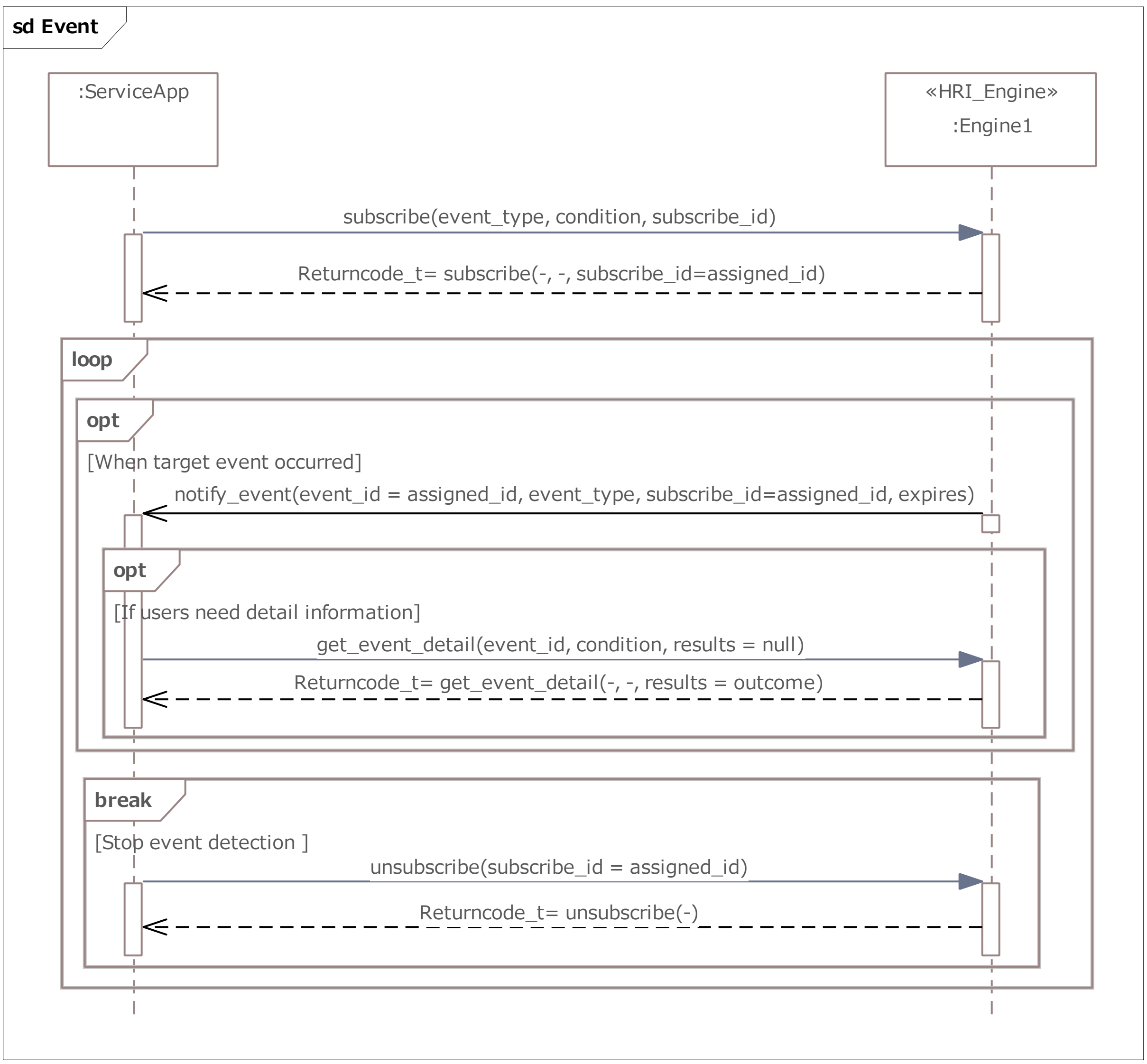


Figure 8.8: Sequence Diagram of Event Interface

8.2.2.4.1 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.

8.2.2.4.2 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.

8.2.2.5 Streaming Interface

The Streaming Interface enables the service application to receive a data stream from and transmit it to the HRI engine. It mainly intends to send audio and video streams between service operators and clients facing the HRI engine. As streaming data are expected to be generated continuously and more frequently than sensor events, it is used when the use of Event Interface is unsuitable. The interface defines several Commands and Events to control the streams but does not define the details of transport and encoding. The sequence diagram of the entire Streaming Interface is shown in Figure 8.9.

The service application initiates a streaming connection by a connect_stream() command. Before connecting the stream, parameters for encoding and transport shall be set by a set_parameter() command. The service can obtain available parameters by a get_parameter() query to find suitable parameters. Both the HRI Engine and the service application then establish a streaming connection, but the details of the connection are out of the scope of RoIS. RoIS only handles the management of streams between the service application and the HRI Engine. The stream is closed when the service application sends a disconnect_stream() command.

The service application can suspend the receiving stream by sending a suspend_stream() command. It can also resume the suspended stream by sending a resume_stream() command. On the other hand, the HRI Engine can send a notify_stream_status() event when it needs to suspend, resume, or close the stream. When the service application receives such events, it shall handle the event and control the stream appropriately. The service application can also query the status of the stream at the HRI Engine.

Though the streaming interface allows the handling of a two-way streaming connection, it does not define methods and events to control the endpoint at the service application from the HRI engine. The service application can suspend and resume the sending stream without any notification and the HRI Engine cannot request suspend and resume the receiving streams.

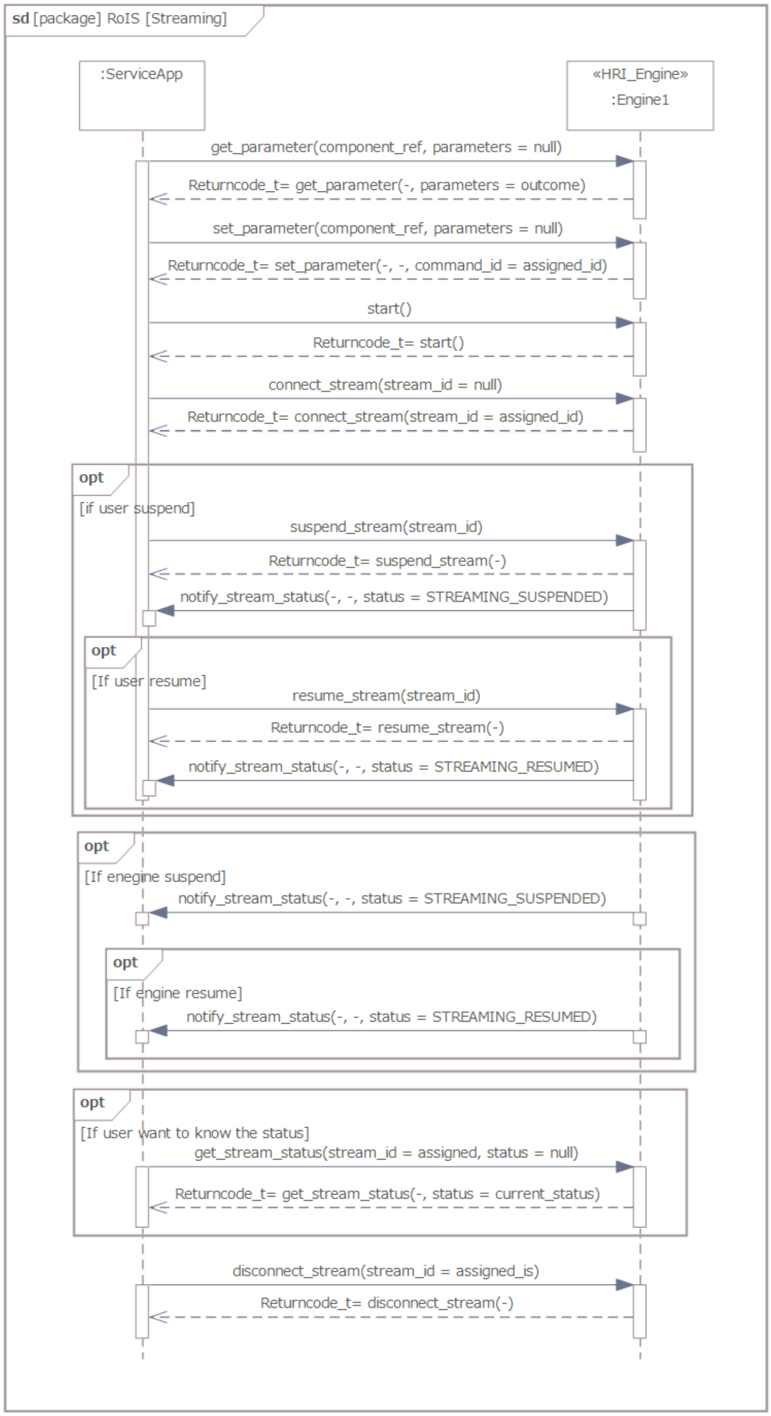


Figure 8.9: Sequence Diagram of Streaming Interface

8.2.3 Interfaces

The overall configuration of the interfaces in the RoIS Framework is shown in Figure 8.10.

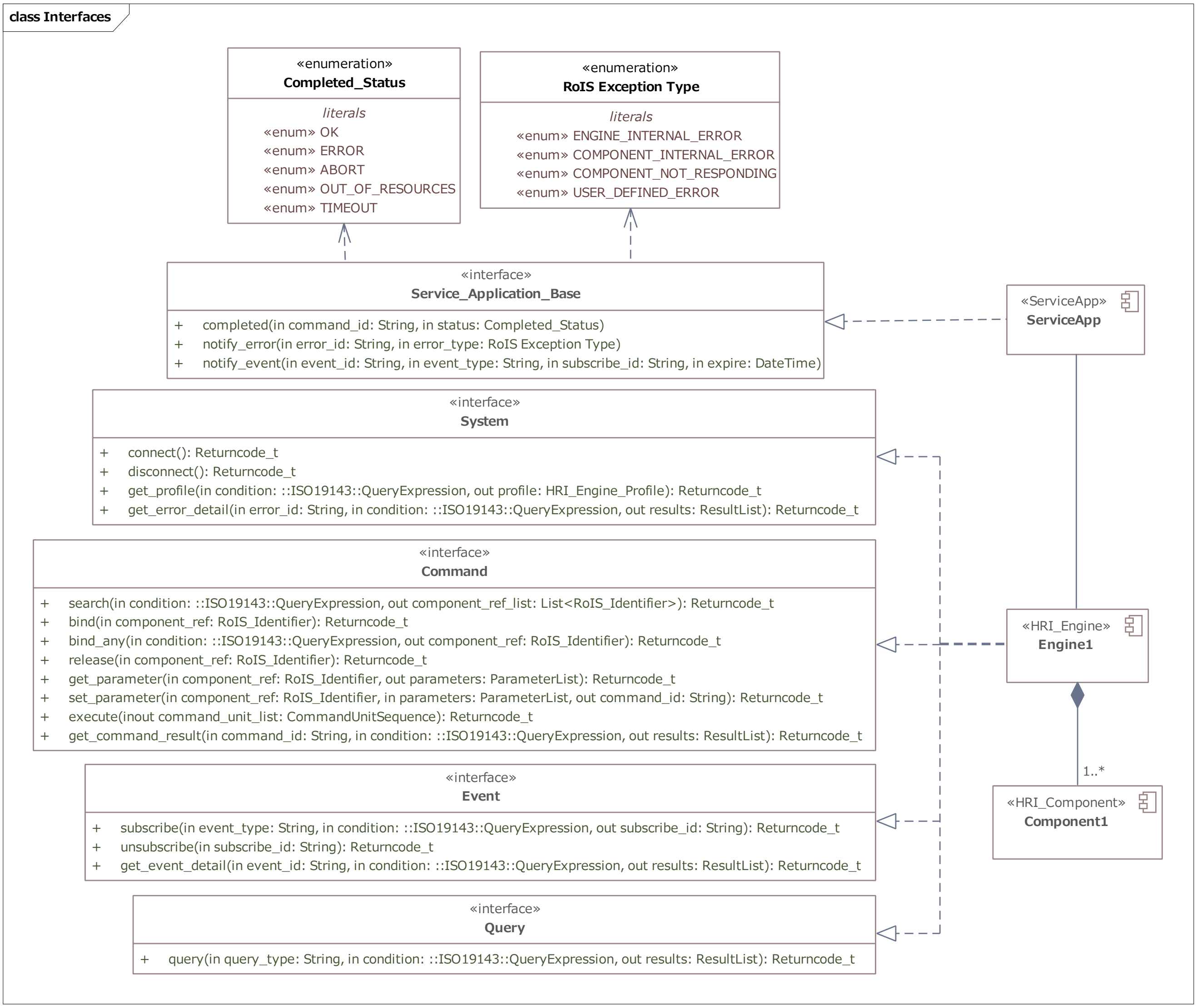


Figure 8.10: RoIS Interfaces

8.2.3.1 Interfaces for HRI Engine

The interfaces for the HRI Engine are defined in Table 8.2 to Table 8.6. The streaming interface is not defined in Figure 8.10 since the interface is defined upon existing interfaces.

Table 8.2: 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 8.3: 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 8.4: 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 8.5: 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.

Table 8.6: Streaming Interface

Description: The interface required to enable the HRI Engine to receive stream-related requests from the Service Application.			
Derived From: None			
Operations			
connect_stream	Connect to a new stream of the component bound (as well as Event subscription). The message is defined as a Command Message.		
out	stream_id	String	A stream ID is assigned by the HRI Engine for each established connection.
disconnect_stream	Disconnect a stream specified by a stream ID. (as well as canceling an Event subscription). The message is defined as a Command Message.		
in	stream_id	String	Specifies the ID of the stream to disconnect.

query_stream_status	Obtains stream status of HRI Engine, in addition to the event notification from the HRI Engine to Services. The message is defined as a Query Message.		
in	stream_id	String	Specifies the ID of the stream.
out	status	StreamStatus	Status of the stream returned from the HRI Engine.
suspend_stream	Requests to suspend the stream. The message is defined as a Command Message.		
in	stream_id	String	Specifies the ID of the stream.
resume_stream	Requests to resume the (suspended) stream. The message is defined as a Command Message.		
in	stream_id	String	Specifies the ID of the stream.
notify_stream_status	Notifies stream event from the HRI Engine to services as an Event notification. The service also needs to subscribe/unscribe events using Event interface.		
in	event_id	String	Holds the ID of the event notification assigned when sending the event message.
in	event_type	StreamStatus	Status of the stream changed at the HRI Engine.

8.2.3.2 Interfaces for Service Application

The interface provided on the service-application side is defined in Table 8.7.

Table 8.7: 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 [ISO8601]	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	ExceptionType	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.

ExceptionType and *Completed_Status* are defined in Table 8.8 and Table 8.9.

Table 8.8: 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 8.9: 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.	

Stream_Status is defined in Table 8.10.

Table 8.10: Stream_Status enumeration

STREAMING_NOT_RUNNING	The stream is connected but not yet started, or it is closed.
STREAMING_NOT_CONNECTED	The stream is not connected.
STREAMING_RUNNING	The stream is running.
STREAMING_SUSPENDED	The stream is suspended (as an event) or suspending (as a status).
STREAMING_RESUMED	The stream is resumed (as an event).
Note: Corresponding situations of these statuses shall be returned for queries or notified as events.	

8.2.4 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.

8.2.4.1 Command Message

The data structure of the command message exchanged by execute() is shown in Figure 8.11.

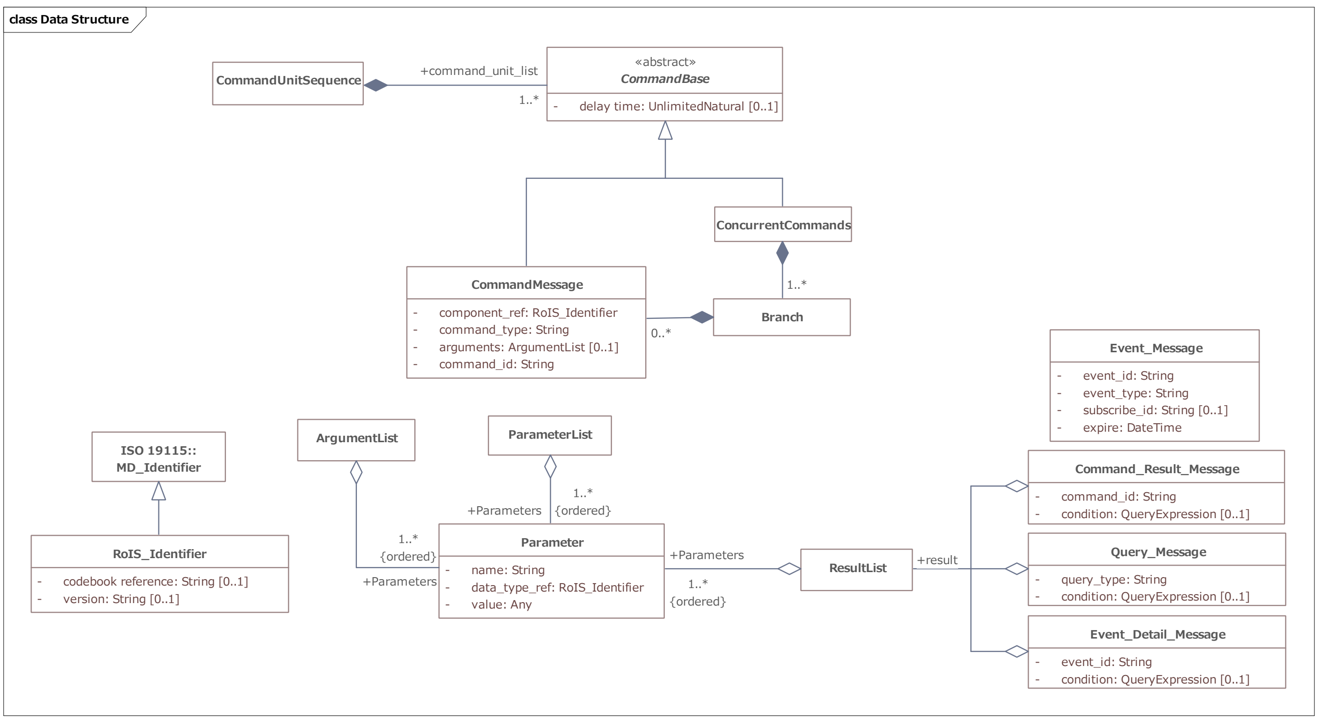


Figure 8.11: Data Structure of Command Message.

RoIS_Identifier is defined for describing an ID with the reference codebook for the ID. The detail of this data type is depicted in Table 8.11.

Table 8.11: 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 8.12 to Table 8.20.

Table 8.12: CommandUnitSequence class

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

Table 8.13: 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 8.14 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 8.15: 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 8.16: Branch class

Description: A concrete data class for specifying a combination of commands to the HRI Engine that expresses a procedure for operating several command messages sequentially.				
Derived From :				
Attributes				
command list	CommandMessage	M	N ord	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 8.17: 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 8.18: ResultList classs

Description: A data class for specifying a list of result parameters.				
Derived From: None				
Attributes				
Parameters	Parameter	M	N ord	Result parameters

Table 8.19: ArgumentList class

Description: A data class for specifying a list of argument parameters.				
Derived From: None				
Attributes				
Parameters	Parameter	M	N ord	Argument parameters

Table 8.20: ParameterList class

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

Derived From: None				
Attributes				
Parameters	Parameter	M	N ord	Configuration parameters

8.2.4.2 Command Result Message

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

Table 8.21: 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

8.2.4.3 Query Message

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

Table 8.22: 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

8.2.4.4 Event Message

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

Table 8.23: 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	O	1	Time limit for obtaining detailed results by get_event_detail().

8.2.4.5 Event Detail Message

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

Table 8.24: 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

8.2.4.6 Error Message

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

Table 8.25: 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 [ISO8601]	O	1	Time limit for obtaining detailed results by get_error_detail().

8.2.4.7 Error Detail Message

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

Table 8.26: 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

8.3 RoIS Profiles

8.3.1 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.

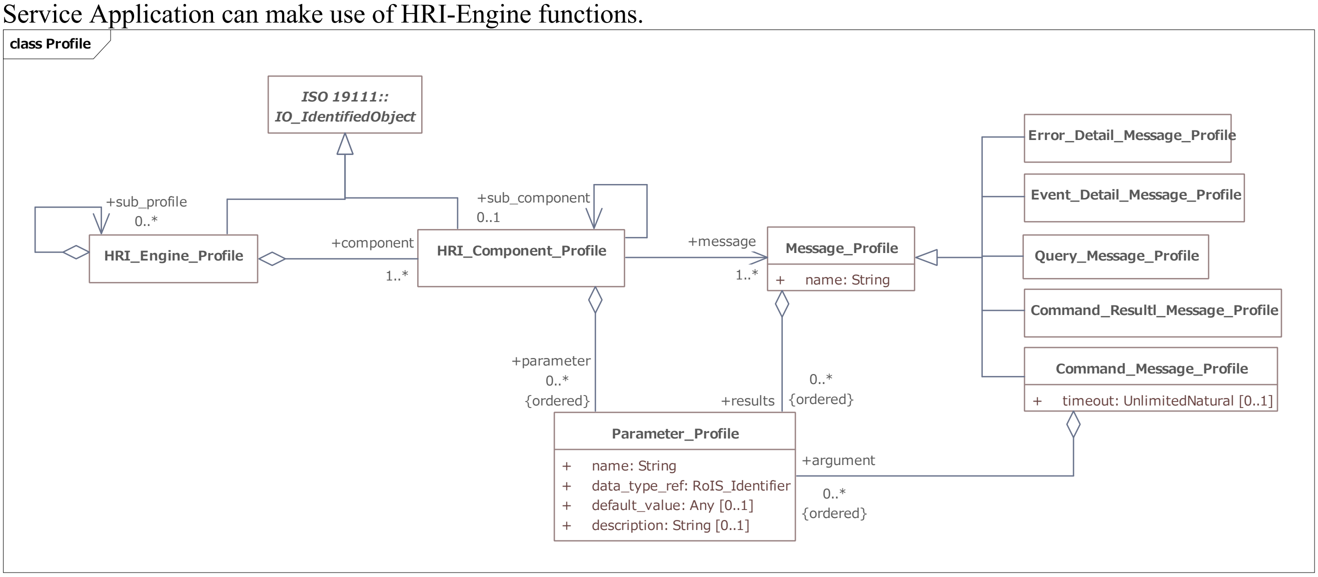


Figure 8.12: RoIS Profile. RoIS profile mainly consists of 4 types of profiles, i.e., "HRI_Engine_Profile," "HRI_Component_Profile," "Message_Profile," and "Parameter_Profile."

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.

RoIS defines following four types of profiles to be explained in the following sections:

- Parameter Profile
- Message Profile
- HRI Component Profile
- HRI Engine Profile

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.

8.3.2 Parameter Profile

The Parameter Profile defines parameters for message arguments and HRI-Engine and HRI-Component parameters.

This profile defines parameter identifier (parameter name), data type, and default value. Items to be defined in this profile are listed in Table 8.27.

Table 8.27: 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

8.3.3 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.

This profile 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. Items to be defined in this profile are listed in Table 8.28.

Table 8.28: 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 8.29.

Table 8.29: 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 8.30.

Table 8.30: Command_Result_Message_Profile

Description: Profile for defining messages for command interface type.
Derived From: Message_Profile_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 8.31.

Table 8.31: 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 8.32.

Table 8.32: 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 8.33.

Table 8.33: Error_Detail_Message_Profile

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

8.3.4 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.

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. Items to be defined in this profile are listed in Table 8.34.

Table 8.34: 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.

8.3.5 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.

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. Items to be defined in this profile are listed in Table 8.35.

Table 8.35: 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.

8.4 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.

8.4.1 Basic HRI Components

In the RoIS Framework, the HRI Components shown in Table 8.36 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 A.

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 8.36: 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	Detects 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
audio streaming	Contorols audio streaming between HRI engines and Services

video streaming	Contorols video streaming between HRI engines and Services

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

Table 8.37: 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 8.38: 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.

8.4.2 RoIS Component Ontology

Data types of parameters in RoIS are statically defined as sub classes of RoSO data types. Parameters used in RoIS functional components are defined as properties and range of them are defined upon RoSO vocabularies. RoIS basic functional components are defined as subclasses of roso:Sensor or roso:Actuator.

Table 8.39: Robotic Interaction Service (RoIS) Component Ontology Metadata

Metadata Term	Value
OntologyIRI	https://www.omg.org/spec/RoIS/RoboticInteractionServiceComponentOntology/
rdfs:label	Robotic Interaction Service (RoIS) Component Ontology
dct:abstract	The Robotic Interaction Service Components Ontology provides vocabularies to describe RoIS basic functional components
cmns-av:copyright	Copyright © 2024-2025 Japan Robot Association
cmns-av:copyright	Copyright © 2024-2025 Korea Association of Robot Industry

cmns-av:copyright	Copyright © 2024-2025 Shibaura Institute of Technology
cmns-av:copyright	Copyright © 2024-2025 National Institute of Advanced Industrial Science and Technology, Japan
cmns-av:copyright	Copyright © 2024 Université Sorbonne Paris Nord
cmns-av:copyright	Copyright © 2024-2025 Object Management Group
dct:references	http://purl.org/dc/terms/
dct:references	http://www.w3.org/2004/02/skos/core#
dct:title	Robotic Interaction Service Component Ontology
owl:versionIRI	https://www.omg.org/spec/RoIS/20250801/RoboticInteractionServiceComponentOntology/

Following properties and classes are defined in RoIS Ontology that are used to describe components' requirements as ontology.

Table 8.40: Robotic Interaction Service (RoIS) Component Ontology Details

Properties

Name	Annotations	Property Axioms
hasDetectionRegion (has detection region)	Definition: indicates a region in which a component can detect targets	Parent Property: roso:*hasAttribute* Domain: roso:Function Range: roso:Region
hasDetectionThreshold (has detection threshold)	Definition: indicates a spatial interval by which a component can distinguish detected targets	Parent Property: roso:*hasAttribute* Domain: roso:Function Range: roso:SpatialInterval
hasDetectionTimelimit (has detection timelimit)	Definition: indicates a time limit within which a component is expected to detect the target	Parent Property: roso:*hasAttribute* Domain: roso:Function Range: cmns-dt:TimeInterval
hasMaximumInterval (has maximum interval)	Definition: indicates a periodic interval within which a component notifies events in maximum	Parent Property: roso:*hasAttribute* Domain: roso:Function Range: cmns-dt:TimeInterval
hasMimimumDistance (has minimum distance)	Definition: indicates a minimum spatial distance that a robot can approach while following the target	Parent Property: roso:*hasAttribute* Domain: roso:Function Range: roso:SpatialInterval
hasMinimumInterval (has minimum interval)	Definition: indicates a periodic interval by which a component can detect targets	Parent Property: roso:*hasAttribute* Domain: roso:Function Range: cmns-dt:TimeInterval
hasTarget (has target)	Definition: indicates an agent or an object as a target of the function	Parent Property: roso:*hasAttribute* Domain: roso:Function Range: cmns-pts:Agent ∪ roso:PhisicalThing
hasTimeLimit (has time limit)	Definition: indicates a time limit by which a component completes the function	Parent Property: roso:*hasAttribute* Domain: roso:Function Range: cmns-dt:TimeInstant

Classes

Name	Annotations	Class Expressions
AudioStreaming (audio streaming)	Definition: component function to transmit audio streaming	Parent Class: roso:Function
FaceDetection (face detection)	Definition: component function to count the number of faces detected in the detection region	Parent Class: roso:Sensing
FaceLocalization (face localization)	Definition: component function to localize positions of faces detected in the detection region	Parent Class: roso:Sensing
Follow (follow)	Definition: component function to move following a target agent	Parent Class: roso:Actuation

GestureRecognition (gesture recognition)	Definition: component function to recognize gestures represented by other agents	Parent Class: roso:Sensing
Move (move)	Definition: component function to move along the indicated path	Parent Class: roso:Actuation
Navigation (navigation)	Definition: component function to navigate another agent to the indicated goal point	Parent Class: roso:Actuation
SoundDetection (sound detection)	Definition: component function to count the number of sound sources detected in the detection region	Parent Class: roso:Sensing
SoundLocalization (sound localization)	Definition: component function to localize positions of sound sources detected in the detection region	Parent Class: roso:Sensing
SpeechRecognition (speech recognition)	Definition: component function to recognize speech sound to text	Parent Class: roso:Sensing
SpeechSynthesis (speech synthesis)	Definition: component function to synthesize speech sound from text	Parent Class: roso:Actuation
PersonDetection (person detection)	Definition: component function to count the number of persons detected in the detection region	Parent Class: roso:Sensing
PersoIdentification (person identification)	Definition: component function to identify persons detected in the detection region	Parent Class: roso:Sensing
PersonLocalization (person localization)	Definition: component function to localize positions of persons detected in the detection region	Parent Class: roso:Sensing
Reaction (reaction)	Definition: component function to perform motions to interact with other agents	Parent Class: roso:Actuation
VideoStreaming (video streaming)	Definition: component function to transmit video streaming	Parent Class: roso:Function

8.4.3 System Information

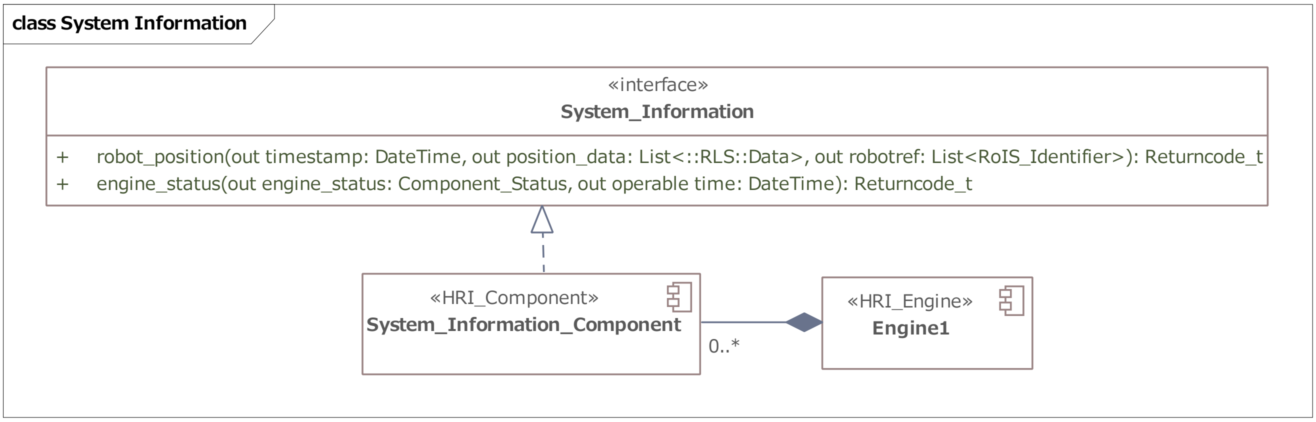


Figure 8.13: System Information

Table 8.41: 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 [ISO8601]	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	List<DateTime> [ISO8601]	O	Operable time of the HRI Engine that includes this basic component.

8.4.4 Person Detection

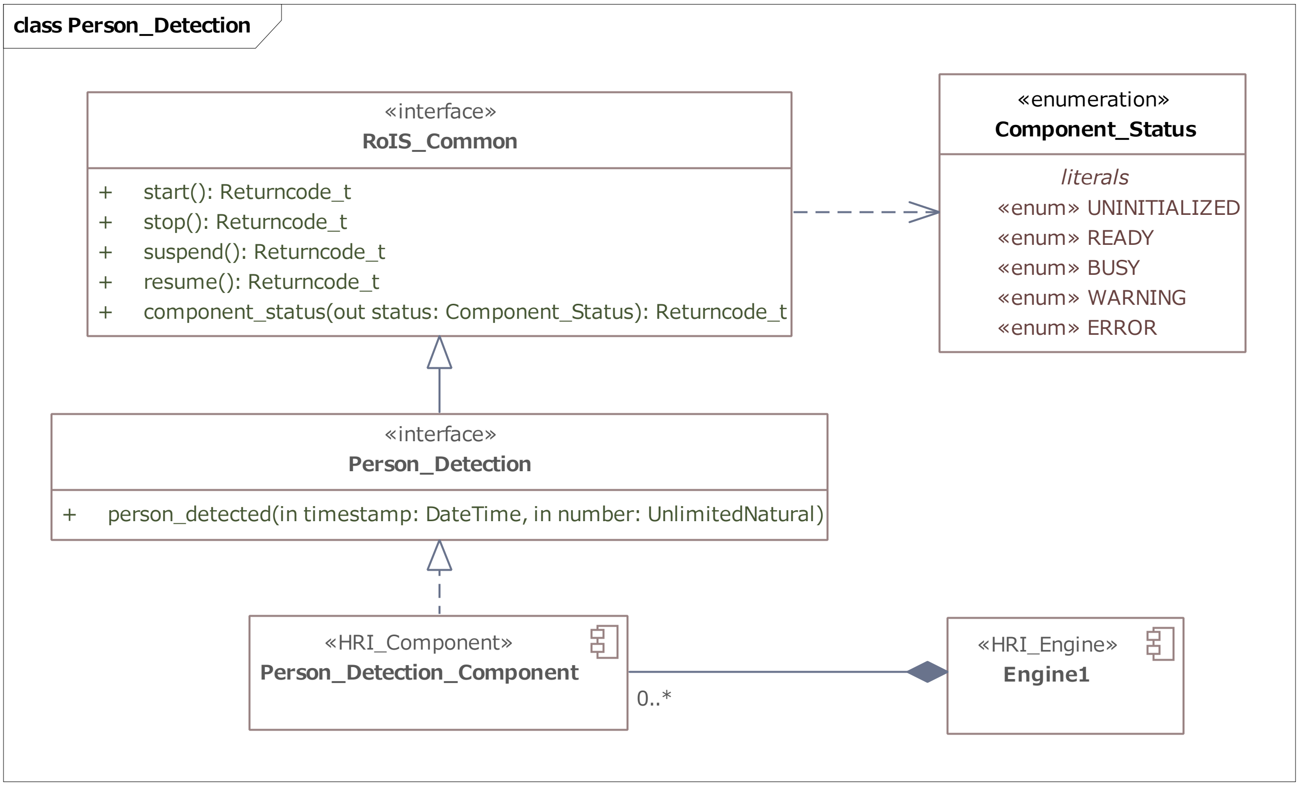


Figure 8.14: Person Detection

Table 8.42: Person Detection

Description: This is a component for detecting 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 [ISO8601]	M	Measurement time.
result	number	UnlimitedNatural	M	Number of detected persons

8.4.5 Person Localization

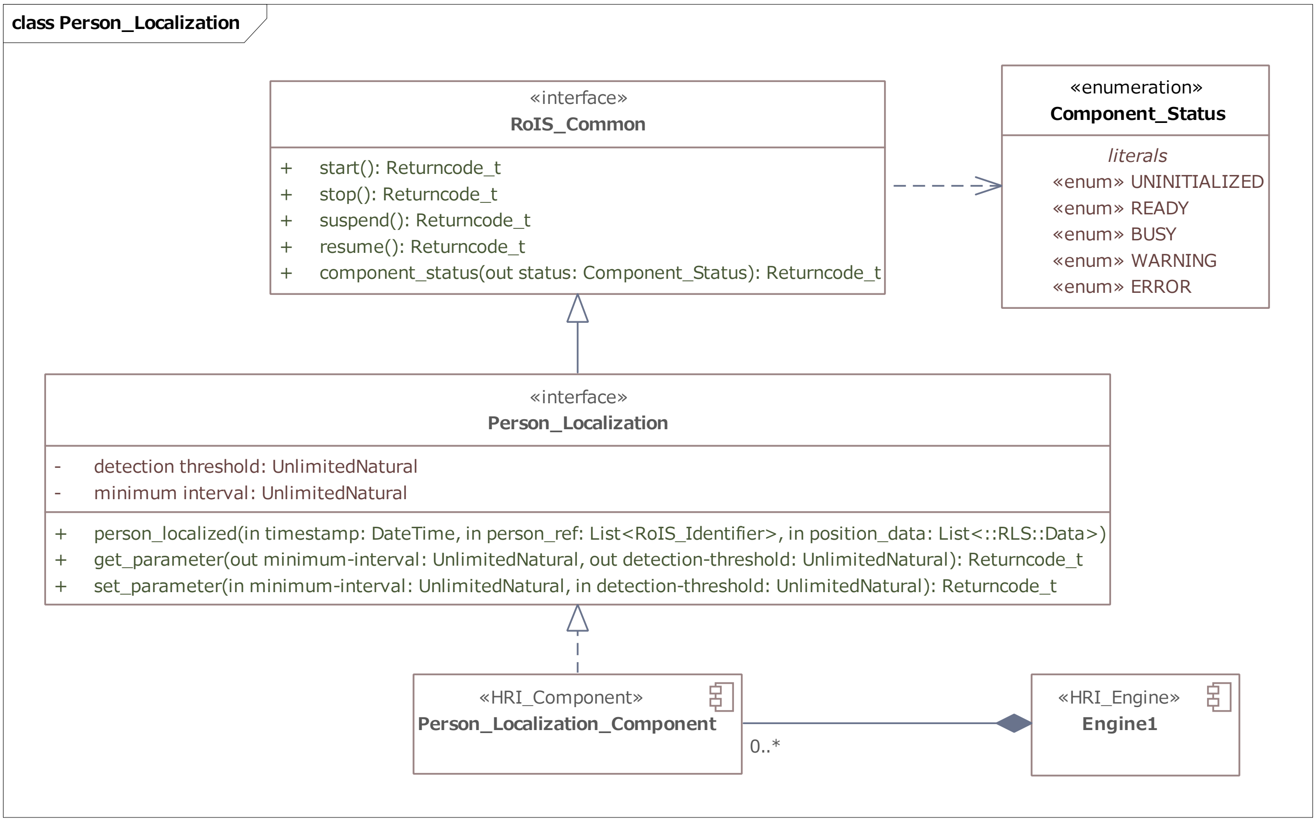


Figure 8.15: Person Localization

Table 8.43: Person Localization

Description: This is a component for detecting position of persons. This component notifies position of the detected people when the position has been localized. This functionality is essential for typical robotic services; for example, when a robot finds a person close to it, the robot may approach to the person and start asking if there is something the robot can do for the person. In some advanced robotic services, an environmental sensing system may find out a person is lost and order robots to approach the person for help.				
Command Method				
set_parameter	Specifies person localization parameters.			
argument	detection threshold	UnlimitedNatural	O	This component notifies an event if the distance of movement since previous event notification exceeds the threshold value.
argument	minimum interval	UnlimitedNatural	O	This component notifies an event if the period since previous event notification exceeds the value of minimal interval.
Query Method				
get_parameter	Obtains person localization parameters.			

result	detection threshold	UnlimitedNatural	O	This component notifies an event if the distance of movement since previous event notification exceeds the threshold value.
result	minimum interval	UnlimitedNatural	O	This component notifies an event if the period since previous event notification exceeds the value of minimal interval.
Event Method				
person_localized	Notifies position of people when the position has localized.			
result	timestamp	DateTime [ISO8601]	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.

8.4.6 Person Identification

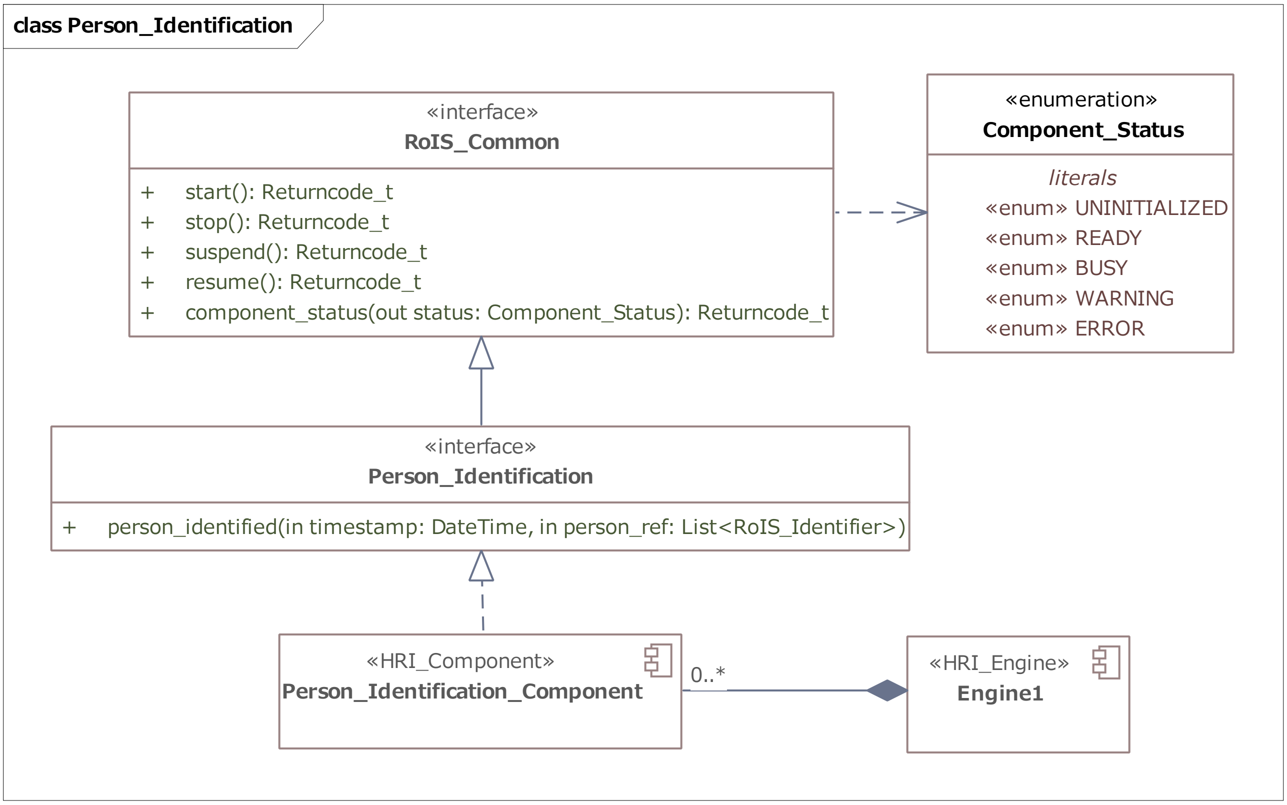


Figure 8.16: Person Identification

Table 8.44: 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 [ISO8601]	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.

8.4.7 Face Detection

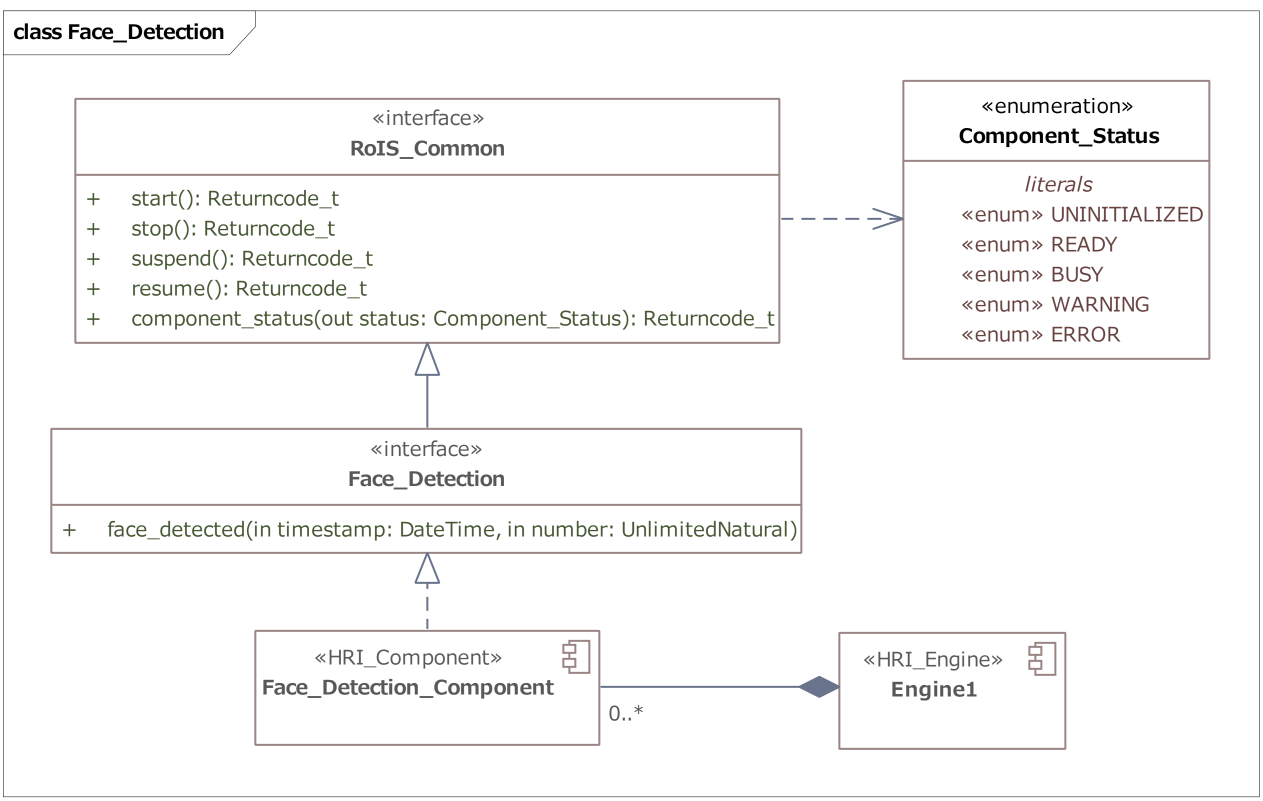


Figure 8.17: Face Detection

Table 8.45: Face Detection

Description: This is a component for detecting number of human faces. This component notifies a number of the detected faces when the number has changed. This functionality is similar to "person_detection" component but it is treated as a separate component. This is because often the detection of human face has an individual meaning in the Service Applications. For example, if a robot detects a person but the person is not facing to the robot, the robot may not talk to the person. In such a case, the robot may move to the other direction of the person or wait until the person turns to the robot. Therefore this functionality is also essential for various robotic services.				
Event Method				
face_detected		Notifies number of human face when the number has changed.		
result	timestamp	DateTime [ISO8601]	M	Measurement time.
result	number	UnlimitedNatural	M	Number of human faces

8.4.8 Face Localization

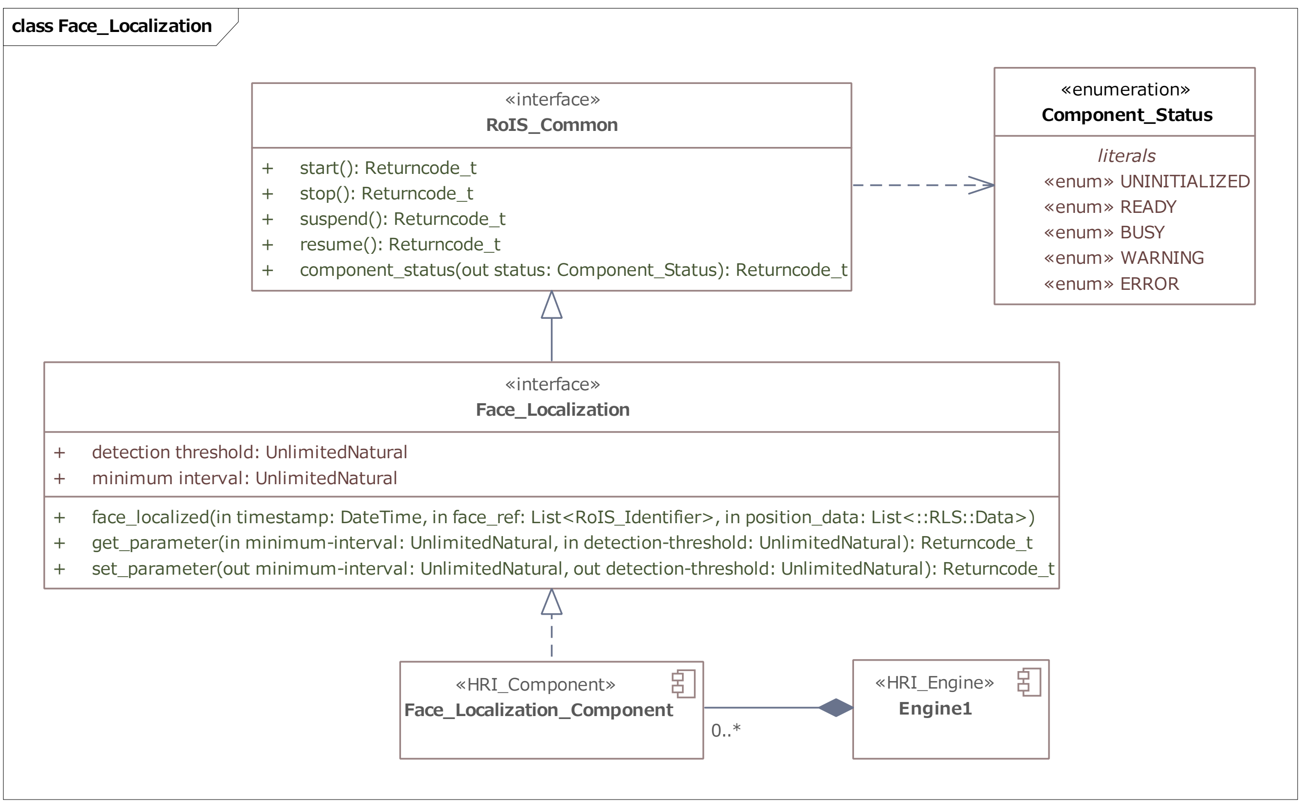


Figure 8.18: Face Localization

Table 8.46: Face Localization

Description: This is a component for detecting the 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 the human, the robot may need to look up the person. In such a 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	UnlimitedNatural	O	This component notifies an event if the distance of movement since previous event notification exceeds the threshold value.
argument	minimum interval	UnlimitedNatural	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	UnlimitedNatural	O	This component notifies an event if the distance of movement since previous event notification exceeds the threshold value.
result	minimum interval	UnlimitedNatural	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 [ISO8601]	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.

8.4.9 Sound Detection

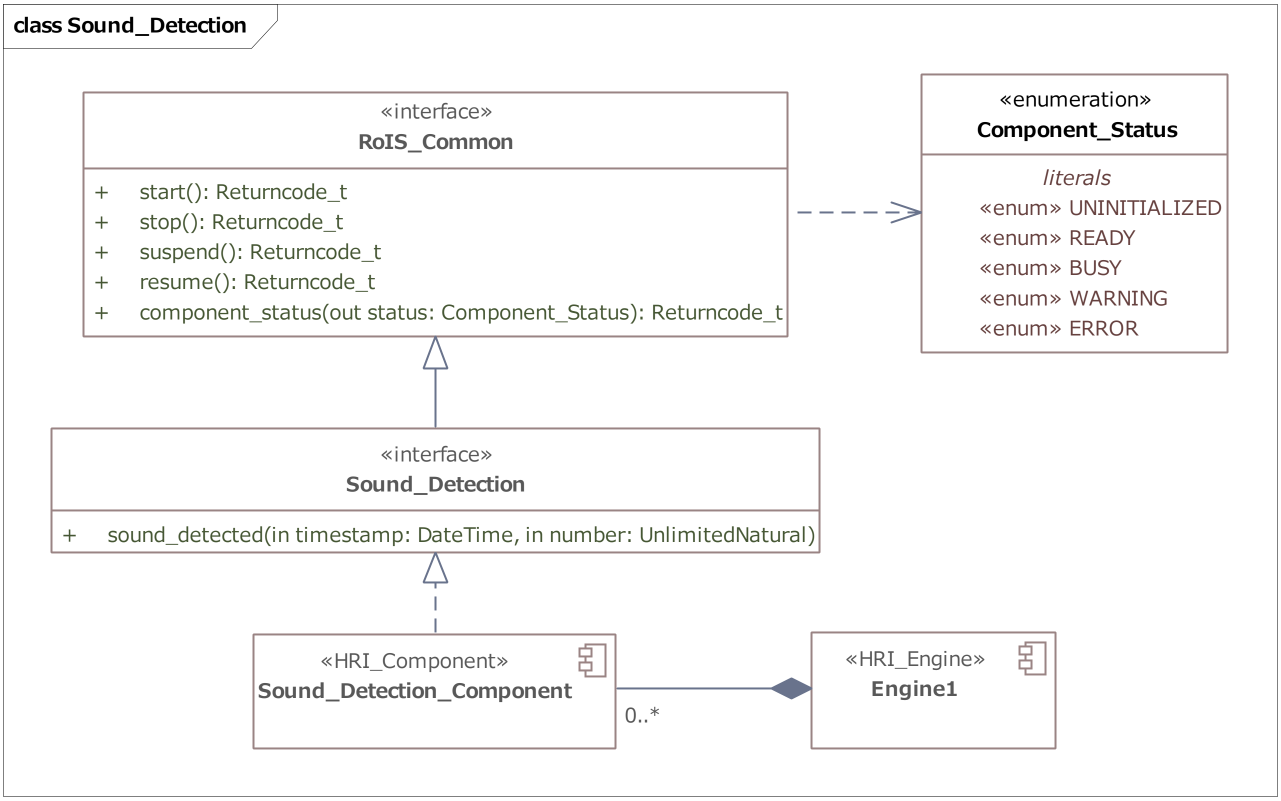


Figure 8.19: Sound Detection

Table 8.47: 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 [ISO8601]	M	Measurement time.
result	number	UnlimitedNatural	M	Number of sound sources. If the component can not detect sound sources separately, this parameter shall be 1 or 0.

8.4.10 Sound Localization

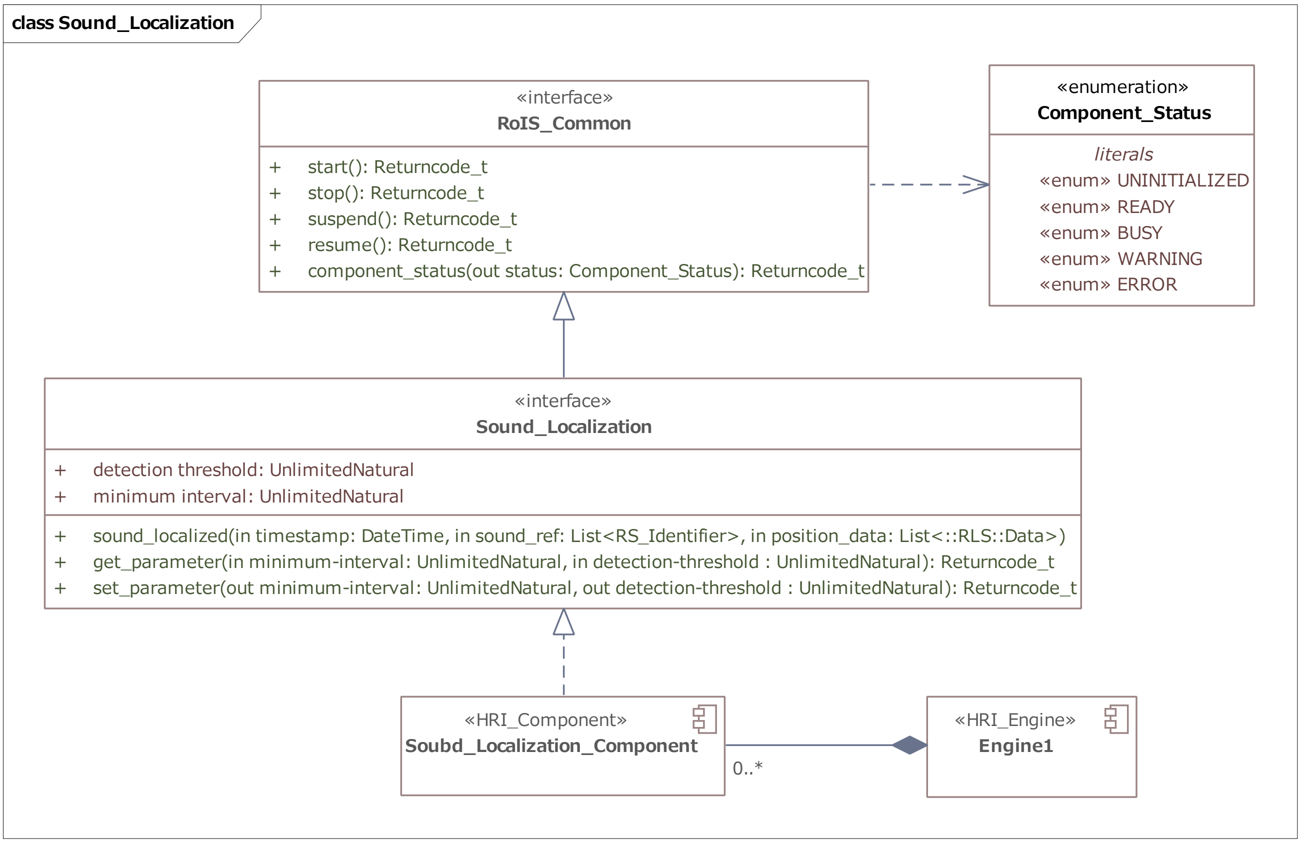


Figure 8.20: Sound Localization

Table 8.48: 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 the user wants to start interaction.				
Command Method				
set_parameter	Specifies sound localization parameters.			
argument	detection threshold	UnlimitedNatural	O	This component notifies an event if the distance of movement since previous notification exceeds the threshold value.
argument	minimum interval	UnlimitedNatural	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	UnlimitedNatural	O	This component notifies an event if the distance of movement since previous notification exceeds the threshold value.
result	minimum interval	UnlimitedNatural	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 [ISO8601]	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.

8.4.11 Speech Recognition

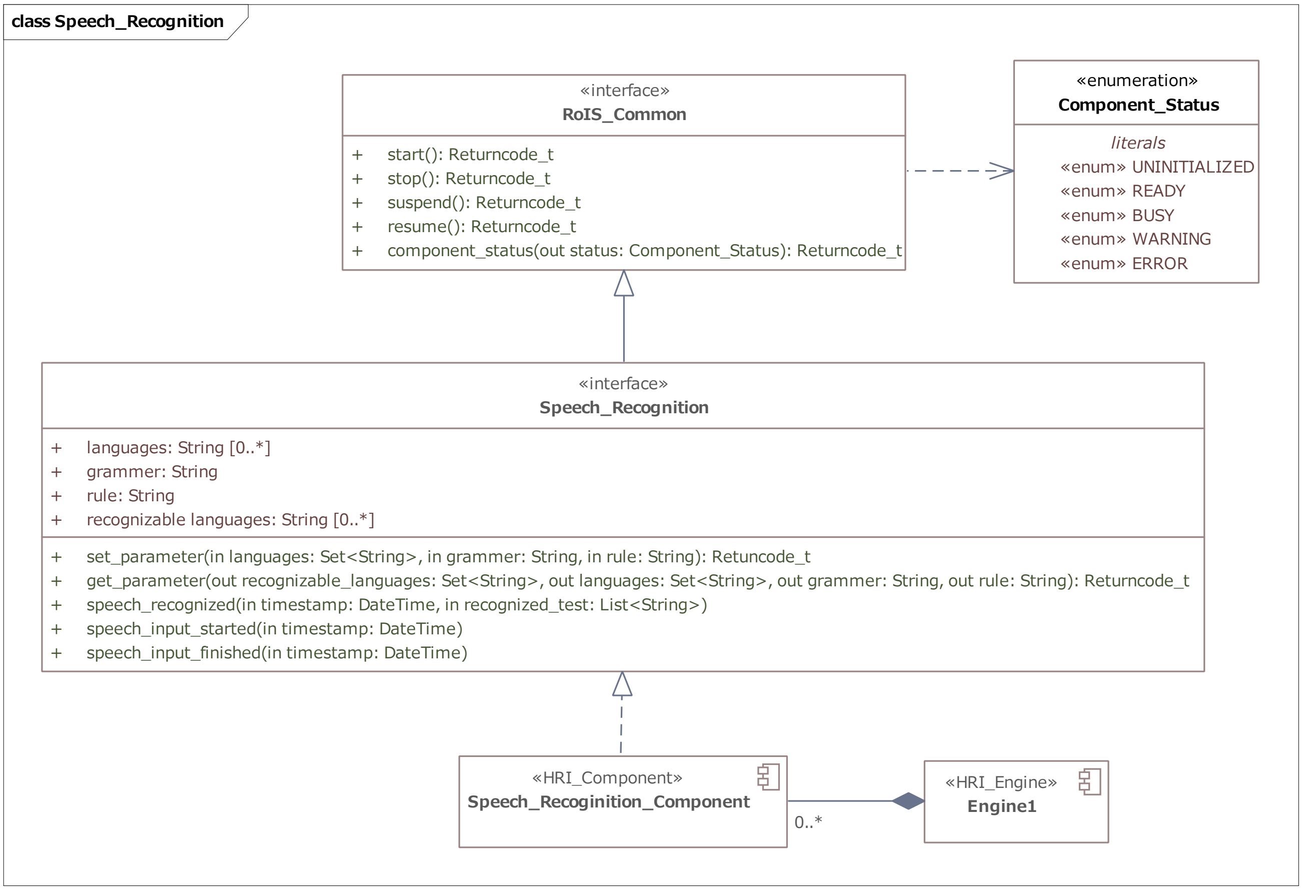


Figure 8.21: Speech Recognition

Table 8.49: Speech Recognition

Description: This is a component for recognizing human speech. This component notifies text data of the recognized speech when the speech has been recognized. This functionality is essential for human robot interactions, from simply ordering the robot to do something to giving enough information to the Service Application for appropriate services. Here, we assume speech recognition algorithm which is not configurable by a descriptive grammar (e.g., W3C-SRGS). See Annex C for speech recognition algorithm which can be configured by a descriptive grammar. Mandatory requirement for the speech recognition component is to return result in string format.				
Command Method				
set_parameter	Specifies speech recognition paramters.			
argument	languages	Set<String> [ISO639- 1]	M	Specifies languages the speech recognizer will recognize.
argument	grammer	String	O	Specifies grammar for the speech recognizer.
argument	rule	String	O	Specifies active rule in the grammer.

Query Method				
get_parameter	Obtains speech recognition paramters.			
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 [ISO8601]	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 [ISO8601]	M	Time when the speech input has started.
speech_input_finished	Notifies the recognizer has detected end of speech input.			
result	timestamp	DateTime [ISO8601]	M	Time when the speech input has ended.

8.4.12 Gesture Recognition

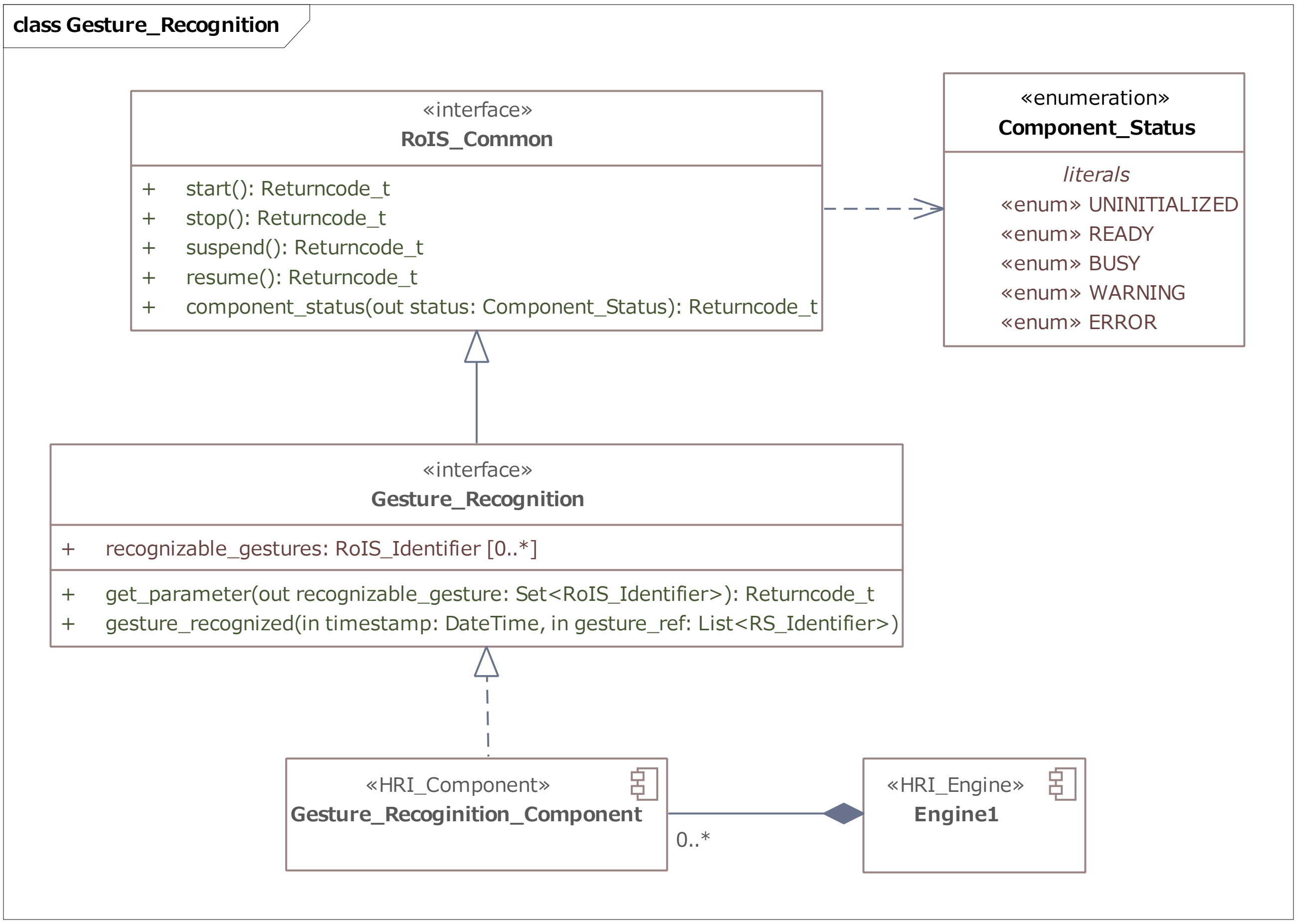


Figure 8.22: Gesture Recognition

Table 8.50: 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 [ISO8601]	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.

8.4.13 Speech Synthesis

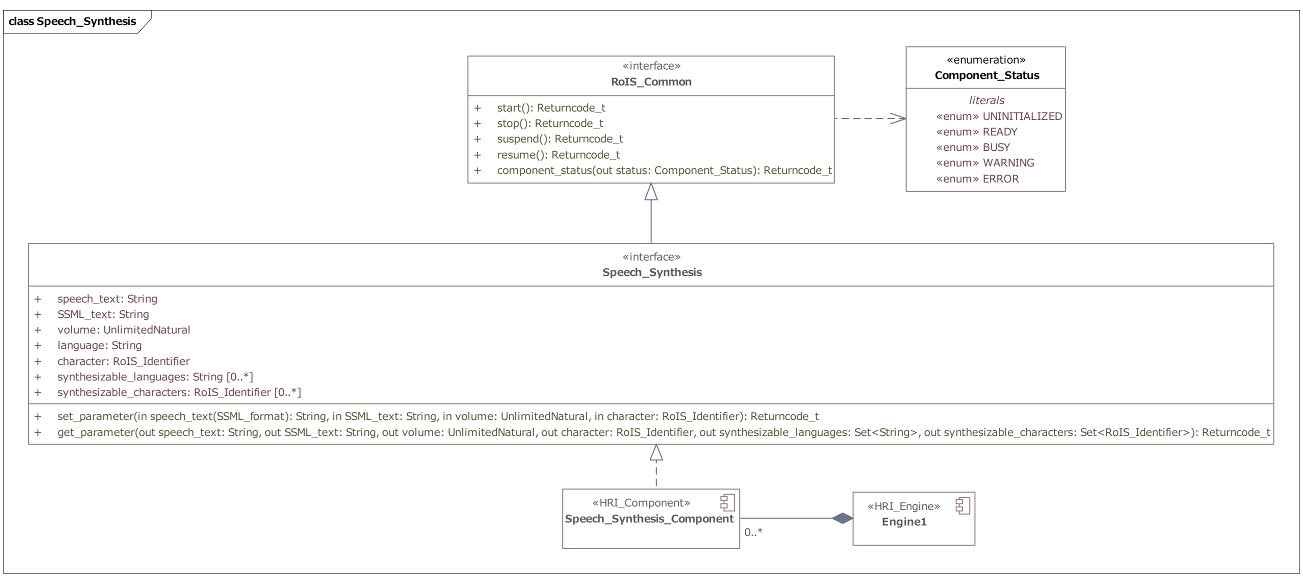


Figure 8.23: Speech Synthesis

Table 8.51: Speech Synthesis

Description: This is a component for generating synthesized speech. This component acts to generate synthesized speech by specifying the speech text. This functionality is essential for human robot interactions. Naturally the robot talks to the user when it communicates with the user. Here, we assume speech synthesis algorithm which can synthesize a voice in multiple characters (e.g., male, female, robotic). W3C-SSML format is used to specify the language and the prosodic parameters. For speech synthesis algorithm which cannot specify the prosodic parameters, XML tags in W3C-SSML format shall be skipped.				
Command Method				
set_parameter	Specifies speech synthesis parameters.			
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	UnlimitedNatural	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 parameters.			
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	UnlimitedNatural	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.				

8.4.14 Reaction

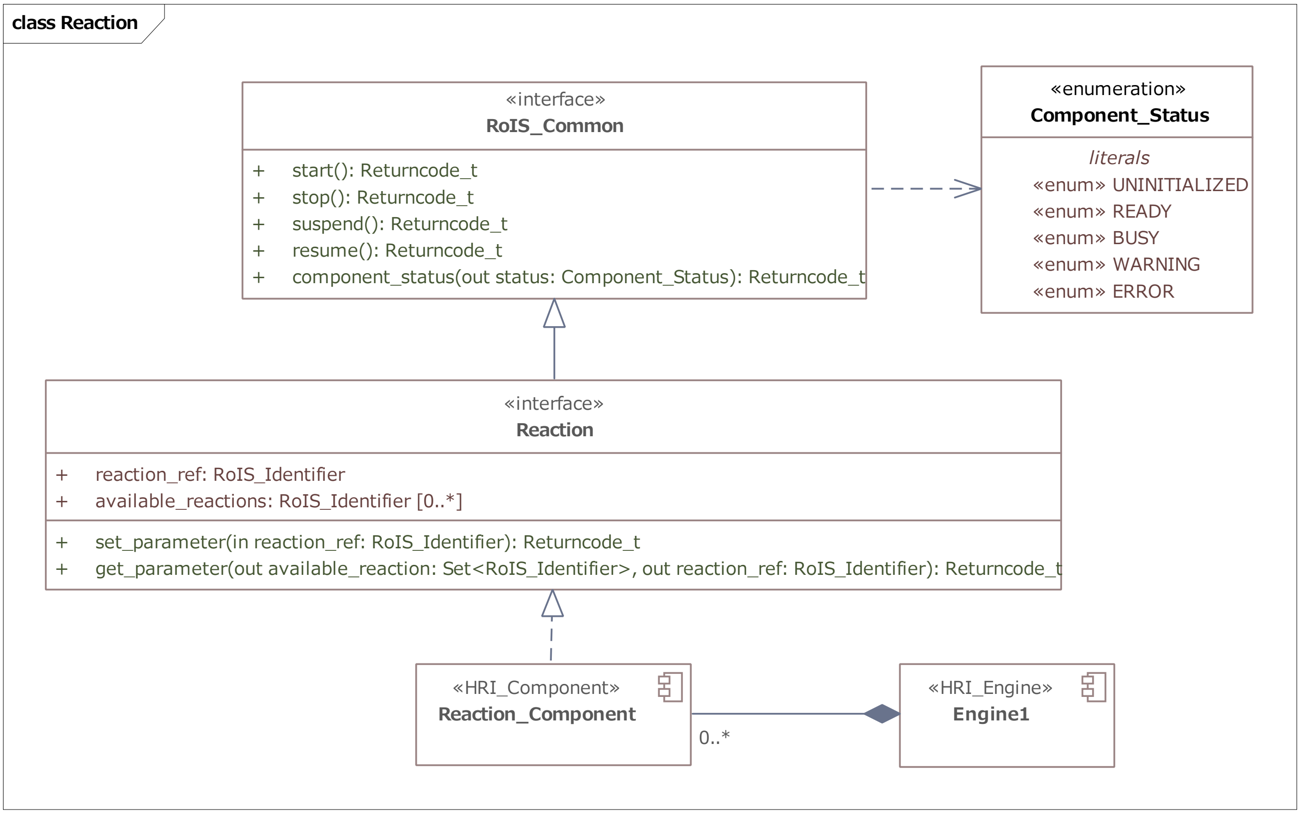


Figure 8.24: Reaction

Table 8.52: Reaction

Description: This is a component for executing specified reaction. This component acts to execute specified reaction by specifying the reaction ID. This functionality is useful for human robot interactions. Generally it is difficult for the Service Application programmers to specify the robot reaction in detail since it depends on the hardware architecture. Therefore, this component provides a simple way to specify the robot reaction. For example, if the Service Application needs to express “yes”/“no” to the user, the Service Application programmer can execute the reaction only by specifying the reaction ID for “yes”/”no” reaction without regard for the expression method, such as nodding yes/no or showing a message for yes/no on its display. The meaning of reaction is different among such as countries. Also the executable reactions may vary from robot to robot. The reaction shall be simply specified by reaction ID and the Service Application shall understand the meaning of the ID by the reference for the ID.	
Command Method	
set_parameter	Specifies reaction paramters.

argument	reaction ref	RoIS_Identifier	M	Reaction type. The type is specifyed by reaction ID. Reference information related to the ID shall be specified with the each ID.
Query Method				
get_parameter	Obtains reaction parameters.			
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.

8.4.15 Navigation

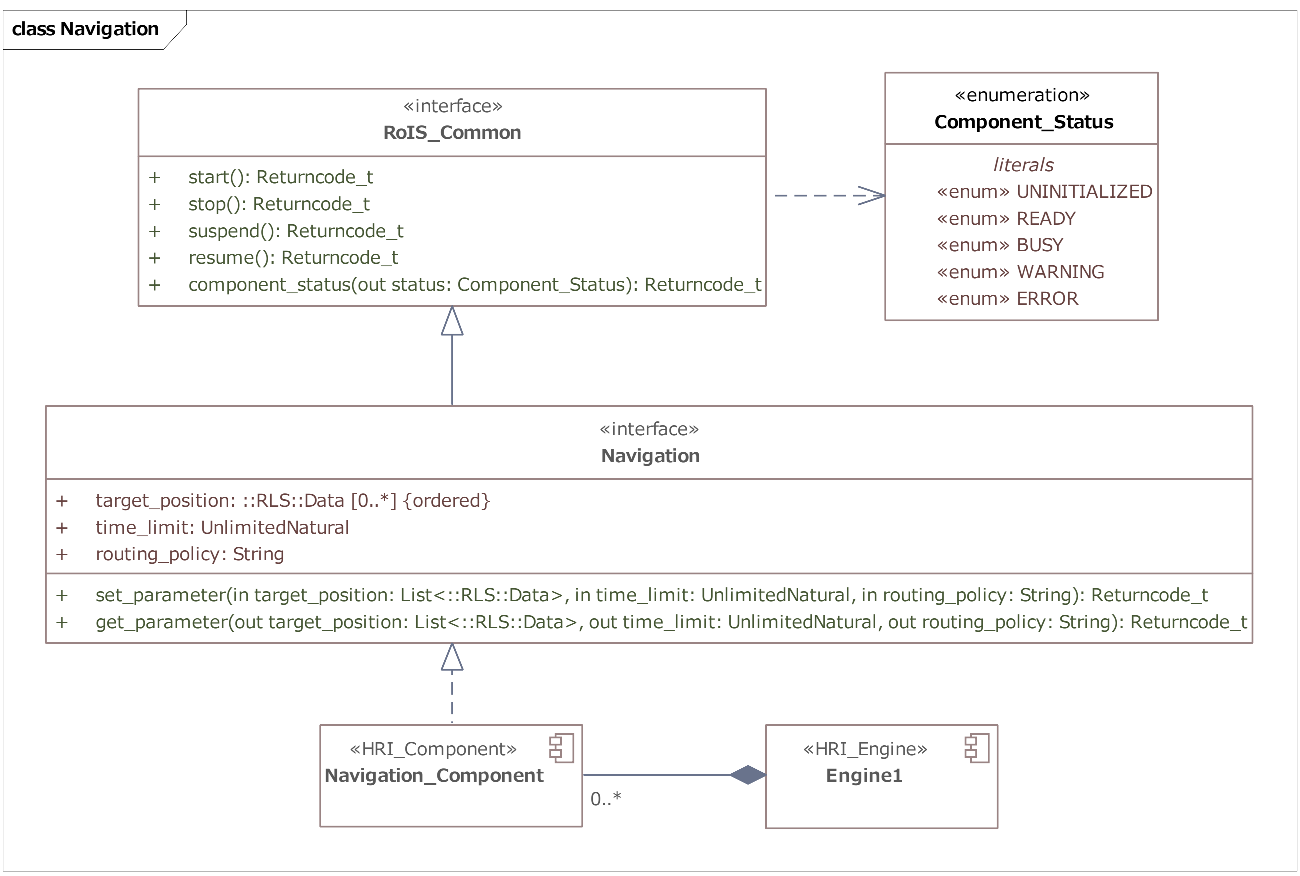


Figure 8.25: Navigation

Table 8.53: 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 navigation 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 contain ID of the target position. The position data of the target position may be included in this entry, or may be obtained by referring by the ID. This may also be accompanied with additional information such as speed.
argument	time_limit	UnlimitedNatural	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	UnlimitedNatural	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.”

8.4.16 Follow

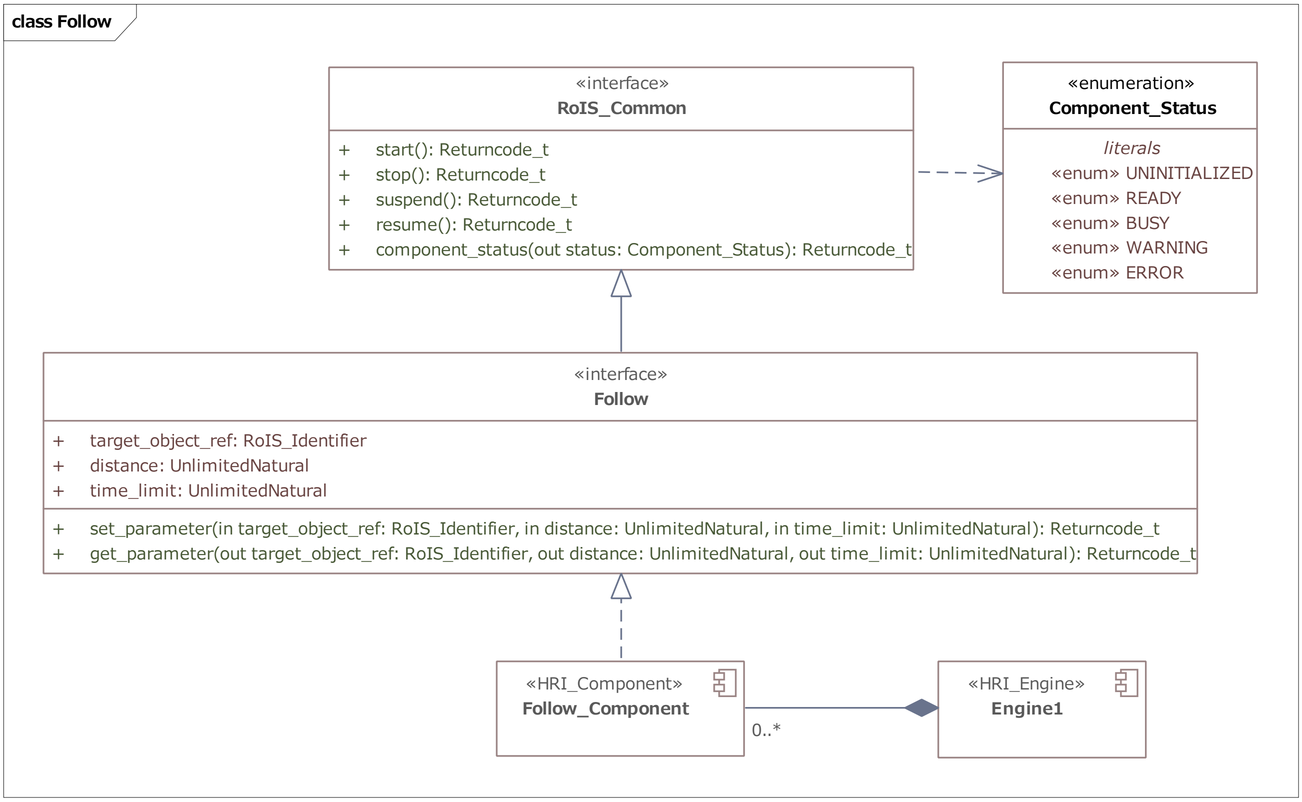


Figure 8.26: Follow

Table 8.54: Follow

<table>
<tr><td colspan="5">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.</td></tr>
<tr><td colspan="5">Command Method</td></tr>
<tr><td colspan="2">set_parameter</td><td colspan="3">Specifies parameters for follow.</td></tr>
<tr><td>argument</td><td>target object ref</td><td>RoIS_Identifier</td><td>M</td><td>Taget object. The object is specified by object IDs.

The reference information related to the ID shall be specified with each ID.</td></tr>
</table>

argument	distance	UnlimitedNatural	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	UnlimitedNatural	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	UnlimitedNatural	M	Minimum distance between the target and the robot. The distance shall be specified in millimeters.
result	time_limit	UnlimitedNatural	O	Time limit for determining whether it is impossible to continue following. The time shall be specified in milliseconds.

8.4.17 Move

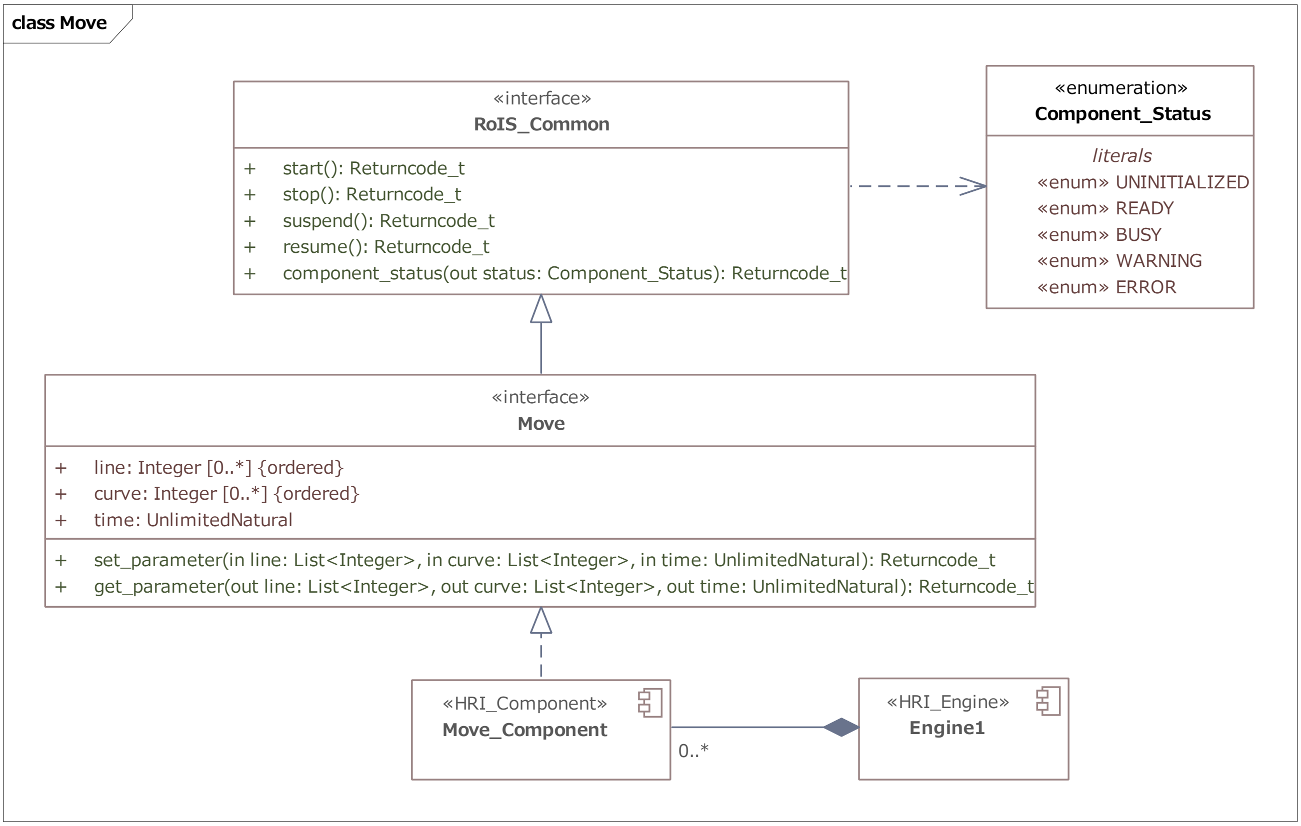


Figure 8.27: Move

Table 8.55: 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 specifying the curve. The radius shall be specified in millimeter and the direction shall be specified in degree.
argument	time	UnlimitedNatural	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 specifying the curve. The radius shall be specified in millimeter and the direction shall be specified in degree.
result	time	UnlimitedNatural	O	Specified operating time. The time shall be specified in milliseconds.
Condition: These elements shall be selected according to the motion.				

8.4.18 Audio Streaming

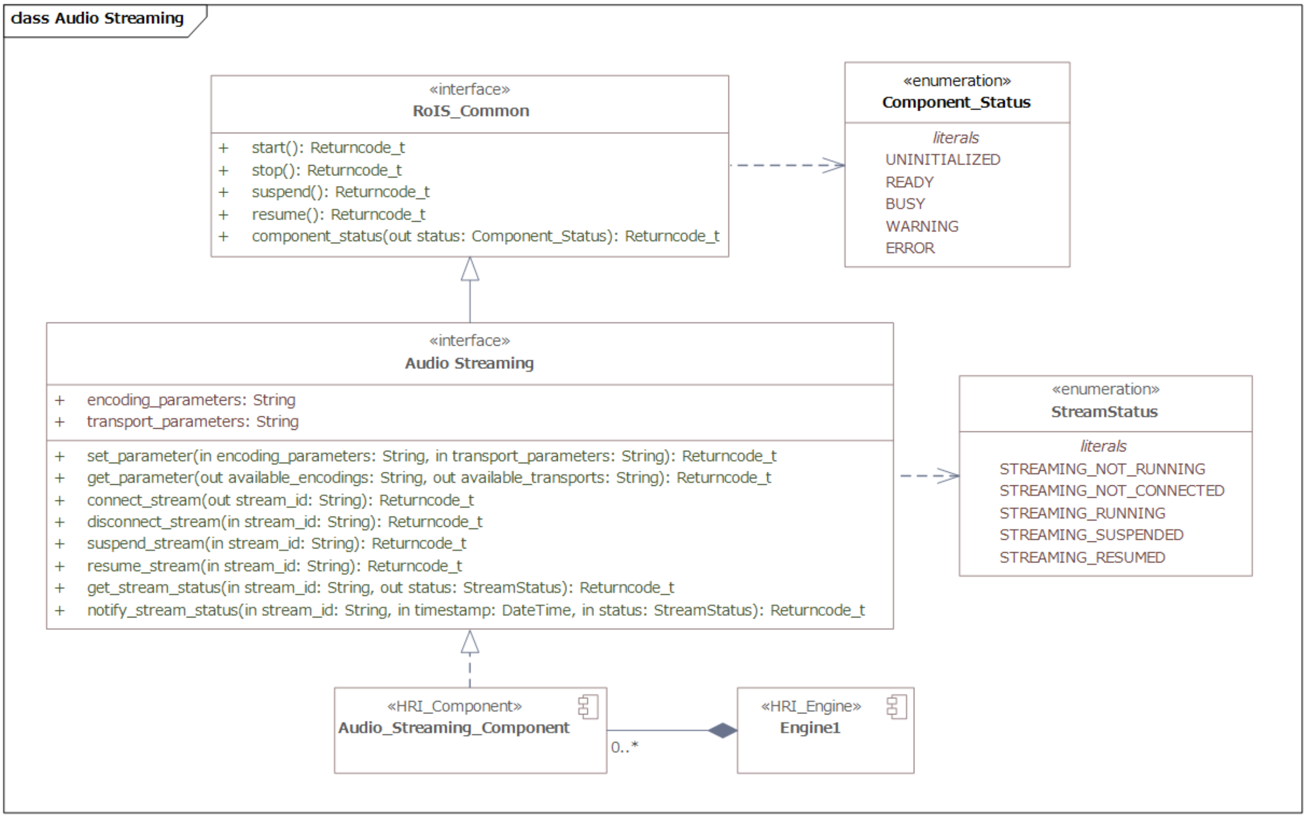


Figure 8.28: Audio Streaming

Table 8.56: Audio Streaming

Description: This component controls audio streaming between HRI engines and services. This component transmits audio streams captured by the component and receives audio streams from services. The encoding and transport are negotiated through parameter exchange. This functionality is essential for performing various robotic services, especially remotely operated by person.				
Command Method				
set_parameter	Specifies audio streaming parameters.			
argument	encoding parameters	String	O	This component encodes an audio stream with specified encoding parameters.
argument	transport parameters	String	O	This component sets up a transport channel with specified transport parameters.
connect_stream				
result	stream_id	String	O	An id assigned to distinguish the stream.
disconnect_stream	Disconnect			

argument	stream_id	String	M	Specifies the stream to disconnect.
suspend_stream		Requests the HRI engine to suspend the specified stream.		
argument	stream_id	String	M	Specifies the stream to suspend.
resume_stream		Requests the HRI engine to suspend the specified stream.		
argument	stream_id	String	M	Specifies the stream to resume.
Query Method				
get_parameter				
result	available encodings	String	O	This component can accept request for audio stream encoding.
result	available transports	String	O	This component can accept request for audio stream transport.
get_stream_status				
argument	stream_id	String	M	Specifies the stream to query its status.
result	status	Stream_Status	O	Status of the stream
Event Method				
notify_stream_status		Notifies status of the stream captured at the services.		
argument	stream_id	String	M	The ID of the stream
argument	timestamp	DateTime [ISO8601]	M	Measurement time
argument	status	Stream_Status	M	Status of the stream

8.4.19 Video Streaming

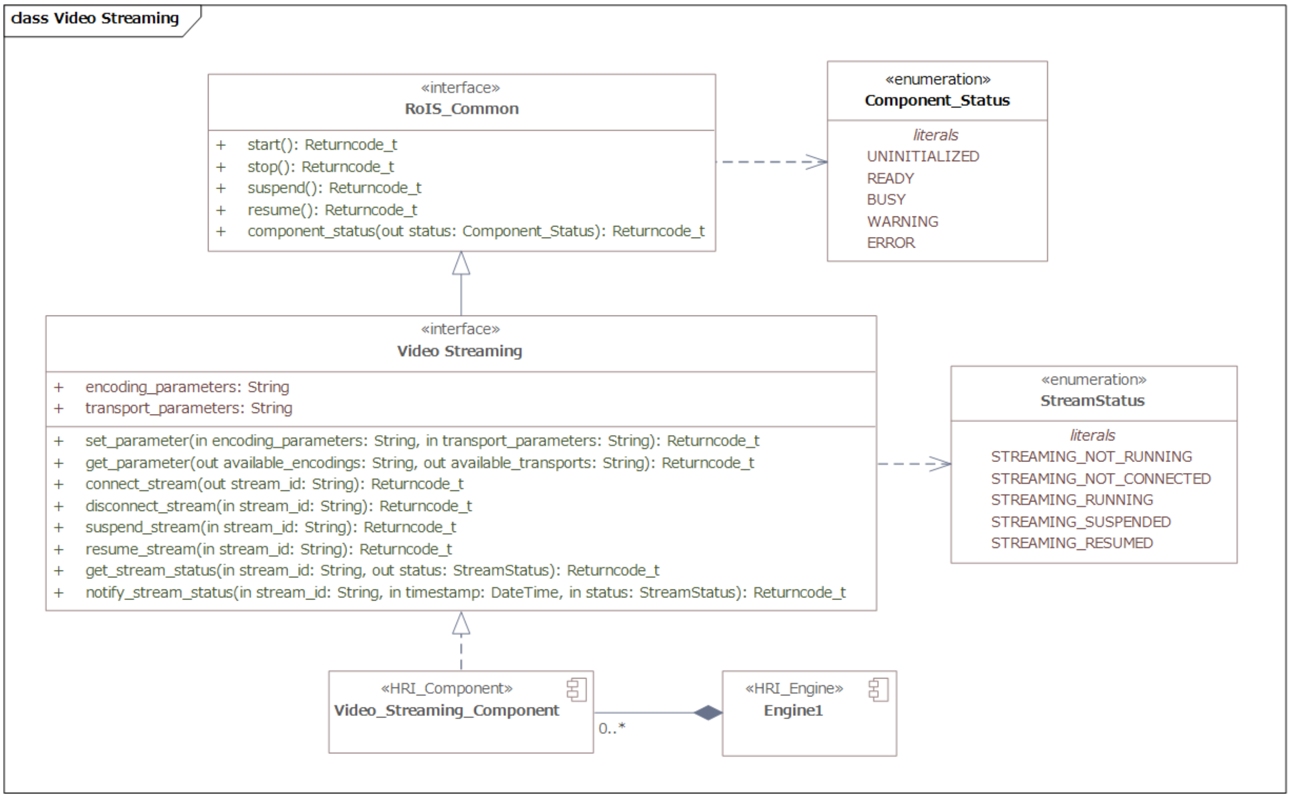


Figure 8.29: Video Streaming

Table 8.57: Video Streaming

Description: This component controls video streaming between HRI engines and services. This component transmits video streams captured by the component and receives audio streams from services. The encoding and transport are negotiated through parameter exchange. This functionality is essential for performing various robotic services, especially remotely operated by person.				
Command Method				
set_parameter	*Specifies audio streaming parameters.*			
argument	encoding parameters	String	O	This component encodes an video stream with specified encoding parameters.
argument	transport parameters	String	O	This component sets up a transport channel with specified transport parameters.
connect_stream				
result	stream_id	String	O	An id assigned to distinguish the stream.
disconnect_stream	Disconnect			
argument	stream_id	String	M	Specifies the stream to disconnect.

suspend_stream	Requests the HRI engine to suspend the specified stream.			
argument	stream_id	String	M	Specifies the stream to suspend.
resume_stream	Requests the HRI engine to suspend the specified stream.			
argument	stream_id	String	M	Specifies the stream to resume.
Query Method				
get_parameter				
result	available encodings	String	O	This component can accept request for audio stream encoding.
result	available transports	String	O	This component can accept request for audio stream transport.
get_stream_status				
argument	stream_id	String	M	Specifies the stream to query its status.
result	status	Stream_Status	O	
Event Method				
notify_stream_status	Notifies status of the stream captured at the services.			
argument	stream_id	String	M	The ID of the stream
argument	timestamp	DateTime [ISO8601]	M	Mearsurement time
argument	status	Stream_Status	M	Status of the stream

This page intentionally left blank.

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”>
```

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.

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:CommaneMessageProfileType” rois:name=”start” />
    <rois:MessageProfile xsi:type=”rois:CommaneMessageProfileType” rois:name=”stop” />
    <rois:MessageProfile xsi:type=”rois:CommaneMessageProfileType” rois:name=”suspend” />
    <rois:MessageProfile xsi:type=”rois:CommaneMessageProfileType” 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>
```

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>
```

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.

```
<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” />
   . . . . .
```

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>
    <rois:SubProfile>
        <gml:identifier>urn:x-rois:def:HRIEngine:ATR::SubHRI02</gml:identifier>
        <gml:name>SubHRI02</gml:name>
```

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”>
            <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>
```

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 attribute ‘xsi:type’ is specified as ‘rois:ConcurrentCommandsType.’

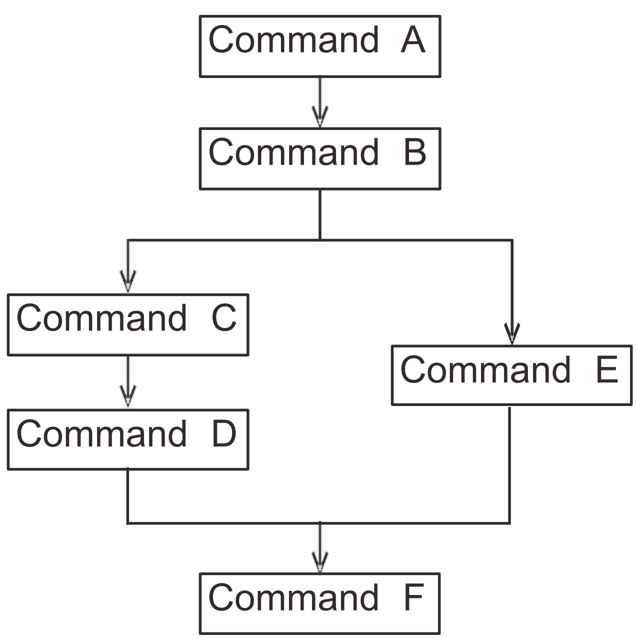


Figure B.1: Structure of the CommandUnitSequence example

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 B.1. 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)

Table C.1: Speech Recognition ([W3C-SRGS])

Description: Recognize speech input. Here, we assume speech recognition algorithm which *is* configurable by a descriptive grammar ([W3C-SRGS]). Mandatory requirement for the speech recognition component is to return N-best result. For the speech recognition algorithm which can only output one candidate, returning a list filled with 1-best result is recommended. String of recognized text can contain either a word or a sentence.				
Command Method				
set_parameter	Specifies speech recognition parameters.			
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 [ISO8601]	M	Time when the recognition has completed.
result	timestamp_speech_start	DateTime [ISO8601]	O	Time when the speech input has started.
result	timestamp_speech_end	DateTime [ISO8601]	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 [ISO8601]	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 referring 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 [ISO8601]	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 referring 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.

C.4 Wheelchair Robot

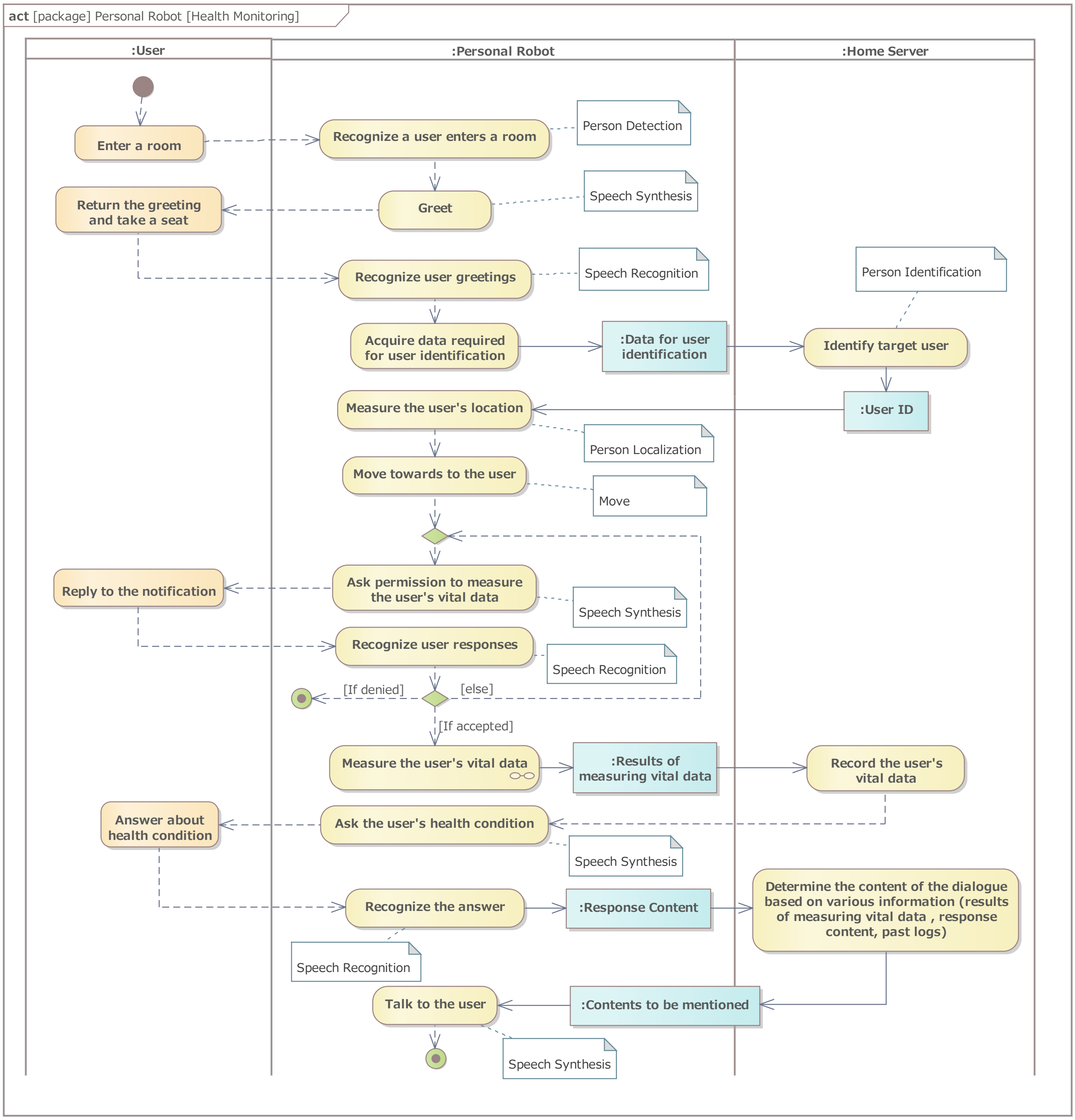
Table C.6: Wheelchair Robot

Description: This component defines a common interface to control the semi-autonomous wheelchair robot that extends the existing definition of Navigation component in RoIS 1.0. The component has facilities to localize itself, to avoid collisions with obstacles and people in a corridor, and to notify event of 'brake' with 'wc_status_changed' event.				
Command Method				
brake		Changes wheelchair brake status.		
argument	status	Boolean	M	Changes wheelchair brake status.
set_parameter		Specifies parameters to control the wheelchair robot.		
argument	preferred_max_speed	Double	O	Preferred max-speed defined by the user in mm/sec.
argument	behaviour_tag	String	O	Tag of the interaction scenario to be executed.
argument	navigation_mode	String	O	Specifies control mode from two literals “Location” or “Tag.”
Query Method				
wc_status_change		Gets the current status of wheelchair		
result	position	String	M	Robot position in comma separated double values [x.x, y.y].
result	brake_status	Boolean	M	Wheelchair brake status.
result	pause_status	Boolean	M	Wheelchair pause status.
result	wc_status	String	M	Six possible value literals: running, obstacle, path_error, stopped, brakes, paused.
result	nav_status	String	M	Two possible value literals: running, stopped.
get_parameter		Obtains parameters to control the wheelchair robot		
result	preferred_max_speed	Double	O	Preferred max-speed defined by the user in mm/sec.
result	behavior_tag	String	O	Tag of the interaction scenario to be executed.
result	navigation_mode	String	O	Specifies control mode from two literals “Location” or “Tag.”
Event Method				
wc_status_change		Notifies the status change of wheelchair as an event as well as the query message with same name.		
result	position	String	M	Robot position in comma separated double

				values [x.x, y.y].
result	brake_status	Boolean	M	Wheelchair brake status.
result	pause_status	Boolean	M	Wheelchair pause status.
result	wc_status	String	M	Six possible value literals: running, obstacle, path_error, stopped, brakes, paused.
result	nav_status	String	M	Two possible value literals: running, stopped.

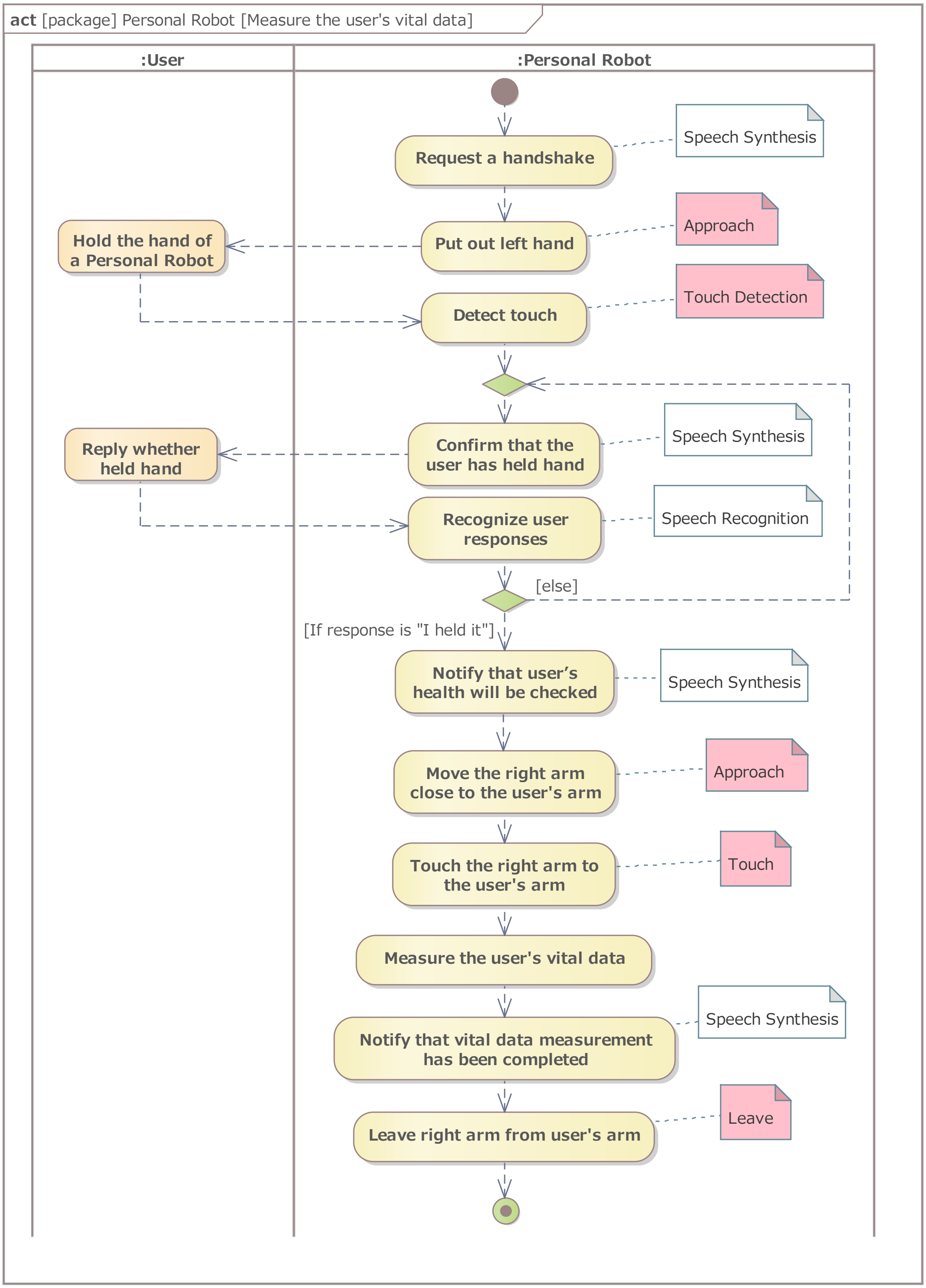
C.5 Example of interactive measuring vital data with a personal robot

FigureC.5.1: shows a scenario in which a personal home robot measures a user's vital data. (In the activity diagram below, the RoIS component responsible for each activity is described with a comment.



FigureC.1: Example of interactive measuring vital data with a personal robot

An example of a detailed scenario of "Measure the user’s vital data" is shown below.



FigureC.2: Detailed scenario example of " Measure the user’s vital data"

The user-defined HRI components defined based on the above scenario are shown below.

C.5.1. Approach

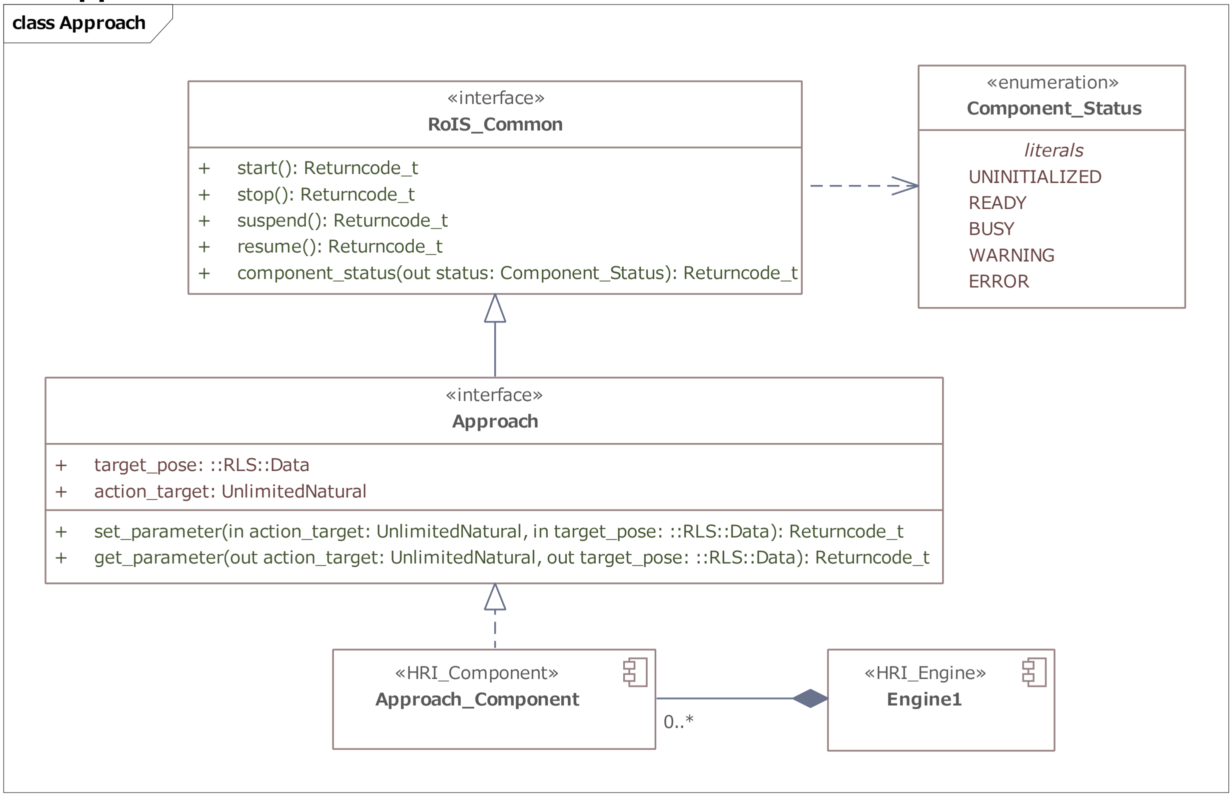


Figure C.3: Approach

Table C.6: Approach

Description: This component is to be used to move a specified part close to the target pose. This component performs autonomous movement to a specified pose. Approach function is essential for typical robot services to move a hand or arm to a target pose. This component allows Service Application to command robots to perform approach without considering the actual arm configuration or degrees of freedom. The target pose shall be specified as a spatial pose based on a coordinate system appropriate to the specified part to be moved. The actual motion trajectory to the target pose, and strategies such as trajectory generation and obstacle avoidance are left to the the component implementation.				
Command Method				
set_parameter	Specifies parameters for approach.			
argument	action_target	UnlimitedNatural	M	Part of the robot structure which the action is performed. Specify the entire robot, hand position, arm posture, etc.
argument	target_pose	Data [RLS]	M	Target pose of the action. It is defined in a coordinate system according to the part of the action.
Query Method				
get_parameter	Obtains parameters for approach.			
result	action_target	UnlimitedNatural	M	Part of the robot structure which the action is performed. The entire robot, hand position, arm posture, etc. are specified.
result	target_pose	Data [RLS]	M	Target pose of the action. It is defined in a coordinate system according to the part of the action.

C.5.2. Leave

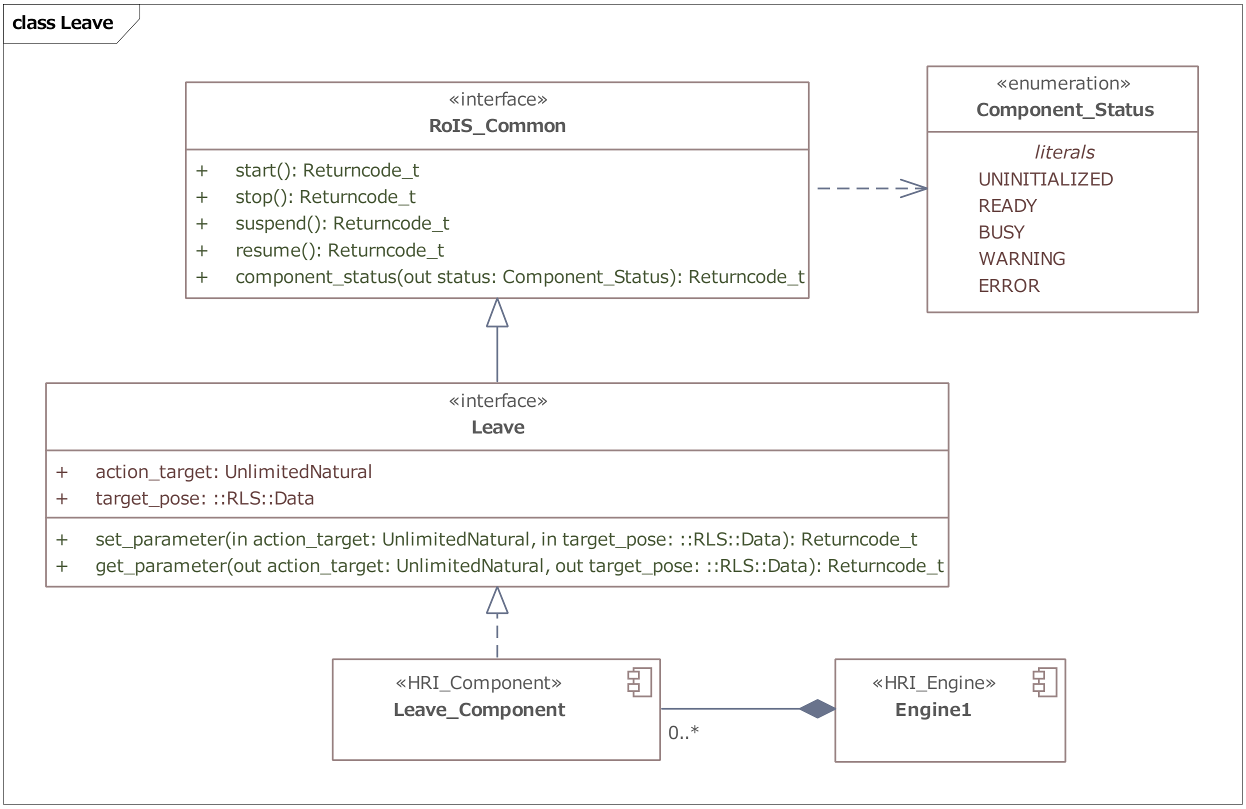


Figure C.4: Leave

Table C.7: Leave

Description: This component is used to move a specified part away to a pose from the target. This component is used in conjunction with Approach. This component performs autonomous return of a specified part to a specified pose, such as to its initial pose. Similar to the approach function, the leave function is essential for typical robot services to move a hand or arm away to a target pose. This component allows Service Application to command robots to perform leave without considering the actual arm configuration or degrees of freedom. The target pose shall be specified as a spatial pose based on a coordinate system appropriate to the specified part being moved. The actual motion trajectory to the target pose, and strategies such as trajectory generation and obstacle avoidance are left to the the component implementation.				
Command Method				
set_parameter		Specifies parameters for leave.		
argument	action_target	UnlimitedNatural	M	The part on which the action is performed. Specify the entire robot, hand position, arm posture, etc.
argument	target_pose	Data [RLS]	M	Target pose of the action. It is defined in a coordinate system according to the part of the action.
Query Method				
get_parameter		Obtains parameters for leave.		
result	action_target	UnlimitedNatural	M	The part on which the action is performed. The entire robot, hand position, arm posture, etc. are specified.
result	target_pose	Data [RLS]	M	Target pose of the action. It is defined in a coordinate system according to the part of the action.

C.5.3. Touch Detection

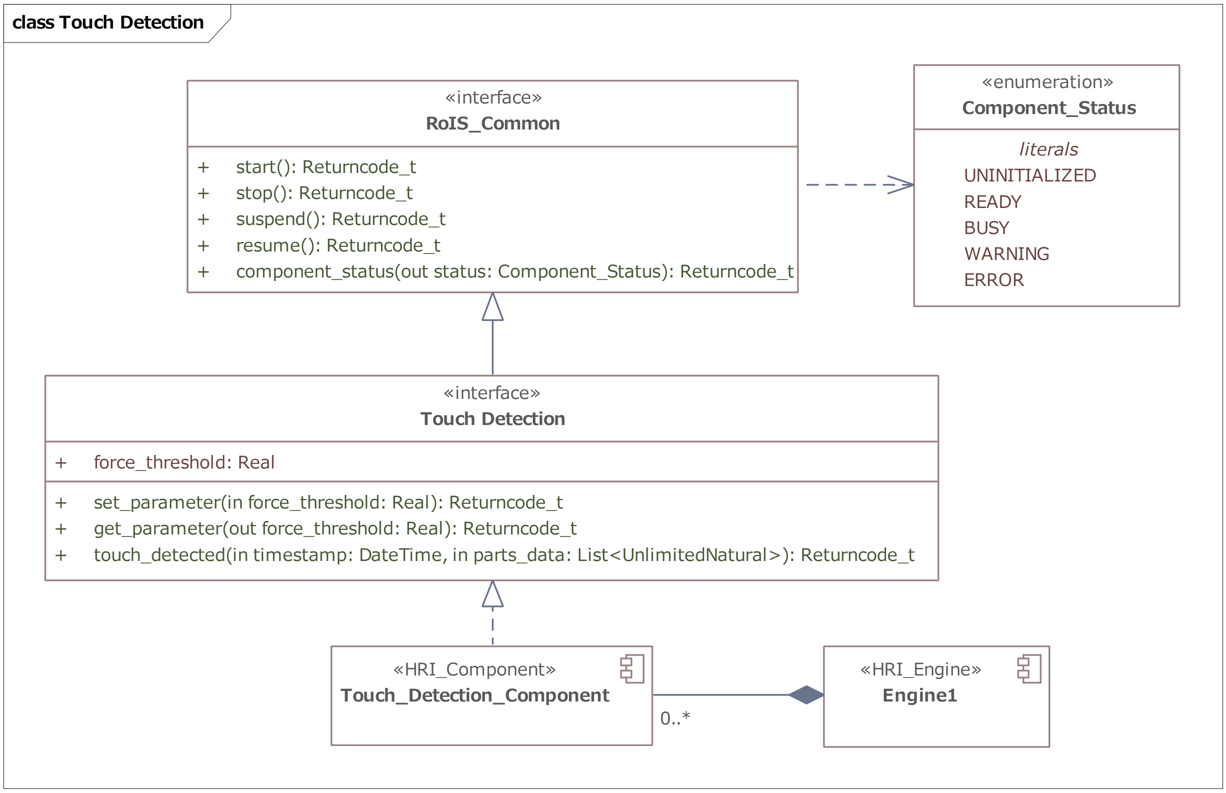


Figure C.5: Touch Detection

Table C.8: Touch Detection

Description: This component is used for detecting number of touch points. This component notifies a list of identifiers of detected contact parts when the number of contact parts changes. The body part is defined according to the service to be performed, such as the right upper arm, the left lower arm, or the hand. Touch Detection function is essential for typical robotic services that involve contact with some objects. Additionally, for services that do not intend for contact to occur, Touch Detection can also be used to stop the service when contact is detected in order to prevent harm to those around. This component also detects changes in the number of contact points, and can also notify when contact is lost. The method of contact detection, the detectable parts, etc. are left to the component implementation.				
Command Method				
set_parameter	Specifies parameters for touch detection.			
argument	force_threshold	Real	M	The force threshold for determining contact.
Query Method				
get_parameter	Obtains parameters for touch detection.			
result	force_threshold	Real	M	The force threshold for determining contact.
Event Method				
touch_detected	Notifies touch detected.			
argument	timestamp	DateTime [ISO8601]	M	Measurement time.
argument	parts_data	List<UnlimitedNatural>	M	A list of identifiers of the parts where contact was detected.

C.5.4. Touch

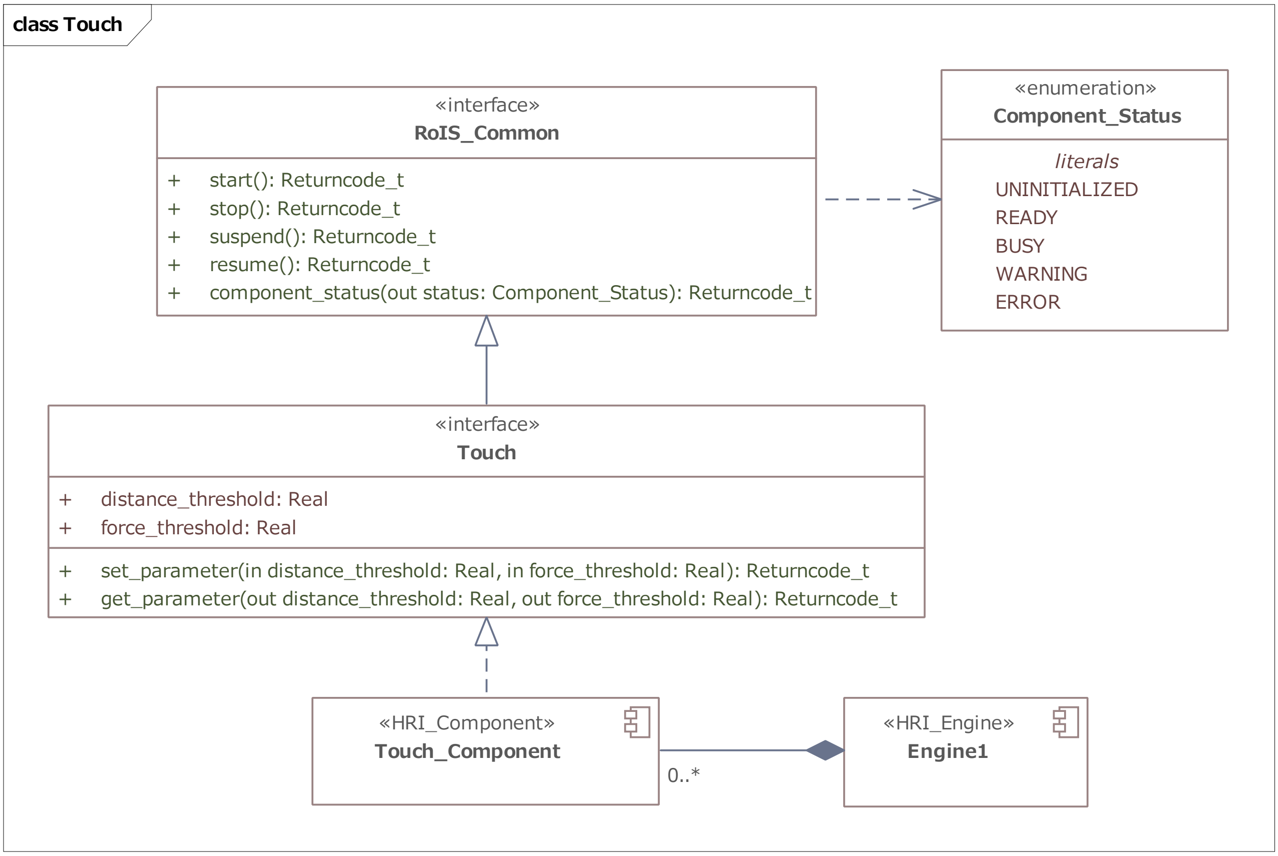


Figure C.6: Touch

Table C.9: Touch

Description: This component performs make contact with a human using the specified pose or force. Touch function is essential for robots to perform services that involve human contact. This component is solely responsible for contacting with humans. To move the robot close to the contact point, robot services need to use the approach component in advance. Ensuring safety in the event of contact depends on the component's implementation.				
Command Method				
set_parameter		Specifies parameters for touch.		
argument	distance_threshold	Real	C	Distance threshold to contact location. Used when contact force threshold is not specified.
argument	force_threshold	Real	C	Contact Force Threshold. Used when distance threshold is not specified.
get_parameter		Obtains parameters for touch.		
result	distance_threshold	Real	C	Distance threshold to contact location. Setted when force threshold is not specified.
result	force_threshold	Real	C	Contact Force Threshold. Setted when distance threshold is not specified.

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

Annex E: Examples of Conditions (informative)

Conditions are used as arguments in function calls described below.

1. In System Interface, to inquire system status (get_profile() and get_error_detail()).
2. In Command Interface, to search and bind components (search(), bind_any()), and to retrieve result of command execution (get_command_result()).
3. In Query Interface, to inquire component status (query()).
4. In Event Interface, to subscribe events (subscribe()), and to retrieve event detail (get_event_detail()).

Those conditions are defined as QueryExpression in [ISO19143].

Following subsections illustrate examples of conditions used to search components.

E.1 Empty Condition

The outer element SearchCondition is introduced to describe a condition. ComponentCondition is a container to describe a filter for component selection. ComponentGroup condition is also used to describe filters for multiple components, so it can contain a filter and several component conditions.

```
<?xml version=”1.0” encoding=”UTF-8” ?>
<unr:SearchCondition xmlns:unr=”http://www.irc.atr.jp/std/unr/0.1” xmlns:fes=”http://www.opengis.net/fes/2.0”>
  <unr:ComponentGroupCondition>
    <fes:filter/>
    <unr:ComponentCondition id=”cond1” type=”” mode=”exclusive”>
      <fes:filter/>
    </unr:ComponentCondition>
  </unr:ComponentGroupCondition>
```

E.2 Search By Property Values

The following example illustrates a condition to find components by specifying property values.

```
<?xml version="1.0" encoding="UTF-8" ?>

<unr:SearchCondition xmlns:unr="http://www.irc.atr.jp/std/unr/0.1"
        xmlns:fes="http://www.opengis.net/fes/2.0">

    <unr:ComponentGroupCondition
            types="RobotProfileId type2" aliases="a b">
        <fes:And>

            <fes:Within>

                <fes:ValueReference>a/Property/location</fes:ValueReference>

                <fes:Literal>Floor_3</fes:Literal>

            </fes:Within>

            <fes:PropertyIsEqualTo>

                <fes:ValueReference>a/Identifier</fes:ValueReference>
                <fes:Literal>example_robot1</fes:Literal>

            </fes:PropertyIsEqualTo>

            <fes:Within>

                <fes:ValueReference>Property/location</fes:ValueReference>

                <fes:Literal>Floor_2</fes:Literal>

            </fes:Within>

            <fes:PropertyIsEqualTo>
                <fes:ValueReference>a/Identifier</fes:ValueReference>

                <fes:Literal>example_robot1</fes:Literal>

            </fes:PropertyIsEqualTo>

        </fes:And>

    <unr:ComponentCondition>

        <fes:filter/>

    </unr:ComponentCondition>

</unr:SearchCondition>
```

E.3 Search by location name

The following example illustrates a condition to find components by specifying locations of them.

```
<?xml version="1.0" encoding="UTF-8" ?>

<unr:SearchCondition xmlns:unr="http://www.irc.atr.jp/std/unr/0.1"
        xmlns:fes="http://www.opengis.net/fes/2.0">

    <unr:ComponentCondition id="cond2"

            type="unr:x-rois:def:component:OMG::ExampleComponent" mode="exclusive">
        <fes:And>

            <fes:PropertyIsEqualTo>

                <fes:ValueReference>Property/min_speed</fes:ValueReference>

                <fes:Literal>200</fes:Literal>

            </fes:PropertyIsEqualTo>

            <fes:PropertyIsEqualTo>

                <fes:ValueReference>Name</fes:ValueReference>

                <fes:Literal>dummy1</fes:Literal>

            </fes:PropertyIsEqualTo>

            <fes:PropertyIsEqualTo>
                <fes:ValueReference>RobotId</fes:ValueReference>

                <fes:Literal>dummy_robot1</fes:Literal>

            </fes:PropertyIsEqualTo>

        </fes:And>

    </unr:ComponentCondition>

</unr:SearchCondition>
```

E.4 Search by location coordinates

The following example illustrates a condition to find components by specifying location coordinates described by using GML elements.

```
<?xml version="1.0" encoding="UTF-8" ?>
<unr:SearchCondition xmlns:unr="http://www.irc.atr.jp/std/unr/0.1"
            xmlns:fes="http://www.opengis.net/fes/2.0"
            xmlns:gml="http://www.opengis.net/gml/3.2">
    <unr:ComponentGroupCondition types="RobotProfileId type2" aliases="a b">
        <fes:And>
            <fes:Within>
                <fes:ValueReference>a/Property/location</fes:ValueReference>
                <gml:Envelope srsName="urn:atr:def:crs:IDK::01">
                    <gml:lowerCorner>13.0983 31.5899</gml:lowerCorner>
                    <gml:upperCorner>35.5472 42.8143</gml:upperCorner>
                </gml:Envelope>
            </fes:Within>
            <fes:PropertyIsEqualTo>
                <fes:ValueReference>a/Identifier</fes:ValueReference>
                <fes:Literal>example_robot1</fes:Literal>
            </fes:PropertyIsEqualTo>
            <fes:Within>
                <fes:ValueReference>Property/location</fes:ValueReference>
                <gml:Envelope srsName="urn:atr:def:crs:IDK::01">
                    <gml:lowerCorner>13.0983 31.5899</gml:lowerCorner>
                    <gml:upperCorner>35.5472 42.8143</gml:upperCorner>
                </gml:Envelope>
            </fes:Within>
            <fes:PropertyIsEqualTo>
                <fes:ValueReference>a/Identifier</fes:ValueReference>
                <fes:Literal>example_robot1</fes:Literal>
            </fes:PropertyIsEqualTo>
        </fes:And>
        <unr:ComponentCondition id="cond1" type=""></unr:ComponentCondition>
    </unr:ComponentGroupCondition>
```

Annex F: RoIS Service Description Examples

(Informative)

F.1 RoIS Example Usecase with RoSO

Figure 7.1 shows an example of a robot scenario for a robotic reception service. In the RoIS development model, though the robot service developers choose appropriate robotic functional components to compose their robotic systems and services, there is no description of how they can choose such components suitable for their purpose.

In a closed environment, such as a flat 20m-by-20m square entrance lobby space, when a messenger robot finds a person in the environment, it then approaches the person. After identifying the person, the robot looks up if there are messages to the person, and then, if some messages are found, it tells the messages to the person.

Figure F. 1 and the machine readable file RoboticReceptionService.ttl provides a turtle description of an example service description.

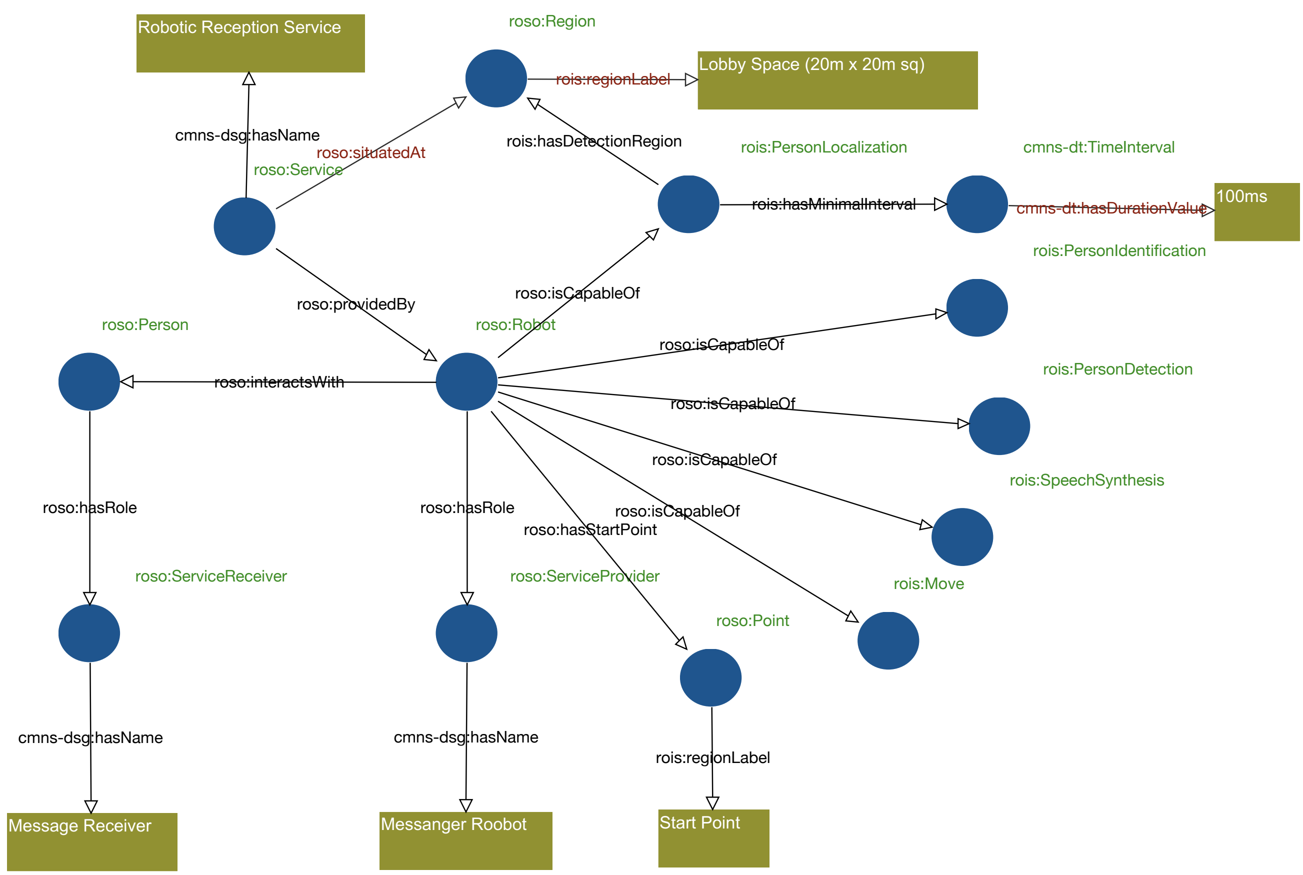


Figure F.1 A service description example of Robotic Reception Service

F.2 Cybernetic Avatar Service with RoSO/RoIS

This subsection describes examples of cybernetic avatar services described with RoSO and RoIS.

A cybernetic avatar is a robot that represents a person and interacts with another person (in case it is also represented by another avatar) to achieve a service scenario. As the operator of an avatar needs to understand the service environment and circumstances of the avatar, the avatar is usually capable of providing video and audio streams captured by itself to the operator and also providing interactive actions via speech and motion.

A cybernetic avatar is not a fully autonomous robot but is partly or totally controlled by a human operator. That is, a service provided with a cybernetic avatar is a sequence of interactions between service recipients and service operators. In simple cases, a service session may be operated by a service operator, but in complex cases, it may be operated by a team of operators. An operator may control multiple avatars at once so that can provide plural service sessions simultaneously.

A cybernetic avatar is implemented as an HRI engine equipped with several HRI components and located in front of service customers, and a service system with operators' consoles is implemented as a client of the HRI engine of the avatar.

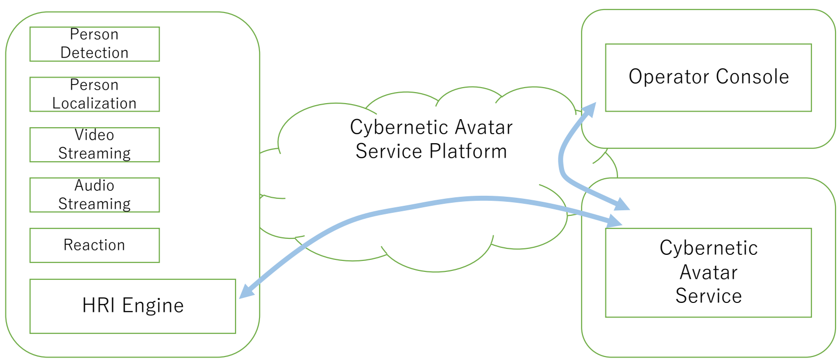


Figure F.2: Cybernetic Avatar Service Environment

The following sections describe 1) a cybernetic avatar composed of a set of RoIS components, 2) an operation console of cybernetic avatars as a client of a set of cybernetic avatar components, 3) a communication platform that connects cybernetic avatars and their consoles, and finally 4) descriptions of cybernetic avatar services with RoSO/RoIS ontologies.

F.2.1 Cybernetic Avatar with RoIS components

A cybernetic avatar provides interaction service for a person by representing its operator. As an operator exists behind the avatar, the avatar itself does not need to be equipped with autonomous functions, but video and audio streams to be sent to operators so that operators can recognize the customers' behaviors and make decisions.

The HRI Engine of a cybernetic avatar is capable of detecting person around it and also localizing the person in the environment.

The CA has audio streaming and video string functions. The operator therefore can connect to camera and mic to receive the environment around the CA. The stream is also used to send the operators (or synthesized) voice to the customer. The CA have reaction component to make motions and also use voice stream that is generated at service console by transforming operators voice in real-time or partly reusing recorded or synthesized voice.

F.2.2 Cybernetic Avatar Service Operator Console

It is expected a hundred of CAs will be deployed; some will be operated by experienced operators to support customers in the experiment field, and others will be operated by remote visitors to participate in the event. The experienced operators will control multiple CAs at once to provide services through CAs. Virtual visitors will control a CA per person as one's avatar participating in the event and representing the operator.

Each CA is controlled through operator console that helps the operator communicate with someone through CA's expression. The operator receives audio and video streams in addition to the locations of the people detected around the CA. The operator chose the CAs motions and send transformed voice. The console is, therefore, a part of a service application that receives sensor events from a PersonDetection component and a PersonLocalization component, actuates the CA through a Reaction component, and also connects to an AudioStreaming component and a VideoStreaming component to communicate with persons in front of the CA.

F.2.3 Cybernetic Avatar Communication Platform

A cybernetic avatar communication platform provides communication function between CAs and service consoles. It manages a lot of CAs in the service environment and manages control sessions between service operators and CAs. The platform is a middleware that enables a service console as a Service Application of the RoIS framework to control a CA as an HRI Engine, and also is a communication infrastructure that handles streaming connections used beneath the RoIS streaming components.

The CA platform for experiments implements its streaming connection upon WebSocket that connects a service console implemented as a browser application with the streaming components on the CAs. Service Application endpoints of RoIS are instantiated on the platform, give control consoles to operators as browser applications, and bind components on CAs as RoIS HRI Engines.

F.2.4 Example: Product explanation at large stores

F.2.4.1 Overview

A store clerk remotely operates a semi-autonomous tele-operated robot (CA) to explain/recommend products to customers who come to the store. Regarding the location-fixed type CA, the CA is installed next to the product to be explained/recommended, and when the customer comes near the CA, the CA automatically plays the prepared explanation/recommendation contents. When a customer verbally asks a question to the CA, the question is transmitted to the clerk who is the tele-operator, and the clerk answers verbally or by selecting prepared answer contents with U/I. Regarding the movable type CA, in addition to the location-fixed type service, it will approach the customer to start the service, guide the customer to the location of the product.

F.2.4.2 Benefit

For the store clerk, since it is possible to respond without going to the site, it is not only possible to save the physical strength to stand all day for explanation and save travel time, and it is not necessary to be in the same store. It is possible to respond from other store or home. It will also be possible to provide services such as complying with the customer's national language from other country.

For customers, it is difficult to ask about products in stores with few store clerks or in stores where store clerks are busy. CA makes easier to ask about products. In addition, it is difficult to find what you want to ask from predetermined question items such as reading product advertisements or interactive digital signage, but with CA you can ask directly verbally, so stress is reduced.

Another advantage is that it can prevent infection such as COVID-19 for both clerks and customers.

F.2.4.3 Deployment Example

[Service from a location-fixed type CA]

A location-fixed type semi-autonomous tele-operated robot (CA) is installed on a shelf next to a product which is to be recommended to customers.

The CA is capable to find customers around the shelf by using a person detection component that can detect up to 10 persons in 150 cm from the CA.

The CA tries to connect to one of the CA's tele-operators when it detects customers within the range.

The CA is also capable to detect the position (direction and distance) of customers within the same detection range so that the CA can turn to the near-by customer (or a group of customers) before starting recommendation.

The CA is not capable of autonomous conversation that means it does not have any facility for speech recognition.

The CA is capable to transmit the customers speech (audio and video signal) to the tele-operator of the CA.

When a customer stops by the CA for more than 0.3 seconds (that requires the location of persons are to be updated 0.1 second frequency), the CA initiates interaction with the customer automatically just playing a pre-defined motion and speech. A reaction component and a speech synthesis component are required on the CA.

After the tele-operator clerk is ready connected, the CA receives commands for reaction and speech from the clerk.

If there aren't any operators assigned, the CA plays pre-recorded recommendation without interacting with the customers.

F.2.4.4 RoSO Description

A service desciption diagram using RoSO and RoIS ontology is shown in Figure F.3 and a machine readable file CyberneticAvatarService.ttl.

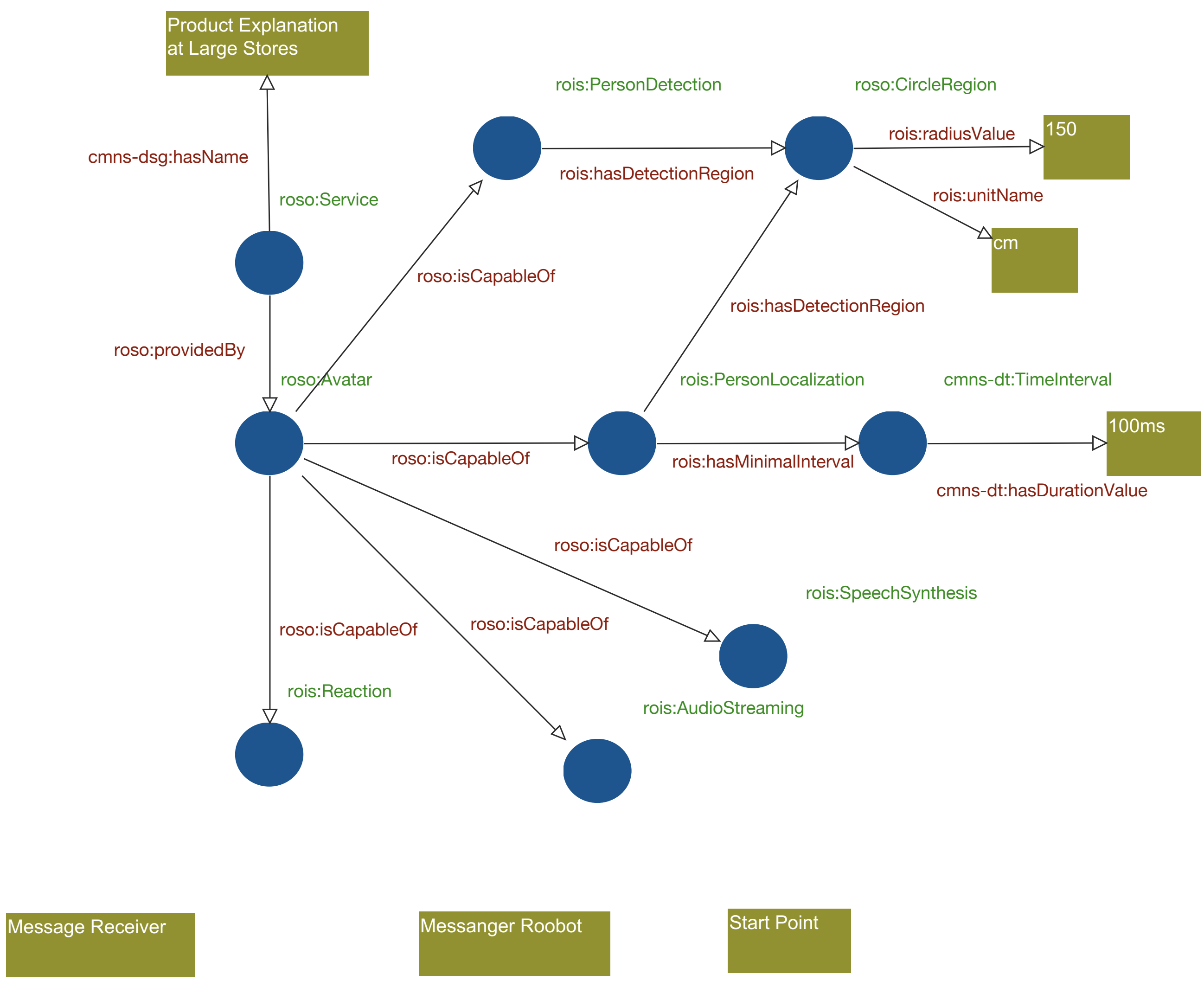


Figure F.3 A service description example of Cybernetic Avatar Service

This page intentionally left blank.