Date: April 2016

OBJECT MANAGEMENT GROUP®

Finite State Machine Component for RTC (FSM4RTC)

V1.0

OMG Document Number: formal/2016-04-01

Standard document URL: http://www.omg.org/spec/FSM4RTC/1.0

Normative Machine Consumable File(s):

http://www.omg.org/spec/FSM4RTC/20150901/ComponentObserver.idl
http://www.omg.org/spec/FSM4RTC/20150901/DataPort.idl
http://www.omg.org/spec/FSM4RTC/20150901/ExtendedFsmService.idl
http://www.omg.org/spec/FSM4RTC/20150901/fsm4rtc.xmi

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

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

LICENSES

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

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

PATENTS

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

GENERAL USE RESTRICTIONS

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

DISCLAIMER OF WARRANTY

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

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

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 109 Highland Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

CORBA®, CORBA logos®, FIBO®, Financial Industry Business Ontology®, FINANCIAL INSTRUMENT GLOBAL IDENTIFIER®, IIOP®, IMM®, Model Driven Architecture®, MDA®, Object Management Group®, OMG®, OMG Logo®, SoaML®, SOAML®, SysML®, UAF®, Unified Modeling Language®, UML®, UML Cube Logo®, VSIPL®, and XMI® are registered trademarks of the Object Management Group, Inc.

For a complete list of trademarks, see http://www.omg.org/legal/tm_list.htm. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

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

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object

Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting

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:

http://www.omg.org/spec

Specifications are organized by the following categories:

Business Modeling Specifications

Middleware Specifications

- **CORBA/IIOP**
- **Data Distribution Services**
- **Specialized CORBA**

IDL/Language Mapping Specifications

Modeling and Metadata Specifications

- **UML, MOF, CWM, XMI**
- **UML Profile**

Modernization Specifications

Platform Independent Model (PIM), Platform Specific Model (PSM), Interface Specifications

- **CORBAServices**
- **CORBAFacilities**

OMG Domain Specifications

CORBA Embedded Intelligence Specifications

CORBA Security Specifications

Signal and Image Processing Specifications

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

OMG Headquarters
109 Highland Avenue
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

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

Typographical Conventions

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

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

Helvetica/Arial – 9 or **10 pt. Bold:** OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier new/Courier – 10 pt. Bold`: Programming Languages

Helvetica/Arial – 10 pt.: Exceptions

Issues

The reader is encouraged to report any technical or editing issues/problems with this by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under documents, Report a Bug/Issue.

1 Scope

This specification defines the following items by extending the RTC specifications:

1. Service interface which provides FSM component meta data including an FSM structure together with appropriate data models.
2. Service interface which provides the current state of the FSM component.
3. Service interface which notifies internal actions of the FSM component including state transitions.
4. Extended RTC::PortService which receives structured event data from outside.
5. Data model to describe structured event data including events with parameters.

2 Conformance

2.1 Changes to RTC Specification

This specification does not modify the adopted RTC specification. It reuses and/or adds functionality on top of the current RTC specification.

2.2 Conformance points

This specification defines the following conformance points:

1. Component Observer (see 7.2.4)
2. Extended FSM Service (see 7.2.5)
3. Data Port Profiles (see 7.2.6)

Conformance with the “FSM4RTC” specification requires conformance with all the mandatory conformance points.

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.

[UML] Object Management Group, OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.5, http://www.omg.org/spec/UML/2.5/Beta1/

[RTC] Robotic Technology Component specification, http://www.omg.org/spec/RTC/1.1/

[SDO] Super distributed Object Specification, http://www.omg.org/spec/SDO/1.1/

4 Terms and Definitions

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

Robotic Technology Component (RTC)

A logical representation of a hardware and/or software entity that provides well-known functionality and services.

Super Distributed Object (SDO)

A logical representation of a hardware device or a software component that provides well-known functionality and services.

Extensible Markup Language (XML)

A markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

XML Metadata Interchange (XMI)

An OMG standard for exchanging metadata information via XML.

State Chart XML (SCXML)

An XML-based markup language which provides a generic state-machine based execution environment based on UML Statecharts.

5 Symbols

There are no special symbols or terms.

6 Additional Information

6.1 Acknowledgements

The following company submitted this specification:

- Honda R&D Co., Ltd.
  Fundamental Technology Research Center
  8-1 Honcho, Wako-shi, Saitama, 351-0188 Japan
  Contact: Makoto Sekiya (makoto_sekiya@n.f.rd.honda.co.jp)

The following company supported this specification:

- National Institute of Advanced Industrial Science and Technology

7 Finite State Machine Component for Robotic Technology Components (FSM4RTC)

7.1 General

According to the RTC specification, a Finite State Machine (FSM) component can be defined as Figure 7.1. However, access methods and interfaces to ensure interoperability of the FSM component are not defined in the specification.

Thus, tools and other RTCs are not able to get notifications, the current state, and the structure from the FSM component in an interoperable way. In addition to that, the definition of ports in the RTC specification is not sufficient to provide RTCs with the standard data communication method.

Figure 7.2 shows a use case as a solution. **ComponentObserver** gets notifications from the FSM components. **ExtendedFsmService** is an interface for setting/getting the current state and an FSM structure data model which contains states and transition rules of the FSM. Using **DataPort**, other RTCs can send events with data to the FSM components.

This specification uses **SDOService** and key/values properties of **PortProfile** and **ConnectorProfile** to extend the RTC specification so that components conform to the RTC specification can communicate both existing RTCs and extended RTCs.

The PIM for the above interface is specified in sub clause 7.2 and the PSM is specified in sub clause 7.3.

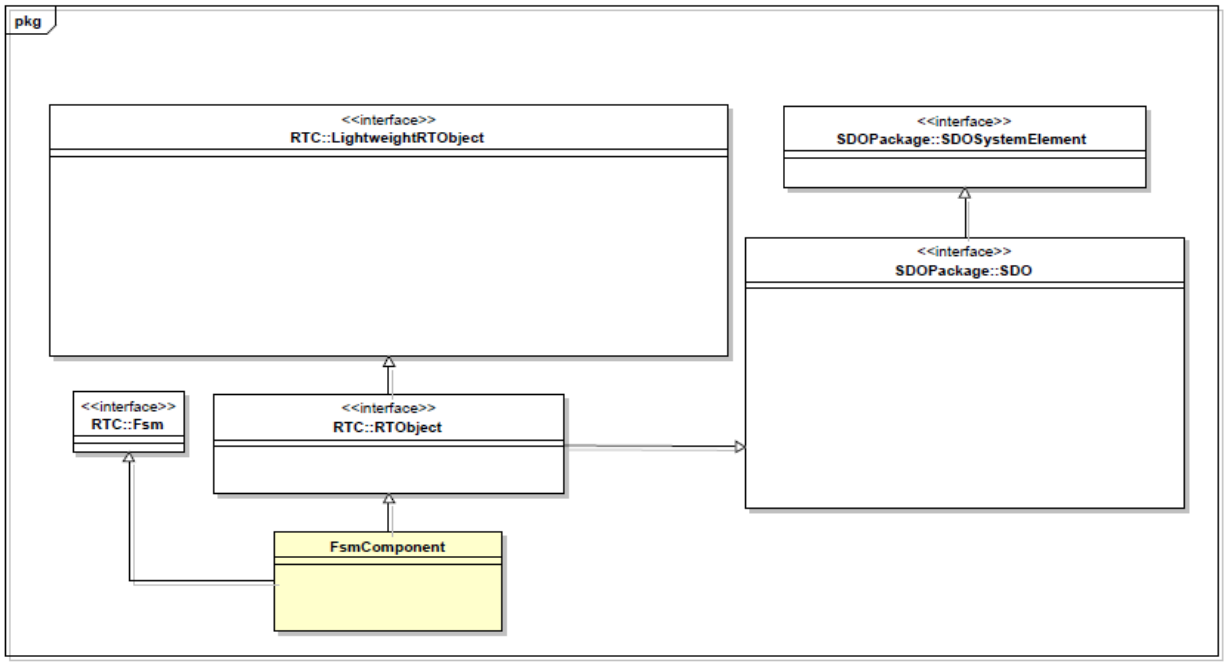

Figure 7.1 – An example declaration of FSM component (non-normative)

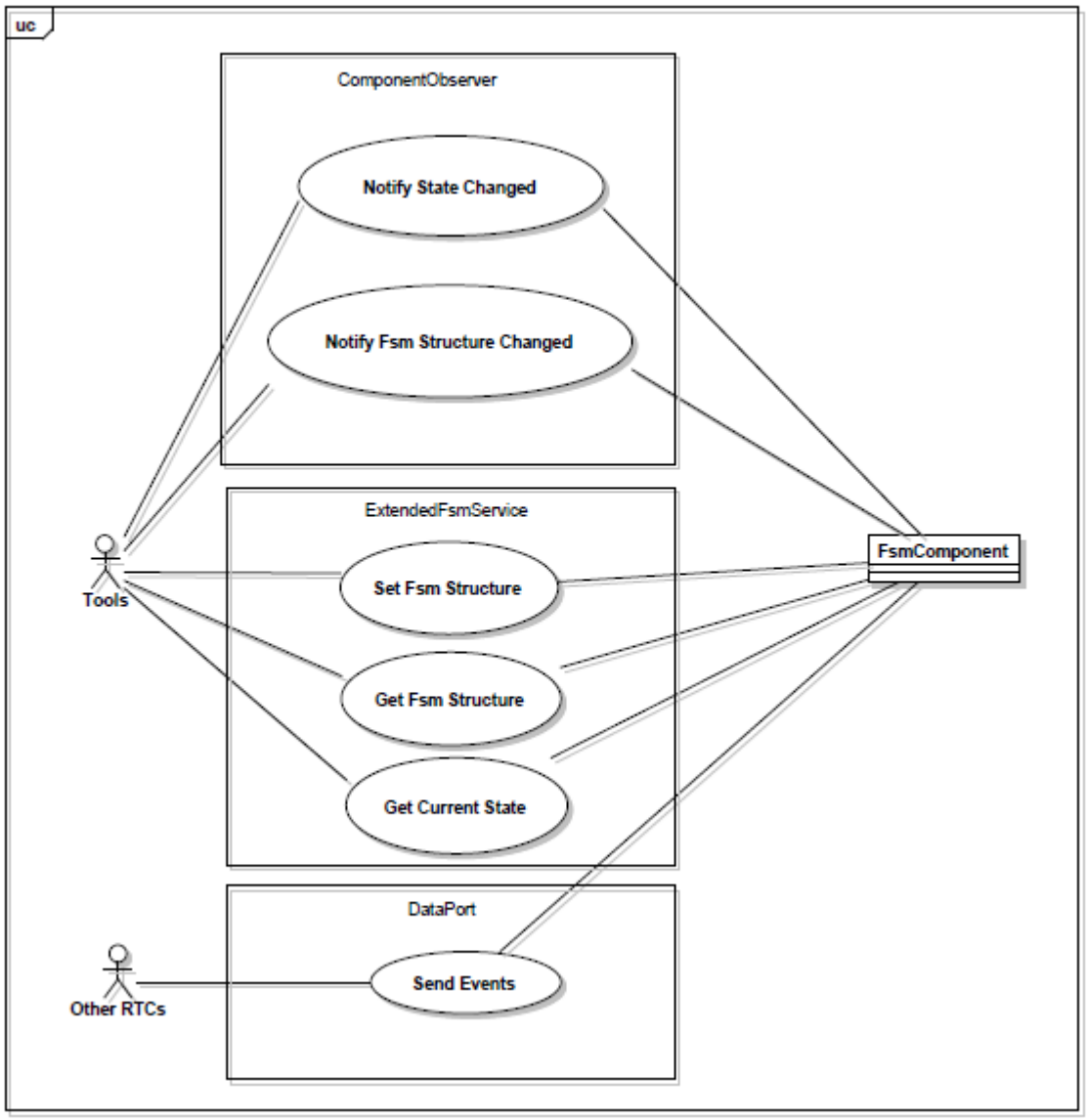

Figure 7.2 – Proposed use case of FSM component (non-normative)

7.2 Platform Independent Model (PIM)

7.2.1 Overview

This sub clause specifies the PIM for service interfaces and data models. At first, in 7.2.3, basic types are introduced. Sub clause 7.2.4, "**ComponentObserver**" describes the PIM for the interface and data model, which are used to receive notifications from RTCs. Sub clause 7.2.5, "**ExtendedFsmService**" defines the interfaces and data models to access and manipulate the structure of the FSM. Sub clause 7.2.6, "**Data Port**" introduces **DataPushService** and **DataPullService** interfaces realize push/pull types of data communication models and properties specify the detail parameters for data communication. Figure 7.3 shows an overview UML notation of the PIM.

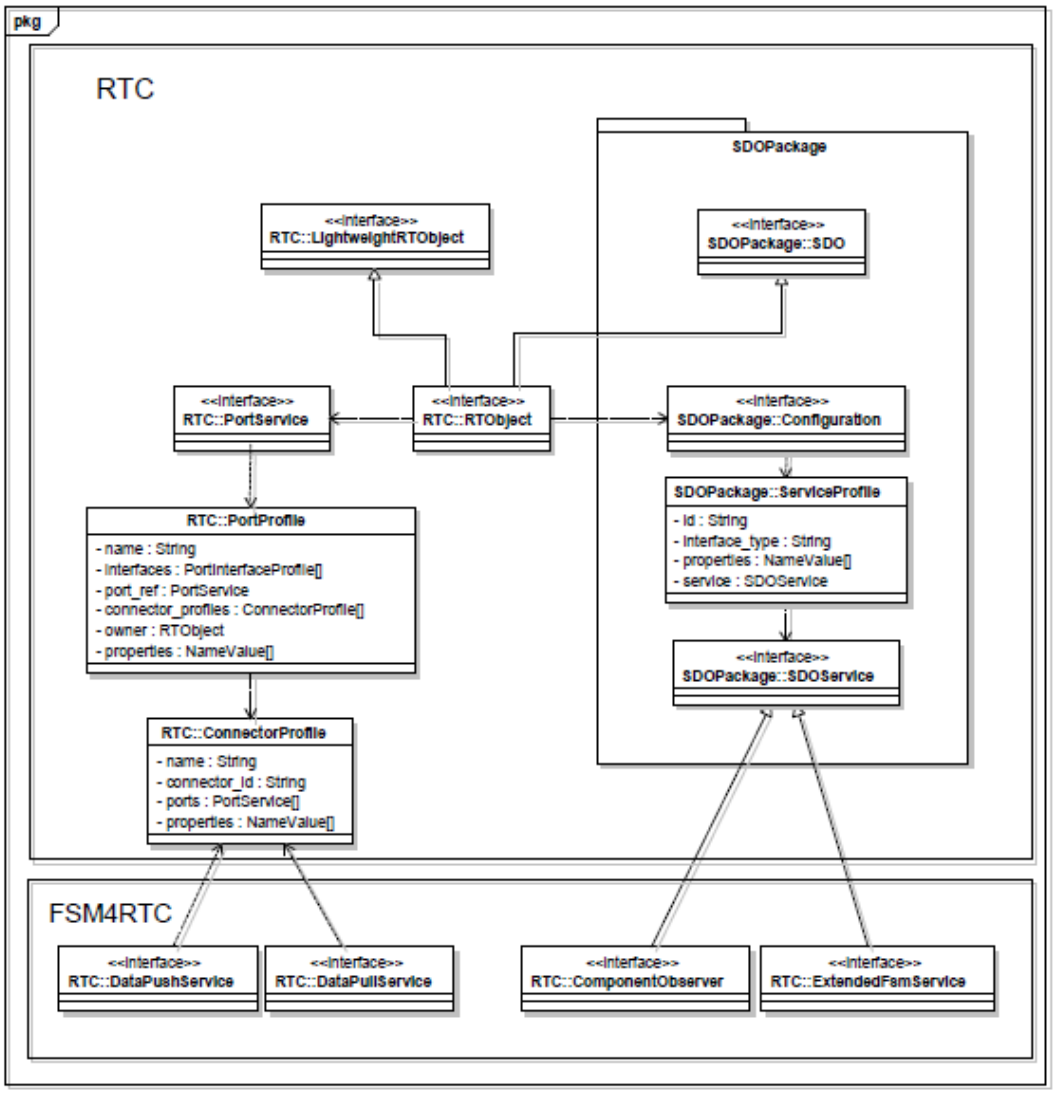

Figure 7.3 – Overview of FSM4RTC PIM

7.2.2 Format and Conventions

This specification uses UML diagrams [UML] to show classes and their relationships. All classes are part of the RTC package extended by FSM4RTC (Finite State Machine Component for RTC) specification. If, in a UML diagram, a class's attribute and operation compartments are suppressed, then this class is elaborated elsewhere. In this case, the diagram might also not show all of the class' associations. However, if a class is shown to have only an attribute or an operation compartment, then this signifies that the not-shown compartment is empty. I.e., if a class is shown with an attribute but no operation compartment, then the class does not have any operations.

7.2.3 Basic Types

This specification reuses the types from [UML], [SDO], [RTC]. These reused types are described in this sub clause.

7.2.3.1 String [UML]

Description

The **String** primitive type represents a character string that can be used for any character set.
String is an instance of **PrimitiveType** [UML].

7.2.3.2 Octet [RTC]

Description

The **Octet** primitive type, a specialization of Integer primitive type, is an unsigned integer within range [0, 255].
Octet is an instance of **PrimitiveType** [UML].

7.2.3.3 ReturnCode_t [RTC]

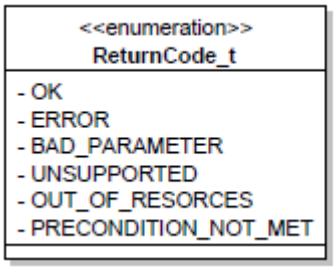

Figure 7.4 – ReturnCode_t

Description

A number of operations in this specification will need to report potential error conditions to their clients. This task shall be accomplished by means of operation "return codes" of type **ReturnCode_t**.

Operations in the PIM that do not return a value of type **ReturnCode_t** shall report errors in the following ways, depending on their return type:

- If an operation normally returns a positive numerical value (such as **get_rate**, see Section 5.2.2.6.4 of [RTC]), it shall indicate failure by returning a negative value.
- If an operation normally returns an object reference (such as **RTObject::get_component_profile**, see Section 5.4.2.2.1 of [RTC]), it shall indicate failure by returning a nil reference.

Attributes

OK	Enumeration to specify the operation completed successfully.
ERROR	Enumeration to specify that the operation failed with a generic, unspecified error.
BAD_PARAMETER	Enumeration to specify that the operation failed because an illegal argument was passed to it.
UNSUPPORTED	Enumeration to specify that the operation is unsupported by the implementation (e.g., it belongs to a compliance point that is not implemented).
OUT_OF_RESOURCES	Enumeration to specify that the target of the operation ran out of the resources needed to complete the operation.
PRECONDITION_NOT_MET	Enumeration to specify that a pre-condition for the operation was not met.

Associations

No additional associations.

7.2.3.4 NameValue [SDO]

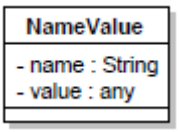

Figure 7.5 – NameValue

Description

NameValue is a pair of a name and its value defined in the sub clause 7.3.2 of [SDO].

Attributes

name: String	A name of a value.
Value: any	The value of the name.

Associations

No additional associations.

7.2.4 ComponentObserver

This sub clause specifies **ComponentObserver**. As Figure 7.6 shows, **ComponentObserver** is an SDO service which notifies status update of an RTC to other tools or RTCs. Kinds of updated status are defined as **RTC::StatusKind**.

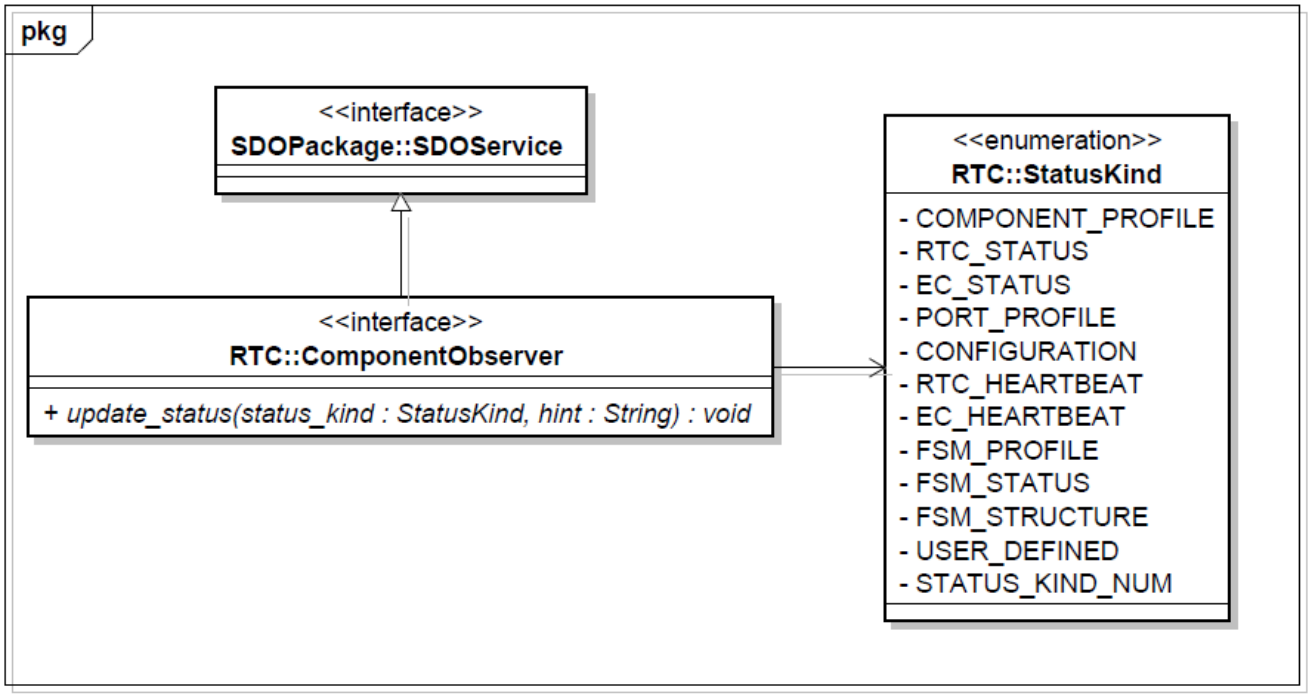

Figure 7.6 – Overview of ComponentObserver PIM

7.2.4.1 StatusKind

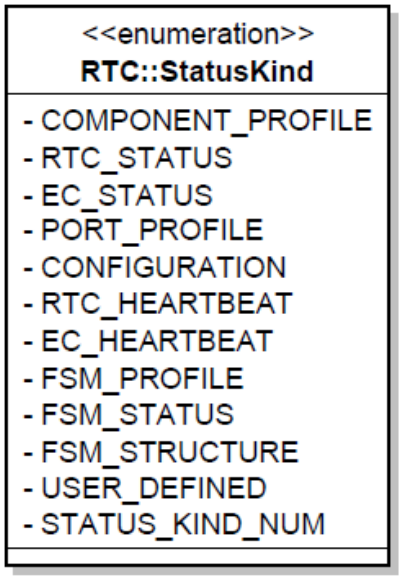

Figure 7.7 – StatusKind

Description

StatusKind is an enumeration type to classify updated status in target RTC.

Attributes

COMPONENT_PROFILE	Enumeration to specify that the target component's RTC::ComponentProfile has been changed.
RTC_STATUS	Enumeration to specify that the target component's status has been changed.
EC_STATUS	Enumeration to specify that the target component's status of execution contexts has been changed.
PORT_PROFILE	Enumeration to specify that the target component's status of ports has been changed.
CONFIGURATION	Enumeration to specify that the target component's configuration has been changed.
RTC_HEARTBEAT	Enumeration to notify that the target component is alive.
EC_HEARTBEAT	Enumeration to notify that the target execution context is alive.
FSM_PROFILE	Enumeration to specify that the target component's FSM profile has been changed.
FSM_STATUS	Enumeration to specify that the target component's FSM status has been changed.

FSM_STRUCTURE	Enumeration to specify that the target component's FSM structure has been changed.
USER_DEFINED	Enumeration to specify a user defined notification.
STATUS_KIND_NUM	Enumeration to specify the number of attributes.

Associations

No additional associations.

7.2.4.2 ComponentObserver Interface

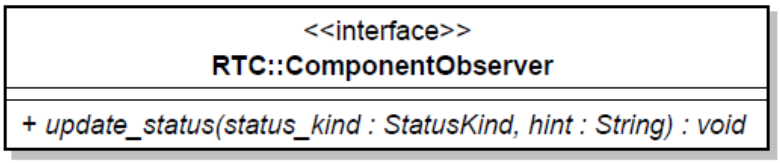

Figure 7.8 – ComponentObserver

Description

ComponentObserver is an interface to notify various status changes in RTC to others. **ComponentObserver** is attached to a target RTC/SDO as an SDO service, and if an RTC/SDO's status changes, a kind of changed status and its hints are notified to observers. A non-normative assumed usage is shown as Figure 7.9.

Operations

update_status(in StatusKind status_kind, in String hint): void	This operation notifies a status update. The **status_kind** indicates the kind of updated status, and the **hint** give some hint about updated status.

Hints

The following hints are defined in this specification to realize interoperability of information from **ComponentObserver**. Implementation may support a part of the hints as necessary.

COMPONENT_PROFILE	The comma separated name of changed profile's key (ex. "instance_name, type_name").
RTC_STATUS	INACTIVE:Execution Context ID (ex. "INACTIVE:1002") ACTIVE:Execution Context ID (ex. "ACTIVE:1002") ERROR:Execution Context ID (ex. "ERROR:1002")

EC_STATUS	ATTACHED:Execution Context ID (ex. “ATTACHED:1002”) DETACHED:Execution Context ID (ex. “DETACHED:1002”) RATE_CHANGED:Execution Context ID (ex. “RATE_CHANGED:1002“) STARTUP:Execution Context ID (ex. “STARTUP:1002“) SHUTDOWN:Execution Context ID (ex. SHUTDOWN:1002“)
PORT_PROFILE	ADD:port name (ex. “ADD:velocity“) REMOVE:port name (ex. “REMOVE:velocity“) CONNECT:port name (ex. “CONNECT:velocity“) DISCONNECT:port name (ex. “DISCONNECT:velocity“)
CONFIGURATION	UPDATE_CONFIGSET:configuration set's name (ex. “UPDATE_CONFIGSET:default“) UPDATE_PARAMETER:<config set's name>.<config param's key> (ex. “UPDATE_ PARAMETER:default.key“) SET_CONFIG_SET:config set's name (ex. “SET_CONFIG_SET:default“) ADD_CONFIG_SET:config set's name (ex. “ADD_CONFIG_SET:option“) REMOVE_CONFIG_SET:config set's name (ex. “REMOVE_CONFIG_SET:option“) ACTIVATE_CONFIG_SET:config set's name (ex. “ACTIVATE _CONFIG_SET:option“)
FSM_STATUS	Name of the current state
FSM_STRUCTURE	Name of the FSM
USER_DEFINED	User defined text

Attributes

No additional attributes.

Associations

No additional associations.

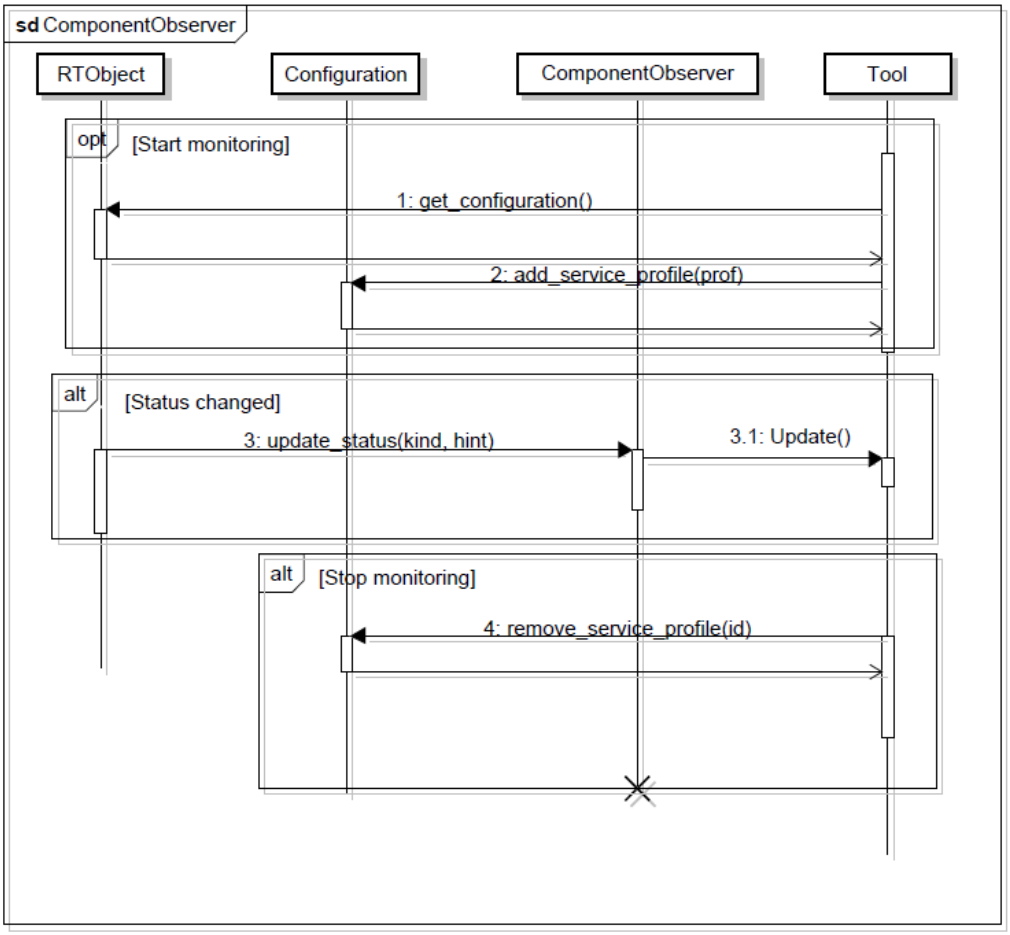

Figure 7.9 – Sequence for adding ComponentObserver (non-normative)

7.2.5 ExtendedFsmService

This sub clause specifies **ExtendedFsmService**. As Figure 7.10 shows, **ExtendedFsmService** is an SDO service. With **ExtendedFsmService**, a RTC can provide extended interfaces to get the current status of the FSM and set/get the structure definition data model of the FSM for other tools and RTCs.

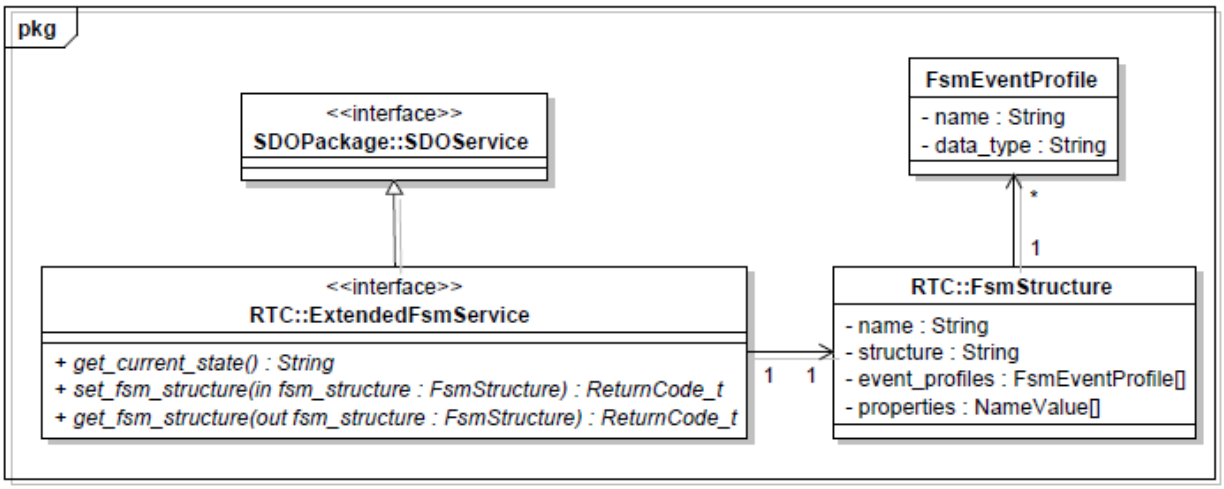

Figure 7.10 – Overview of ExtendedFsmService PIM

7.2.5.1 FsmEventProfile

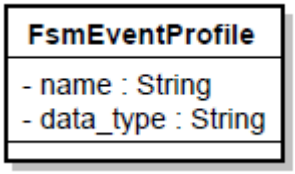

Figure 7.11 – FsmEventProfile

Description

FsmEventProfile is a data model to bind the name of event and its data type of the FSM component.

Attributes

name: String	A name of the FSM
data_type: String	The type of the event data as **CORBA::RepositoryID**

Associations

No additional associations.

7.2.5.2 FsmStructure

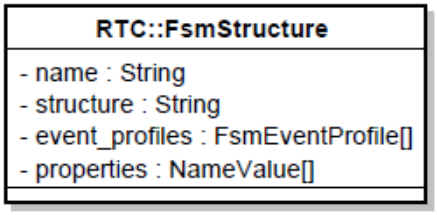

Figure 7.12 – FsmStructure

Description

FsmStructure is a data model to describe a structure of an FSM of the FSM component. **FsmStructure** is used to specify the name and description format of an FSM. Detail usage is explained in 7.2.5.3, "ExtendedFsmService interface."

Attributes

name: **String**	A name of the FSM
structure: **String**	A string formatted description of the structure of the FSM
event_profiles: **FsmEventProfile[]**	An array of **FsmEventProfile**
properties: **NameValue**	Additional properties of the **FsmStructure**

Properties

Names of properties of **FsmStructure** have the dot-separated prefix "fsm_structure".

Description format property of the structure of the FSM	The format of the **structure** attribute
name	fsm_structure.format
value	The specified format name of structure (ex. scxml, xmi).

Associations

No additional associations.

7.2.5.3 ExtendedFsmService interface

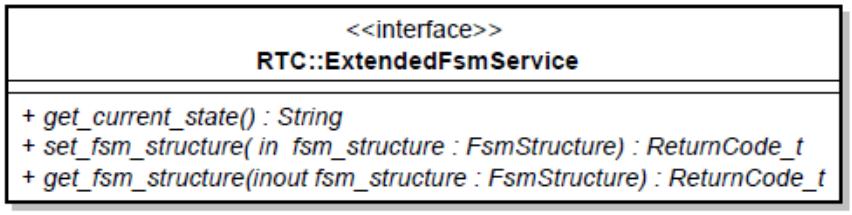

Figure 7.13 – ExtendedFsmService

Description

FsmStructure is a data model to describe a structure of an FSM of the FSM component. **FsmStructure** is used to specify the name and description format of an FSM. Detail usage is explained in 7.2.5.3, "ExtendedFsmService interface."

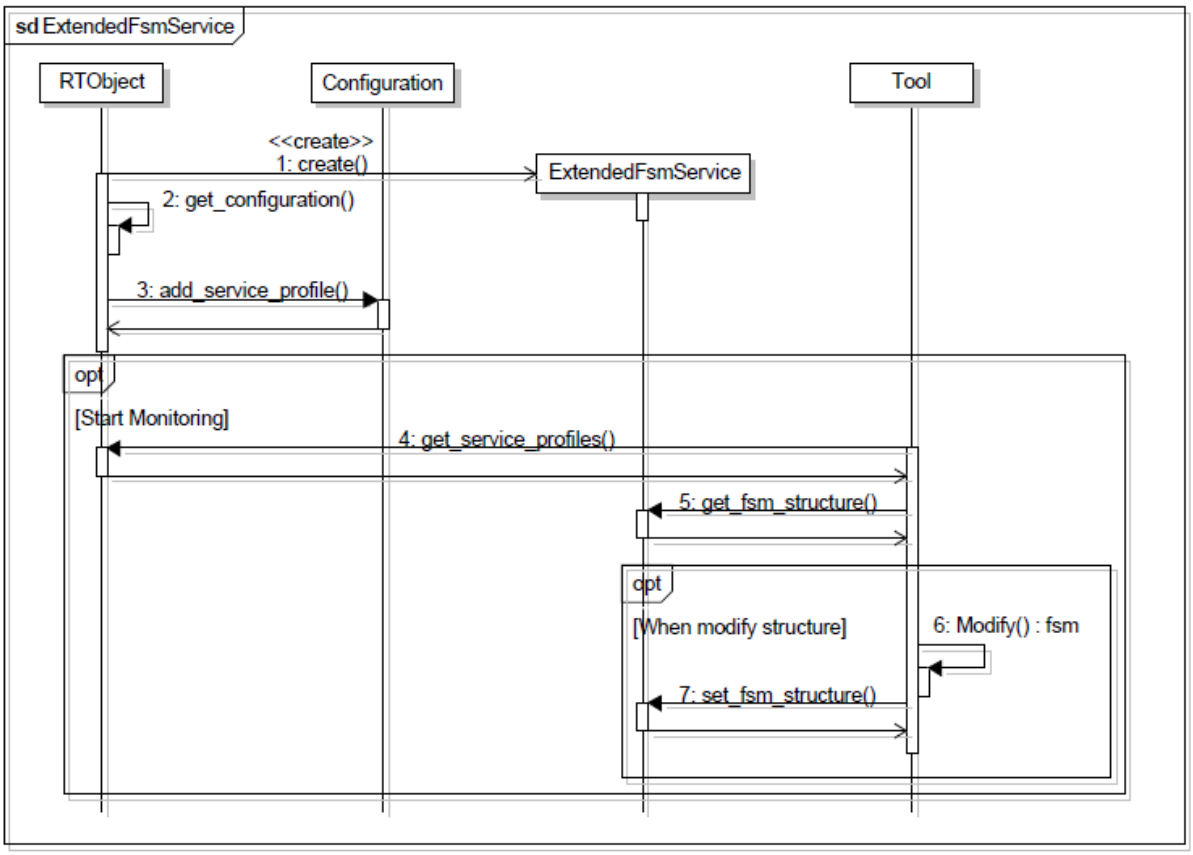

Figure 7.14 – Sequence for creating and using ExtendedFsmService (non-normative)

Description

ExtendedFsmService is an interface to set and get the structure of an FSM in the FSM component from others. This is created by a target RTC as an SDO service and added to its own configuration. A non-normative usage is shown as Figure 7.14. Tools get the reference of an **ExtendedFsmService** via configuration of the target RTC. After getting the **ExtendedFsmService**, tools can get and set the **FsmStructure** of the target RTC.

Operations

get_current_state(): String	This operation returns the current state of an FSM in the target FSM component.
get_fsm_structure(out fsm_structure:FsmStructure): ReturnCode_t	This operation returns the structure of an FSM in the target FSM component. **ExtendedFsmService** returns the name, structure with format specified by fsm_structure.format and **EventProfiles**. RTCs may return UNSUPPORTED if this operation is not implemented.
set_fsm_structure(in fsm_structure:FsmStructure): ReturnCode_t	This operation sets an **FsmStructure** to the target component. Then the target component reconfigures its FSM structure such as transition rules according to the values of the given fsm_structure. RTCs may return UNSUPPORTED if this operation is not implemented.

Attributes

No additional attributes.

Associations

No additional associations.

7.2.6 Data Port

RTC specification provides the definition of **PortService** for RTCs as an interface to communicate each other. As Figure 7.15 shows, however, **PortService** doesn't provide the method to send and receive a certain data type between RTCs. Thus, this specification adds the following data and service models for that purpose.

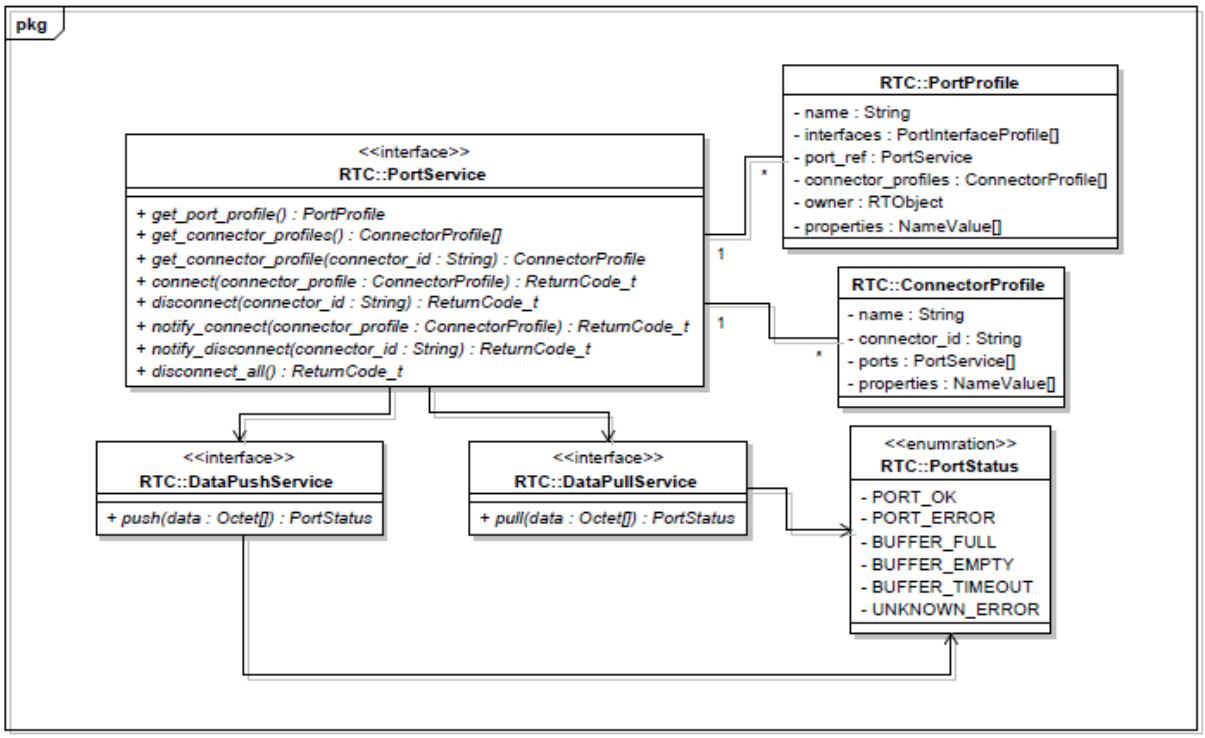

Figure 7.15 – Overview of DataPushService and DataPullService PIM

In the FSM4RTC PIM, two types of communication models are assumed. One is “Sender-push” model and the other is “Receiver-pull” model (Figure 7.16). Figure 7.17 shows how interfaces and data models collaborate to realize these communication models. As Figure 7.17, in the “Sender-push” model, an out port writes data to the buffer of a connector. And then the data is pushed to the buffer of **DataPushService**. Finally an in port reads the data from **DataPushService**. On the other hand, in the “Receiver-pull” model, when an in port calls “read,” the data written by an out port to the buffer of **DataPullService** is pulled from a connector and returned to the in port. “Receiver-pull” model is used to minimize the network communications between senders and receivers by pulling the data when it’s required.

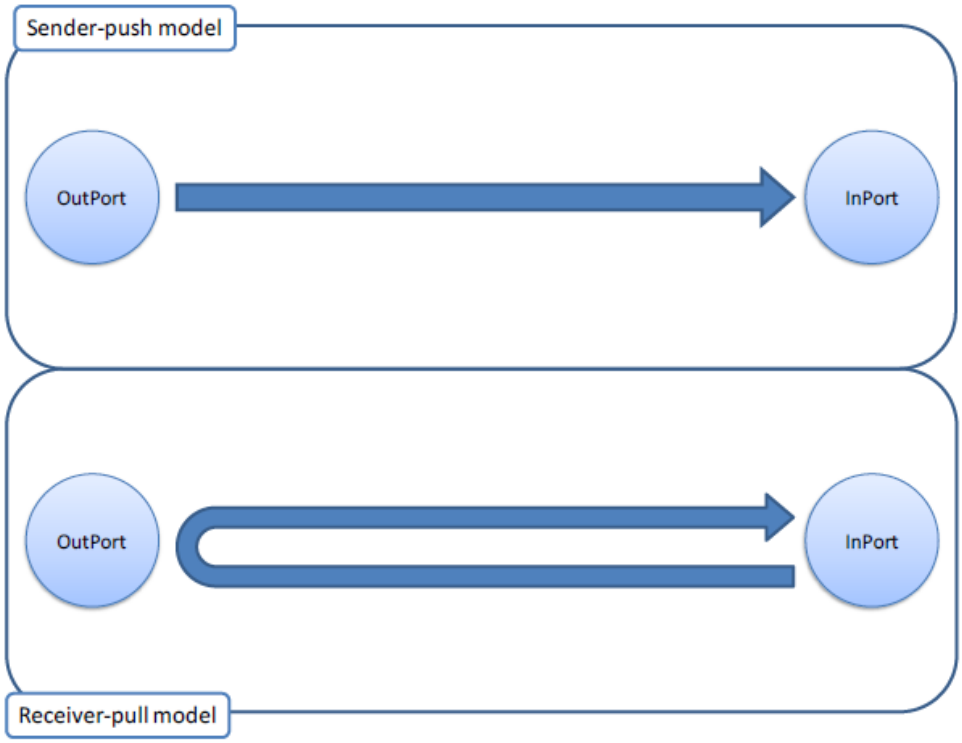

Figure 7.16 – Communication model of data port (non-normative)

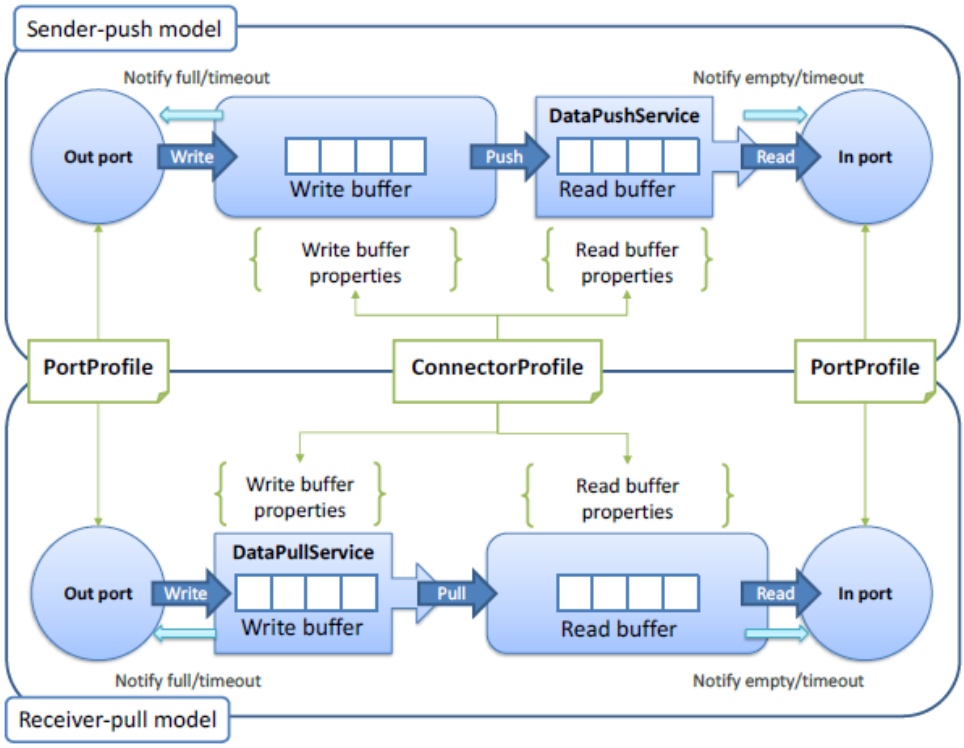

Figure 7.17 – Concept model of data port (non-normative)

7.2.6.1 PortStatus

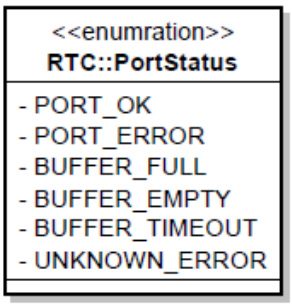

Figure 7.18 — PortStatus

Description

PortStatus is an enumeration type to classify result of operations of **DataPushService** and **DataPullService**.

Attributes

PORT_OK	Enumeration to specify that the result of an action of the data port has been success.
PORT_ERROR	Enumeration to specify that the result of an action of the data port has been failed.
BUFFER_FULL	Enumeration to notify that the buffer of the data port is full.
BUFFER_EMPTY	Enumeration to notify that the buffer of the data port is empty.
BUFFER_TIMEOUT	Enumeration to notify that the write or read from buffer of the data port is timeout.
UNKNOWN_ERROR	Enumeration to specify that the result of an action of the data port has been failed with unknown error.

Associations

No additional associations.

7.2.6.2 PortProfile [RTC]

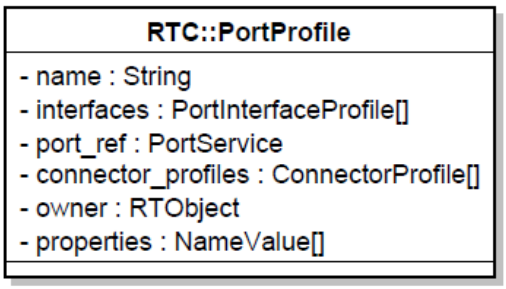

Figure 7.19 – PortProfile

Description

PortProfile is defined in [RTC] describe profiles of a port of an RTC. This specification extends **PortProfile** using the properties attribute as follows. These properties are used to declare supported types of communications of the port.

Properties

Properties of **PortProfile** are used to declare supported types of data ports provided by **PortService**. Names of properties have the dot-separated prefix "dataport". Each property may have comma-separated multiple values. This section defines the minimum set of values of each property to realize interoperability among RTCs. Implementations may support additional values for each property. For example, dataport.interface_type property of an implementation which supports DDS interface includes "dds".

Dataflow type property

Property to define supported data communication models.

name	type	value	description
dataport.dataflow_type	string	push	If this value exists, sender-push model is supported.
		pull	If this value exists, receiver-pull model is supported.

IO mode property

Property to define supported IO modes to write data.

name	type	value	description
dataport.io_mode	string	block	If this value exists, block mode is supported. In block mode, write method of an out port is blocked until the data has been pushed to **DataPushService**.
		nonblock	If this value exists, nonblock mode is supported. In nonblock mode, write method of an out port returns immediately.

Data type property

Property to define the data type used in data ports. **PortService** sets the same data type for all provided data ports.

name	type	value	description
dataport.data_type	string	name of a type	The data type used between connected ports.

Interface type property

Property to define the interface type(s) of a data port.

name	type	value	description
dataport.interface_type	string	name of a type	The name of a supported interface type.

Marshaling type property

Property to define the supported marshaling type(s) of data.

name	type	value	description
dataport.marshaling_type	string	name of a type	The name of a supported marshaling type.

Timestamp policy property

Property to define the supported timestamp policies.

name	type	value	description
dataport.timestamp_policy	string	on_write	If this value exists, a timestamp can be set when an out port writes data.
		on_send	If this value exists, a timestamp can be set before data is pushed to **DataPushService** or pulled from **DataPullService**.
		on_received	If this value exists, a timestamp can be set after data is pushed to **DataPushService** or pulled from **DataPullService**.
		on_read	If this value exists, a timestamp can be set when an in port reads data.
		none	If this value exists, RTCs do not set any timestamp.

Write buffer length property

Property to define the default length of the write buffer.

name	type	value	description
dataport.write-buffer-length	string	integer	A positive integer to define the length of the write buffer [byte].

Write buffer full policy property

Property to define the supported policies when the write buffer is full.

name	type	value	description
dataport.write.buffer.full_policy	string	overwrite	If this value exists, overwrite policy is supported. As overwrite policy, the oldest data is over written when the write buffer is full.
		do_nothing	If this value exists, do_nothing policy is supported. As do_nothing policy, data is not written when the write buffer is full.
		block	If this value exists, block policy is supported. As block policy, writing to the write buffer is blocked until the write buffer is available.

Write buffer timeout property

Property to define default timeout for block policy of the write buffer.

name	type	value	description
dataport.write.buffer.timeout	string	integer	An integer to define the timeout value of blocking [s]

Read buffer length property

Property to define the default length of the read buffer.

name	type	value	description
dataport.read.buffer.length	string	integer	A positive integer to define the length of the read buffer [byte].

Read buffer empty policy property

Property to define the supported policies when the read buffer is empty.

name	type	value	description
dataport.read.buffer.empty_policy	string	read_back	If this value exists, read_back policy is supported. As read_back policy, the read method of an in port returns the last data when the read buffer is empty.
		do_nothing	If this value exists, do_nothing policy is supported. As do_nothing policy, the read method of an in port returns nothing when the read buffer is empty.
		block	If this value exists, block policy is supported. As block policy, the read method of an in port blocks until the read buffer is available.

Read buffer timeout property

Property to define the default timeout for block policy of the read buffer.

name	type	value	description
dataport.read.buffer.timeout	string	string	Timeout of blocking [s]

Read buffer queue policy property

Property to define the supported queue policies of the read buffer.

name	type	value	description
dataport.read.buffer.queue_policy	string	all	If this value exists, all policy is supported. As all policy, all queued data in the read buffer is read at once.
		fifo	If this value exists, fifo policy is supported. As fifo policy, queued data in the read buffer is read with FIFO order.
		new	If this value exists, new policy is supported. As new policy, the latest data in the read buffer is read.

7.2.6.3 ConnectorProfile [RTC]

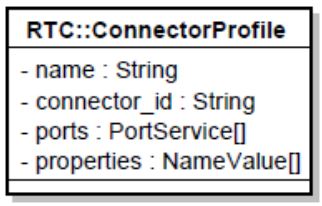

Figure 7.20 – ConnectorProfile

Description

ConnectorProfile is defined in [RTC] to contain information for connecting the ports of collaborating RTCs. This specification extends **ConnectorProfile** using the **properties** attribute as follows. These properties are used to direct a port to provide the interface with specified configuration. If the configuration is acceptable for the port, then an instance of required interface is created and the **PortService::connect** operation shall return **ReturnCode_t::OK**. If the port is unable to provide the interface, the **PortService::connect** operation shall return **ReturnCode_t::BAD_PARAMETER** (Figure 7.21). The acceptable configurations are defined as properties of **PortProfile** (sub clause 7.2.6.2).

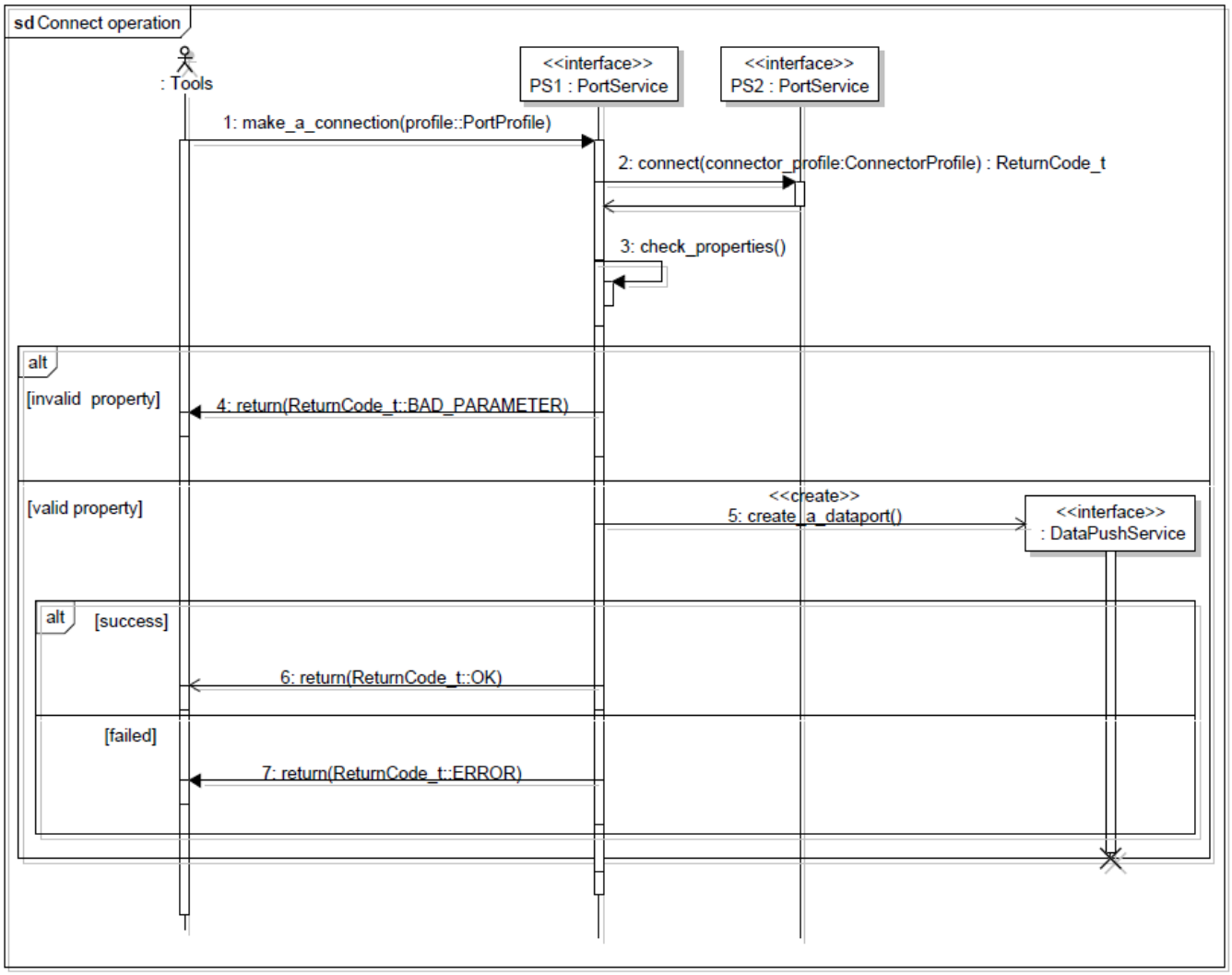

Figure 7.21 – Workflow of connection operations (non-normative)

Properties

Properties of **ConnectorProfile** are used to request **PortService** to provide a specific type of data port between RTCs. Names of properties have the dot-separated prefix "dataport". Each property must have a single value.

Dataflow type property

Property to specify the requested data communication model.

name	type	value	description
dataport.dataflow_type	string	push	Specified if sender-push model is requested.
		pull	Specified if receiver-pull model is requested.

IO mode property

Property to specify the requested IO mode to push or pull data.

name	type	value	description
dataport.io_mode	string	block	Specified if block mode is requested.
		nonblock	Specified if nonblock mode is requested.

Interface type property

Property to specify the requested interface type of a data port.

name	type	value	description
dataport.interface_type	string	name of a type	The name of a requested interface type. If the requested interface is not supported by the target ports, connect operations shall fail.

Marshaling type property

Property to specify the requested marshaling type of data.

name	type	value	description
dataport.marshaling_type	string	name of a type	The name of a requested marshaling type. If the requested marshaling type is not supported by the target ports, connect operations shall fail.

Timestamp policy property

Property to specify the requested timestamp policy.

name	type	value	description
dataport.timestamp_policy	string	on_write	To request that a timestamp can be set when the out port writes data.
		on_send	To request that a timestamp can be set before data is pushed to **DataPushService** or pulled from **DataPullService**.
		on_received	To request that a timestamp can be set after data is pushed to **DataPushService** or pulled from **DataPullService**.
		on_read	To request that a timestamp can be set when an in port reads data.
		none	To request that RTCs do not set any timestamp.

Write buffer length property

Property to specify the requested length of the write buffer.

name	type	value	description
dataport.write.buffer.length	string	integer	Requested length of the write buffer [byte].

Write buffer full policy property

Property to specify the requested policy when the write buffer is full.

name	type	value	description
dataport.write.buffer.full_policy	string	overwrite	Specified if overwrite policy is requested.
		do_nothing	Specified if do_nothing policy is requested.
		block	Specified if block policy is requested.

Write buffer timeout property

Property to specify default timeout for block policy of the write buffer.

name	type	value	description
dataport.write.buffer.timeout	string	integer	Request to set timeout of blocking as specified value[s].

Read buffer length property

Property to specify the default length of the read buffer.

name	type	value	description
dataport.read.buffer.length	string	string	Requested length of the read buffer [byte].

Read buffer empty policy property

Property to specify the supported policies when the read buffer is empty.

name	type	value	description
dataport.read.buffer.empty_policy	string	read_back	Specified if read_back policy is requested.
		do_nothing	Specified if do_nothing policy is requested.
		block	Specified if block policy is requested.

Read buffer timeout property

Property to specify the default timeout for block policy of the read buffer.

name	type	value	description
dataport.read.buffer.timeout	string	integer	Request to set timeout of blocking as specified value [s].

Read buffer queue policy property

Property to specify the supported queue policies of the read buffer.

name	type	value	description
dataport.read.buffer.queue_policy	string	all	Specified if all policy is requested.
		fifo	Specified if fifo policy is requested.
		new	Specified is new policy is requested.

FSM event name property

Property to bind an event name and a data port.

name	type	value	description
dataport.fsm_event_name	string	name of an event	The name of an event bound with the data port.

7.2.6.4 DataPushService Interface

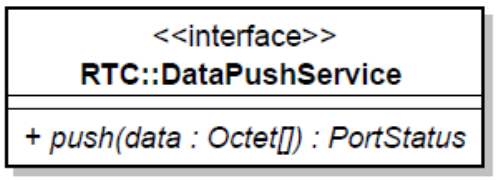

Figure 7.22 – DataPushService

Description

DataPushService is an interface to push an array of **Octet** to the target port with a specified binary format such as Common Data Representation (CDR) format. Figure 7.23 shows a non-normative example of a sequence diagram to create and use **DataPushService**.

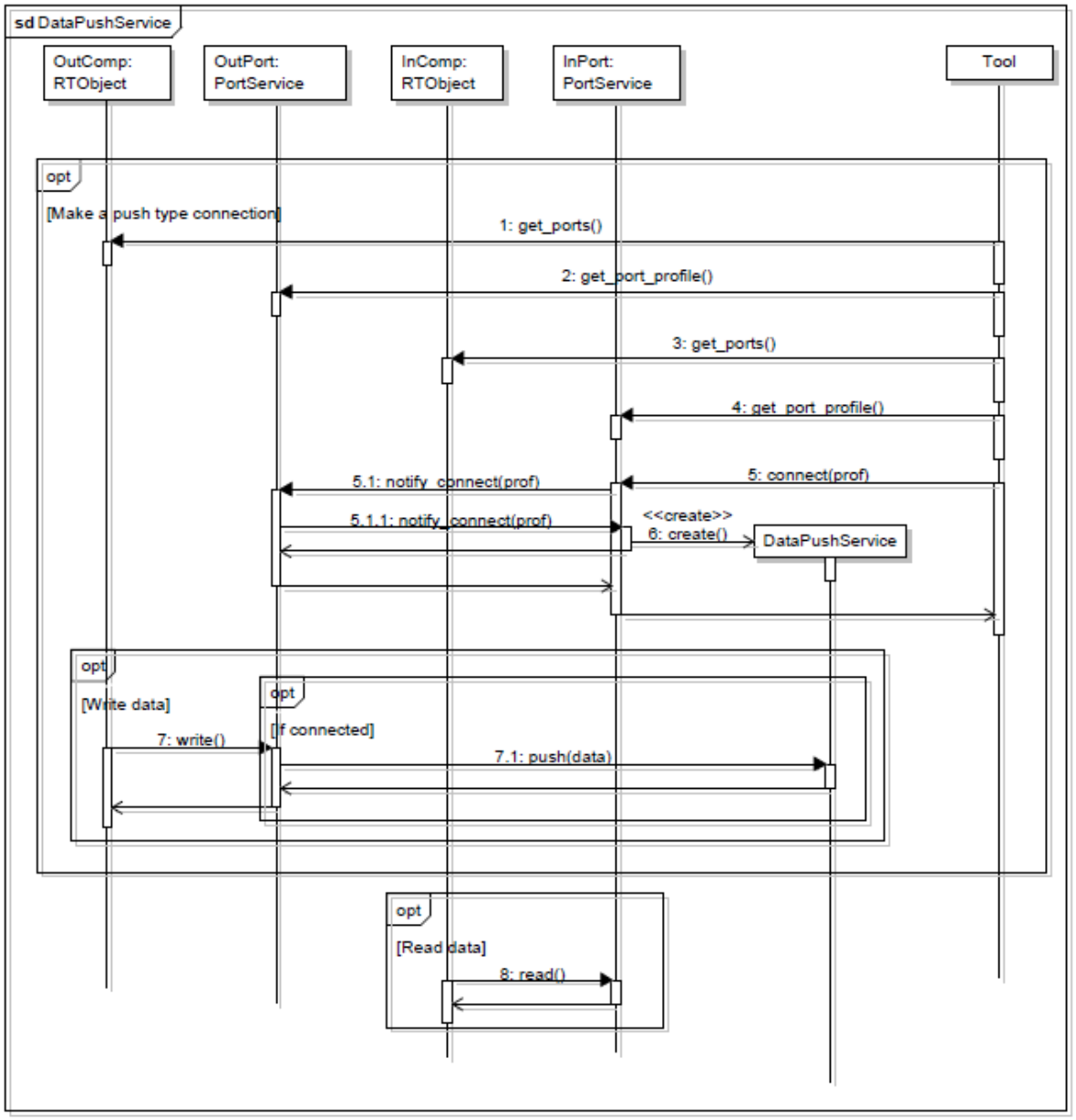

Figure 7.23 – Sequence for creating and using DataPushService (non-normative)

Operations

push(in Octet[] data: PortStatus	This operation pushes an array of **Octet** to the target port with a specified binary format.

Attributes

No additional attributes.

Associations

No additional associations.

7.2.6.5 DataPullService Interface

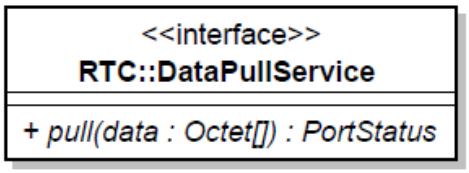

Figure 7.24 – DataPullService

Description

DataPullService is an interface to pull an array of **Octet** from the target port with a specified binary format such as Common Data Representation (CDR) format. Figure 7.25 shoes a non-normative example of a sequence diagram to create and use **DataPullService**.

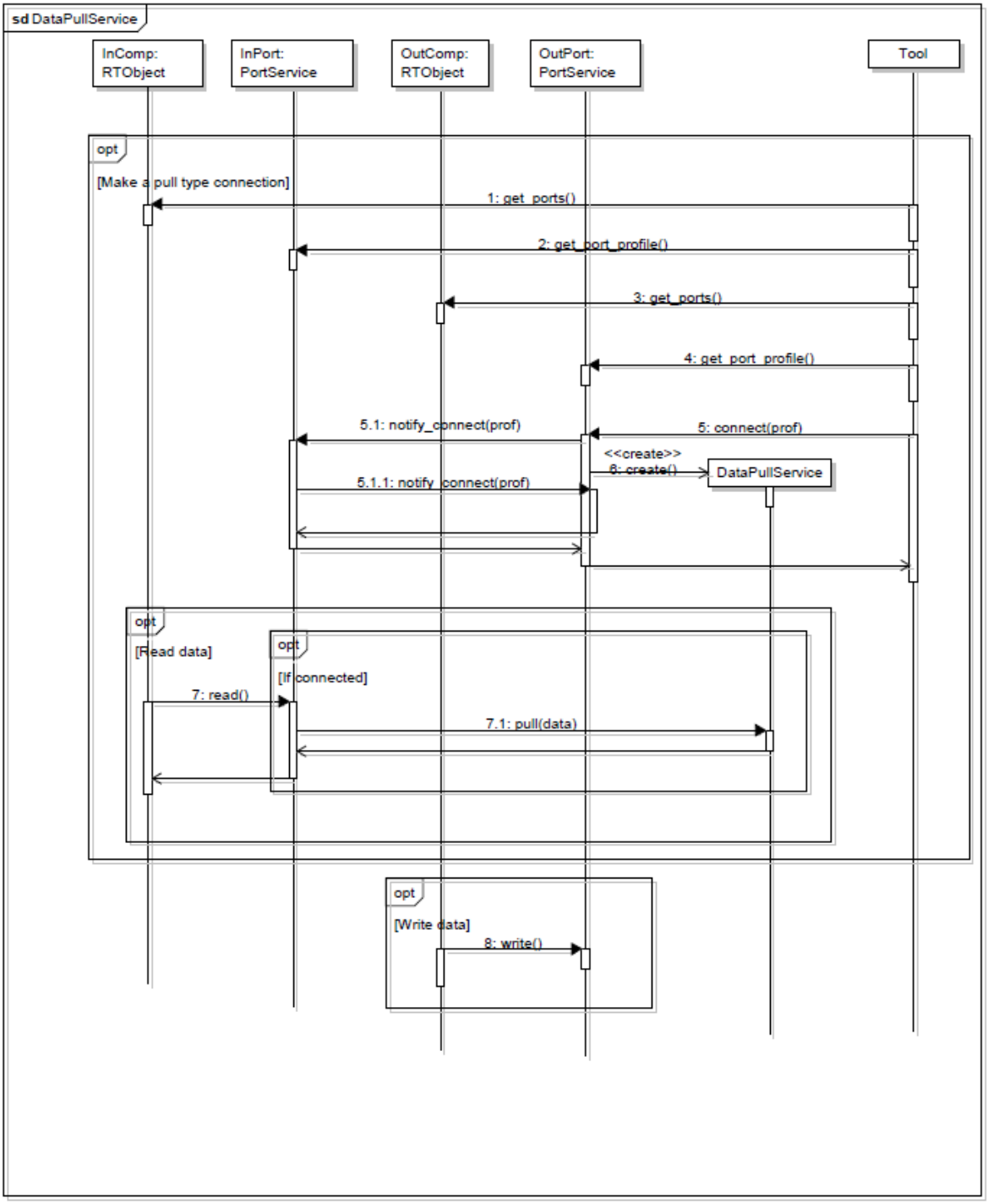

Figure 7.25 – Sequence for creating and using DataPullService (non-normative)

Operations

pull(in Octet[] data: PortStatus	This operation pulls an array of **Octet** from the target port with a specified binary format.

Attributes

No additional attributes.

Associations

No additional associations.

7.3 OMG IDL Platform Specific Model (PSM)

7.3.1 Overview

This sub clause introduces a CORBA specific model for the Finite State Machine Component for RTC (FSM4RTC) Platform Independent Model (PIM) defined in 7.2.

The FSM4RTC PIM defines the interfaces and necessary data structures. In the Platform Specific Model (PSM) these interfaces and the data structures used in the individual methods are mapped according to a CORBA IDL specification. The complete IDL specification is presented in Annex A.

An interface defined in the FSM4RTC PIM is mapped to a CORBA interface. An operation in a PIM interface is mapped to a CORBA operation. The other data types in the FSM4RTC PIM are mapped to the non-interface types in CORBA IDL. The CORBA IDL PSM is compliant with the IDL style guide [1].

In the CORBA IDL PSM, all interfaces as defined in the FSM4RTC PIM are directly mapped to CORBA interfaces. The IDL specification includes corresponding interface declarations. Additionally, all data structures used in the methods of these interfaces are also defined in the IDL specification.

The FSM4RTC IDL specification includes the following interface declarations:

Operations in the PIM that do not return a value of type **ReturnCode_t** shall report errors in the following ways, depending on their return type:

- Interface **ComponentObserver**
- interface **ExtendedFsmService**
- interface **DataPushService**
- interface **DataPullService**

Addition to the interfaces, data structures that are used as parameters in interface methods have to be defined in the PSM.

7.3.2 Basic Types

Basic types (see 7.2.3) shall map to the corresponding IDL types as follows.

7.3.2.1 String [UML]

String is mapped to **string**.

7.3.2.2 Octet [RTC]

Octet is mapped to **octet**.

7.3.2.3 ReturnCode_t [RTC]

ReturnCode_t is mapped to **RTC::ReturnCode_t** from RTC.idl [RTC]

7.3.2.4 NameValue [SDO]

NameValue is mapped to **SDOPackage::NameValue** from SDOPackage.idl [SDO].

NameValue[] is mapped to **SDOPackage::NVList** from SDOPackage.idl [SDO].

7.3.3 RTC module

The interfaces and data structures defined in the CORBA PSM belong to module RTC.

7.3.4 Data Types

This sub clause defines data structures that are used as parameters in FSM4RTC interface methods.

```
typedef SDOPackage::NVList NVList;
typedef sequence<octet> OctetSeq;
enum StatusKind {
  COMPONENT_PROFILE,
  RTC_STATUS,
  EC_STATUS,
  PORT_PROFILE,
  CONFIGURATION,
  RTC_HEARTBEAT,
  EC_HEARTBEAT,
```

```
    FSM_PROFILE,
    FSM_STATUS,
    FSM_STRUCTURE,
    USER_DEFINED,
    STATUS_KIND_NUM
};

struct FsmEventProfile {
    string name;
    string data_type;
};
typedef sequence<FsmEventProfile> FsmEventProfileList;

struct FsmStructure {
    string name;
    string structure;
    FsmEventProfileList event_profiles;
    NVList properties;
}

enum PortStatus {
    PORT_OK,
    PORT_ERROR,
    BUFFER_FULL,
    BUFFER_EMPTY,
    BUFFER_TIMEOUT,
    UNKNOWN_ERROR
}
```

7.3.5 ComponentObserver

7.3.5.1 ComponentObserver Interface

The ComponentObserver interface is mapped to a CORBA interface. The ComponentObserver interface supports an operation, update_status, which allows getting the list of organizations associated with the object implementing this interface.

```
interface ComponentObserver : SDOPackage::SDOService {
    oneway void update_status(in StatusKind status_kind, in string hint);
}
```

7.3.6 ExtendedFsmService

7.3.6.1 ExtendedFsmService Interface

```
interface ExtendedFsmService : SDOPackage::SDOService {
    string get_current_state();
    ReturnCode_t set_fsm_structure(in FsmStructure fsm_structure);
    ReturnCode_t get_fsm_structure(out FsmStructure fsm_structure);
}
```

7.3.7 Data Port

7.3.7.1 DataPushService Interface

```
interface DataPushService {
    PortStatus push(in OctetSeq data);
}
```

7.3.7.2 DataPullService Interface

```
interface DataPullService {
    PortStatus pull(out OctetSeq data);
}
```

This page intentionally left blank.

Annex A: OMG IDL

(normative)

A.1 ComponentObserver.idl

```
#ifndef _COMPONENT_OBSERVER_IDL_
#define _COMPONENT_OBSERVER_IDL_

#include <SDOPackage.idl>

#pragma prefix "omg.org"

module RTC
{
  enum StatusKind
  {
    COMPONENT_PROFILE,
    RTC_STATUS,
    EC_STATUS,
    PORT_PROFILE,
    CONFIGURATION,
    RTC_HEARTBEAT,
    EC_HEARTBEAT,
    FSM_PROFILE,
    FSM_STATUS,
    FSM_STRUCTURE,
    USER_DEFINED,
    STATUS_KIND_NUM
  };
  #pragma version StatusKind 1.0

  interface ComponentObserver : SDOPackage::SDOService
```

```
  {
    oneway void update_status(in StatusKind status_kind,
                              in string     hint);
  };
  #pragma version ComponentObserver 1.0
};

#endif // _COMPONENT_OBSERVER_IDL_
```

A.2 ExtendedFsmService.idl

```
#ifndef _EXTENDED_FSM_SERVICE_IDL_
#define _EXTENDED_FSM_SERVICE_IDL_

#include <RTC.idl>

#pragma prefix "omg.org"

module RTC
{
  struct FsmEventProfile
  {
    string name;
    string data_type;
  };
  #pragma version FsmEventProfile 1.0
  typedef sequence<FsmEventProfile> FsmEventProfileList;

  struct FsmStructure
  {
    string name;
    string structure;
    FsmEventProfileList event_profiles;
    NVList properties;
```

```
    };
    #pragma version FsmStructure 1.0

    interface ExtendedFsmService : SDOPackage::SDOService
    {
        string get_current_state();
        ReturnCode_t set_fsm_structure(in FsmStructure fsm_structure);
        ReturnCode_t get_fsm_structure(out FsmStructure fsm_structure);
    };
    #pragma version ExtendedFsmService 1.0
};

#endif // _EXTENDED_FSM_SERVICE_IDL_
```

A.3 DataPort.idl

```
#ifndef _DATA_PORT_IDL_
#define _DATA_PORT_IDL_

#pragma prefix "omg.org"

module RTC
{
    enum PortStatus
    {
        PORT_OK,
        PORT_ERROR,
        BUFFER_FULL,
        BUFFER_EMPTY,
        BUFFER_TIMEOUT,
        UNKNOWN_ERROR
    };
    #pragma version PortStatus 1.0

    typedef sequence<octet> OctetSeq;
```

```
    interface DataPushService
    {
        PortStatus push(in OctetSeq data);
    };
    #pragma version DataPushService 1.0

    interface DataPullService
    {
        PortStatus pull(out OctetSeq data);
    };
    #pragma version DataPullService 1.0
};

#endif // _DATA_PORT_IDL_
```

Annex B: References

(non-normative)

[1] OMG IDL Style Guide, ab/98-06-03
[2] XML Metadata Interchange, http://www.omg.org/spec/XMI
[3] SCXML State Chart XML, http://www.w3.org/TR/scxml/

This page intentionally left blank.