Platform Independent Model (PIM) and Platform Specific Model (PSM) for Super Distributed Objects (SDO) Specification

This OMG document replaces the submission document (sdo/03-03-01) and the Draft Adopted specification (dtc/03-04-02). It is an OMG Final Adopted Specification and is currently in the finalization phase. Comments on the content of this document are welcomed, and should be directed to *issues@omg.org* by December 15, 2003.

You may view the pending issues for this specification from the OMG revision issues web page *http://www.omg.org/issues/*; however, at the time of this writing there were no pending issues.

The FTF Recommendation and Report for this specification will be published on April 30, 2004. If you are reading this after that date, please download the available specification from the OMG Specifications Catalog.

OMG Adopted Specification
dtc/03-09-01

Platform Independent Model (PIM) and Platform Specific Model (PSM) for Super Distributed Objects (SDO) Specification

September 2003
Final Adopted Specification
dtc/03-09-01

An Adopted Specification of the Object Management Group, Inc.

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, 250 First Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

The OMG Object Management Group Logo®, CORBA®, CORBA Academy®, The Information Brokerage®, XMI® and IIOP® are registered trademarks of the Object Management Group. OMG™, Object Management Group™, CORBA logos™, OMG Interface Definition Language (IDL)™, The Architecture of Choice for a Changing World™, CORBAservices™, CORBAfacilities™, CORBAmed™, CORBAnet™, Integrate 2002™, Middleware That's Everywhere™, UML™, Unified Modeling Language™, The UML Cube logo™, MOF™, CWM™, The CWM Logo™, Model Driven Architecture™, Model Driven Architecture Logos™, MDA™, OMG Model Driven Architecture™, OMG MDA™ and the XMI Logo™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

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

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

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 & Specifications, Report a Bug/Issue.

Contents

Preface

About This Document

Under the terms of the collaboration between OMG and The Open Group, this document is a candidate for adoption by The Open Group, as an Open Group Technical Standard. The collaboration between OMG and The Open Group ensures joint review and cohesive support for emerging object-based specifications.

Object Management Group

The Object Management Group, Inc. (OMG) is an international organization supported by over 600 members, including information system vendors, software developers and users. Founded in 1989, the OMG promotes the theory and practice of object-oriented technology in software development. The organization's charter includes the establishment of industry guidelines and object management specifications to provide a common framework for application development. Primary goals are the reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments. Conformance to these specifications will make it possible to develop a heterogeneous applications environment across all major hardware platforms and operating systems.

OMG's objectives are to foster the growth of object technology and influence its direction by establishing the Object Management Architecture (OMA). The OMA provides the conceptual infrastructure upon which all OMG specifications are based. More information is available at http://www.omg.org/.

The Open Group

The Open Group, a vendor and technology-neutral consortium, is committed to delivering greater business efficiency by bringing together buyers and suppliers of information technology to lower the time, cost, and risks associated with integrating new technology across the enterprise.

The mission of The Open Group is to drive the creation of boundaryless information flow achieved by:

- Working with customers to capture, understand and address current and emerging requirements, establish policies, and share best practices;
- Working with suppliers, consortia and standards bodies to develop consensus and facilitate interoperability, to evolve and integrate specifications and open source technologies;
- Offering a comprehensive set of services to enhance the operational efficiency of consortia; and
- Developing and operating the industry's premier certification service and encouraging procurement of certified products.

The Open Group has over 15 years experience in developing and operating certification programs and has extensive experience developing and facilitating industry adoption of test suites used to validate conformance to an open standard or specification. The Open Group portfolio of test suites includes tests for CORBA, the Single UNIX Specification, CDE, Motif, Linux, LDAP, POSIX.1, POSIX.2, POSIX Realtime, Sockets, UNIX, XPG4, XNFS, XTI, and X11. The Open Group test tools are essential for proper development and maintenance of standards-based products, ensuring conformance of products to industry-standard APIs, applications portability, and interoperability. In-depth testing identifies defects at the earliest possible point in the development cycle, saving costs in development and quality assurance.

More information is available at http://www.opengroup.org/ .

OMG Documents

The OMG Specifications Catalog is available from the OMG website at:

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

The OMG documentation is organized as follows:

OMG Modeling Specifications

Includes the UML, MOF, XMI, and CWM specifications.

OMG Middleware Specifications

Includes CORBA/IIOP, IDL/Language Mappings, Specialized CORBA specifications, and CORBA Component Model (CCM).

Platform Specific Model and Interface Specifications

Includes CORBAservices, CORBAfacilities, OMG Domain specifications, OMG Embedded Intelligence specifications, and OMG Security specifications.

Obtaining OMG Documents

The OMG collects information for each book in the documentation set by issuing Requests for Information, Requests for Proposals, and Requests for Comment and, with its membership, evaluating the responses. Specifications are adopted as standards only when representatives of the OMG membership accept them as such by vote. (The policies and procedures of the OMG are described in detail in the *Object Management Architecture Guide*.) OMG formal documents are available from our web site in PostScript and PDF format. Contact the Object Management Group, Inc. at:

OMG Headquarters

250 First Avenue

Needham, MA 02494

USA

Tel: +1-781-444-0404

Fax: +1-781-444-0320

pubs@omg.org

http://www.omg.org

Typographical Conventions

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

Helvetica bold - OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier bold` - Programming language elements.

Helvetica - Exceptions

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

Acknowledgments

The following companies submitted and/or supported parts of this specification:

- Fraunhofer Fokus
- Hitachi Ltd.
- University of California Irvine

Overview 1

Contents

This chapter contains the following sections.

1.1 Overview

This chapter provides an overview of the scope and objectives of this specification, and describes the resolution to RFP requirements and compliance points.

1.2 Scope

The increasing availability of high-performance and low-cost processor technology is enabling computing power to be embedded densely in various devices (e.g., mobile phones, PDAs and Internet appliances) as well as traditional computers. Furthermore, emerging networking technologies such as wireless LAN, IPv6, and plug-and-play-enabled platforms allow those devices to connect to each other in an easy and ad-hoc manner and to construct a large scale network of devices that provides various applications. Much attention is being paid on ubiquitous or pervasive computing driven by these technological advances. A goal of these networking infrastructures is to provide a distributed community of devices and software components that pool their services for solving problems, composing applications and sharing information. The scope of this

specification is the transition and abstraction of those infrastructure technologies that target resource interconnection on highly distributed environments into a higher layer with OMG technologies (e.g., UML and CORBA).

1.3 Objectives

An SDO is a logical representation of a hardware device or a software component that provides well-known functionality and services. One of the key characteristics in super distribution is to incorporate a massive number of objects, each of which performs its own task autonomously or cooperatively with other objects. Examples of SDOs include abstractions of devices such as mobile phones, PDAs, and home appliances, but are not limited to device abstractions. An SDO may abstract software component and act as a peer in a peer-to-peer networking system. SDOs provide various different functionalities (e.g., TV set, refrigerator and light switch) and abstract underlying heterogeneous technologies. They are organized in an ad hoc manner to provide an application service in mobile environments[1]. For other characteristics in super distribution, please refer the Super Distributed Objects Whitepaper [1].

Today, there are several resource interconnection technologies such as Universal Plug and Play, HAVi, OSGi, ECHONET and Jini. They are, however, restricted to specific platforms, network protocols and programming languages, or they focus on limited application domains. No common model-based standards exist to handle various resources in a unified manner independently of underlying technologies and application domains. The objectives of this specification are to abstract the existing resource interconnection technologies into a higher layer, define their information and computational models in the layer, and make objects defined the models interoperable.

1.4 Compliance

This specification consists of two parts; a Platform Independent Model (PIM) described with UML and a Platform Specific Model (PSM) that specifies a realization of the PIM with CORBA IDL. Every compliant implementation must follow the PIM design of the SDO interface (see Section 2.2.3, “SDOSystemElement,” on page 2-6 for more details). There are two compliance points in implementations; (1) the SDO interface must be implemented based on its CORBA PSM (CORBA IDL), or (2) it must be mapped to another (non-CORBA) target technology without breaking any semantics defined in the PIM and implemented on the target technology.

No partial implementation of optional resource data model or interfaces without mandatory ones is deemed conformant.

Platform Independent Model (PIM) 2

Contents

This chapter contains the following sections.

This section specifies the PIM from information and computational viewpoints of SDOs. Section 2.2, “Resource Data Model,” on page 2-2 describes the PIM for the resource data model, which is used to describe the capabilities and properties of SDOs (i.e., an information aspect of SDOs). Section 2.3, “Interfaces,” on page 2-15 defines the interfaces to access and manipulate resource data (i.e., a computational aspect of SDOs). The PIM is specified using the UML specification in version 1.4.

2.1 Overview of PIM for SDO

An SDO represents a hardware device that may be accessed through existing standards or software components and provides information and interfaces for dynamic access by other applications. As described above, the PIM consists of the resource data model and the interfaces to access and manipulate resource data. The PIM for SDO in this specification is described based on the following policy.

- Attributes to describe several core data are defined, and named values for extensible representation of various attributes.

The resource data model is built as a series of UML classes that represent key information aspects of SDOs. Each class in the resource data model defines a set of attributes that represent static and dynamic properties of SDOs. The attributes are defined as typed variables or named values. The typed variables are used to specify the common attributes that all the implementations share. The named values are used to specify the attributes specific to implementations (applications).

- Basic interfaces are defined as mandatory so that other optional parts can be navigated.

  The interfaces are defined as a set of UML classes that represent key computational aspects of SDOs. Each class defines an interface that contains a set of operations to access and manipulate the SDO resource data.

2.2 Resource Data Model

This section specifies the SDO resource data model, which is used to describe the capabilities and properties of SDOs.

2.2.1 Overview of Resource Data Model

The resource data model includes the following constructs:

- Profiles
  - Device profile, which defines a set of hardware specific properties, (see Section 2.2.7, “DeviceProfile,” on page 2-10.
  - Service profile, which defines a set of software specific properties, (see Section 2.2.8, “ServiceProfile,” on page 2-11).
  - Configuration profile, which defines a set of properties to configure SDOs, (see Section 2.2.10, “ConfigurationProfile,” on page 2-12).
- Organization, which defines a relationship between/among the objects running in SDO system (see Section 2.2.5, “Organization,” on page 2-8).
- Status, which indicates the current status of SDOs (see Section 2.2.9, “Status,” on page 2-12).

Figure 2-1 shows a simplified UML notation of the resource data. The complete diagram is depicted in Appendix B.1.

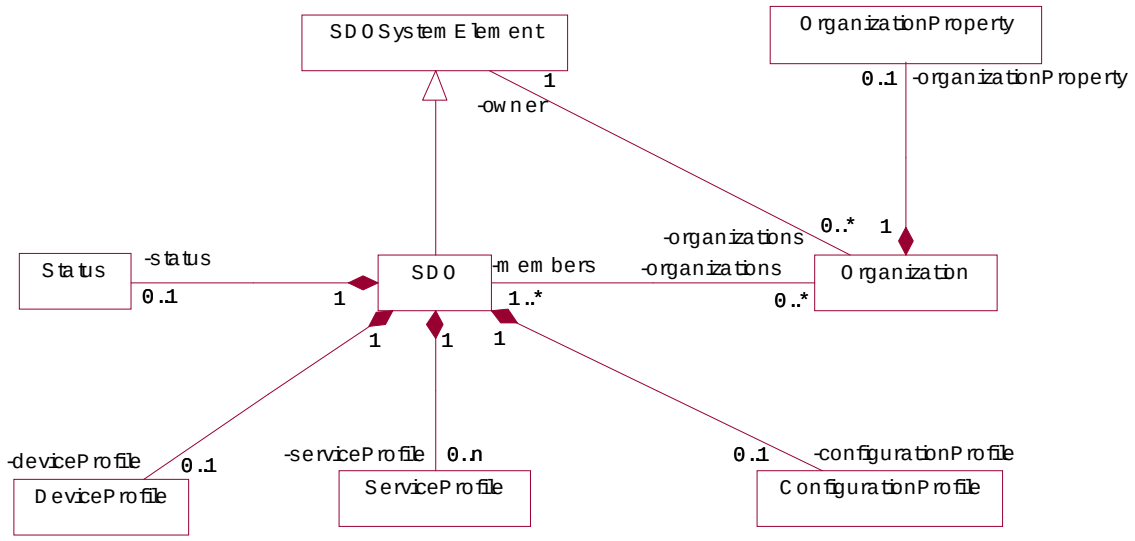
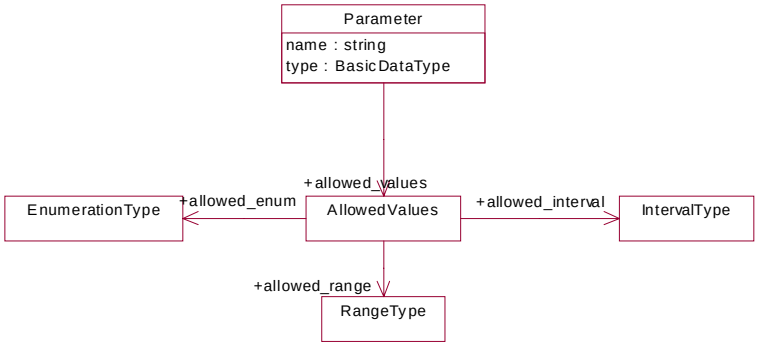
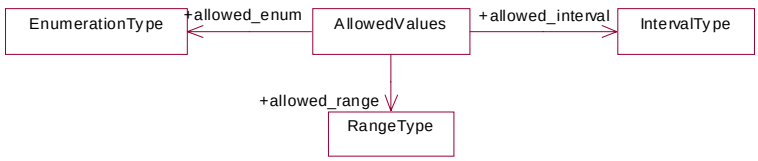


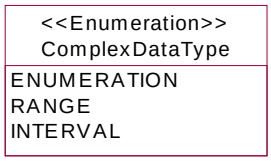
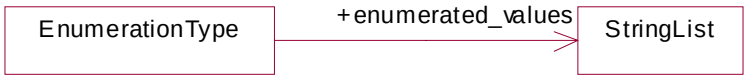
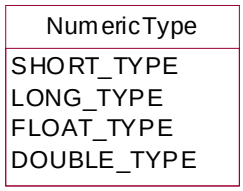
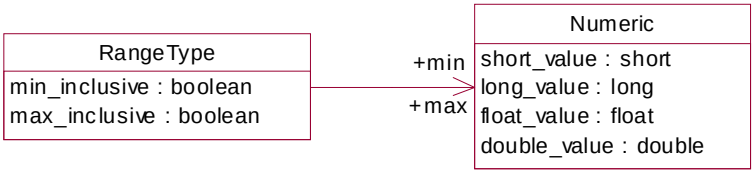
Figure 2-1 SDO resource data model

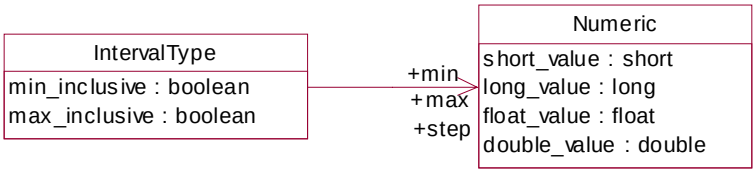
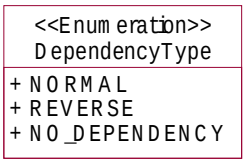
2.2.2 Data Structures Used by Resource Data Model

This section defines the data structures used by the resource data model.

Name	Description
StringList	A list of strings.
UniqueIdentifier	Identifier for the constructs in the resource data model (e.g., SDO). Each identifier is typed as a string and must be unique in a given domain of application deployment.[1]
NameValue	A pair of a name and its value. NameValue + name : string + value : any
NVList	A list of **NameValue** pairs.

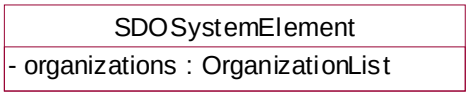
Parameter	Data structure to define a variable (parameter) independently of implementation technologies. The **Parameter** structure defines the name and type of a variable.  Attributes defined in **Parameter**. • **name** – parameter's name • **type** – name of parameter's type. The original value scope of parameter data type can be constrained by definitions allocated in the attribute **allowedValues**. • **allowedValues** - Values that the parameter can accept. This attribute is used only when the value scope inherent to the parameter type must be constrained. For example, the values allowed for a string parameter may be constrained by an enumeration, or the values allowed for a numeric parameter may be constrained by a range. The values allowed for a parameter can be defined in enumeration, range, or interval structures. The value of attribute **allowedValues** is null if there is no constraint on a parameter value, that is, any value can be assigned to the parameter as far as it follows the value scope inherent to the parameter's type.
BasicDataType	Data structure representing a basic data type that can be used to declare a type of a **Parameter**'s type (see description of **Parameter** above). Typically, basic data types include integer, floating point numbers and string. Available basic data types depend on the underlying implementation technology used (e.g., programming language or middleware technology). In the CORBA PSM in this specification, **BasicDataType** is **typedef**-ed with **CORBA::TCKind** (See Section 3.2, "Data types used in CORBA PSM," on page 3-2 and Chapter 4).
AllowedValues	 Data structure used to apply a constraint on the value scope inherent to a given parameter type. The constraint is an enumeration, range, or interval, which is defined with **EnumerationType**, **RangeType**, or **IntervalType** structures, respectively (see below).

ComplexDataType	 **ComplexDataType** enumerates three types of structures, each of which can be used in the attribute **allowedValues** to define a value scope of constraint. The structures that can be used to specify enumeration, range or interval are defined below.
EnumerationType	 The enumeration of values that can be assigned to the attribute AllowedValues of **Parameter**. The enumerated values are always of type string . • **enumerated_values** – a list of string values that a **Parameter** can contain. Example: {“red”, “blue”, “white”}
NumericType	 This enumeration is used by structures **RangeType** and **IntervalType**. It defines numeric types used to specify the bounds of intervals or ranges, and the step between interval values.
RangeType	 Data structure representing a range of values that can be assigned to the attribute AllowedValues of **Parameter**. • **min** – the lower bound of the range • **max** – the upper bound of the range • **minInclusive** – a boolean value showing if the lower bound value is included in the range • **maxInclusive** – a boolean value showing if the upper bound value is included in the range. Example: the range ((int) 0, (int) 20, false, true) defines values {1,2,…20}.

IntervalType	 Data structure representing an interval of values that can be assigned to the attribute AllowedValues of **Parameter** defined as interval. • **min** – the lower bound of the interval • **max** – the upper bound of the interval • **minInclusive** – a boolean value showing if the lower bound value is included in the interval • **maxInclusive** – a boolean value showing if the upper bound value is included in the interval • **step** – the step between the values in the interval. Example: the interval ((int) 0, (int) 20, false, true, 5) defines values {5,10,15,20}.
ParameterList	A list of **Parameter** structures.
DependencyType	Data type used to specify relation between elements in an organization. The value indicates if one side of an **Organization** depends on the other side. If the **Organization** represents dependency relationship, it also indicates which side depends on which side. Enumeration **DependencyType** includes three possible forms of dependency. 

1. It is beyond the scope of this specification to define the format of identifiers and the algorithm to generate them, because they are implementation dependent. For example, some applications may use standardized schemes such as the UUID [2], others may use proprietary ones. Different SDO systems need to follow an agreed scheme for identifiers to maintain the interoperability between SDOs.

2.2.3 SDOSystemElement

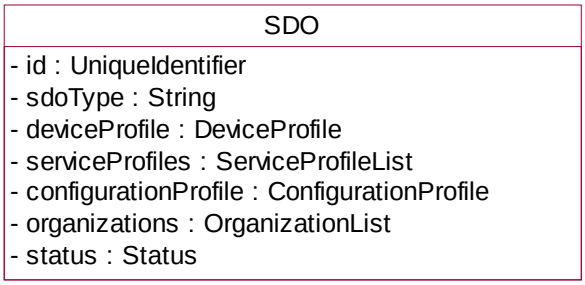


SDOSystemElement is the base class of the classes that represent SDO system's elements. It is used to indicate that its subclasses represent any system elements running on the SDO environments. A representative example of the SDO's system elements is SDOs. SDO is defined as a subclass of **SDOSystemElement** in Section 2.2.4. The system elements that are running on the SDO environments but are not SDOs are hereinafter collectively called non-SDOs in this specification. Examples of non-SDOs include human users and locations. It is left to future specifications to define non-SDOs as additional subclasses of **SDOSystemElement**.

*<**Attributes**>*

Attribute	Type	Description
organizations	**OrganizationList**	A list of **Organization** objects that **SDOSystemElement** has.

2.2.4 SDO

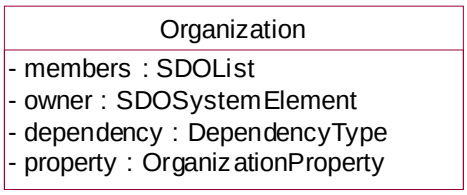


The class **SDO** defines a set of common properties for hardware device and software component representations.

<Attributes>

Attribute	Type	Description
id	**UniqueIdentifier**	Unique identifier for an SDO.
sdoType	**String**	Textual description of the SDO. **sdoType** contains short description of the SDO's functionality.
deviceProfile	**DeviceProfile**	A device profile that an SDO has. Null is assigned when an SDO does not have any device profile. See Section 2.2.7, "DeviceProfile," on page 2-10 for more details about device profile.
serviceProfiles	**ServiceProfileList**	A list of service profiles that an SDO has. Null is assigned when an SDO does not have any service profile. See Section 2.2.8, "ServiceProfile," on page 2-11 for more details about service profile.
configurationProfile	**ConfigurationProfile**	A configuration profile that an SDO has. Null is assigned when an SDO does not have its configuration profile. See Section 2.2.10, "ConfigurationProfile," on page 2-12 for more details about configuration profile.
status	**Status**	A status information of an SDO. Null is assigned when no status information is available. See Section 2.2.9, "Status," on page 2-12 for more details about status.
organizations	**OrganizationList**	A list of references to the list of **Organization** objects that an SDO keeps with other SDOs or objects of type **SDOSystemElement**. Null is assigned when an SDO has no relationship with other SDOs or **SDOSystemElement** objects. See Section 2.2.5, "Organization," on page 2-8 for more details about class **Organization**.

2.2.5 Organization



Organization represents a relationship between/among **SDOSystemElements**. An organization can be established among different SDOs, or between SDOs and a non-SDO. It can also be used to represent a 1-to-1 relationship. The properties of an Organization can be stored in **OrganizationProperty** (see Section 2.2.6, “OrganizationProperty,” on page 2-10).

*<**Attributes**>*

Attribute	Type	Description
members	**SDOList**	A list of SDOs that are the members associated with the **Organization**.
owner	**SDOSytemElement**	The owner of the **Organization**. It can be an SDO or non-SDO (See the text under this table for more detail).
dependency	**DependencyType**	This attribute specifies the dependency relation between the owner and members of the organization. Further details are discussed in text under this table.
property	**OrganizationProperty**	Property of the **Organization**. **OrganizationProperty** is described in Section 2.2.6, “OrganizationProperty,” on page 2-10.

Organization is used to form the following three patterns of topology.

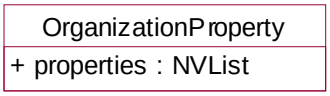
1. Hierarchical organization, which indicates **owner** supervises **members**. In this case, **DependencyType** should hold **NORMAL** value (see description of DependencyType in Page 2-6).
2. Reversely hierarchical organization, which indicates `members` supervise `owner`. In this case, **DependencyType** should hold **REVERSE** value (see description of DependencyType in Page 2-6).
3. Flat organization, which indicates no dependency exists. In this case, **DependencyType** should hold **NO_DEPENDENCY** value (see description of DependencyType in Page 2-6).

Both an SDO and a non-SDO (i.e., an instance of the subclasses specializing **SDOSystemElement**) can act as owner of an **Organization**. When an SDO is an **owner**, Organization can represent any of the above three topology patterns.

- When an Organization represents topology pattern (1), an SDO (**owner**) controls one or more SDOs (**members**). For example, air conditioner (owner) controls a temperature sensor (member), humidity sensor (member), and wind flow controller (member).
- When an Organization represents topology pattern (2), multiple SDOs(members) share an SDO (owner). For example, an amplifier (owner) is shared by several AV components (members) in an AV stereo.
- When a non-SDO is an **owner**, examples of the topology are as follows.
  - User (owner)-SDO (members): When a user (owner) supervises one or more SDOs (members), the **organization** represents topology pattern (1).

- Location (owner)-SDO (members): When one or more SDOs (members) are operating in a specific location (owner), the **organization** represents topology pattern (3). For example, multiple PDAs in a same place (e.g., a room) have equal relationships among them to communicate with each other.

2.2.6 OrganizationProperty

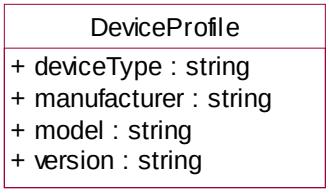


OrganizationProperty contains the properties of an **Organization**. An Organization has zero or one (at most one) instance of OrganizationProperty.

<Attributes>

Attribute	Type	Description
properties	**NVList**	A set of properties of an **Organization**. The property values contained in this attribute are implementation dependent.Examples include the identifier of an **Organization** and the time when an **Organization** is established.

2.2.7 DeviceProfile

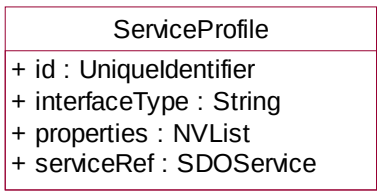


DeviceProfile defines the properties of a device that an SDO represents.

<Attributes>

Attribute	Type	Description
deviceType	**String**	General type name of a device. This attribute describes general kind of devices to categorize them and specify fundamental capability of the device.
manufacturer	**String**	Identifier for the manufacturer of the device.
model	**String**	Model name of the device.
version	**String**	Version number of the device.
properties	**NVList**	Device specific properties in addition to the above four ones. The property values contained in this attribute are implementation dependent.

2.2.8 ServiceProfile



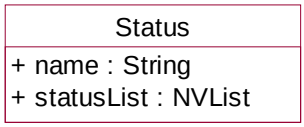
ServiceProfile defines a set of properties of the function provided by a device or software component that an SDO represents. The function is implemented by another object that **serviceRef** refers to (see also Section 2.3.5, “SDOService Interface,” on page 2-29).

For example, an air conditioner SDO has a function (a capability of service) for "fixing room temperature," and has control interfaces (referred by **ServiceRef**) for operations like "set a room's temperature," "set operation mode (heating/cooling/dehumidifying)," "turn on/off power," and so on.

<Attributes>

Attribute	Type	Description
id	**UniqueIdentifier**	Identifier for a service (or function) that an SDO provides. An SDO provides one or more services (functions), and '**id**' is used to distinguish different services (functions).
interfaceType	**String**	The type of the interface through which an SDO provides its service (function). The scheme to describe the interface type depends on underlying implementation technologies. In the proposed CORBA PSM, this attribute contains the repository ID of the IDL interface through which an SDO's service (function) is provided.
properties	**NVList**	List of properties specific to the service (function) that an SDO provides. The property values contained in this attribute are implementation dependent.
serviceRef	**SDOService**	Reference to the object that provides the service (function) represented by this profile.

2.2.9 Status

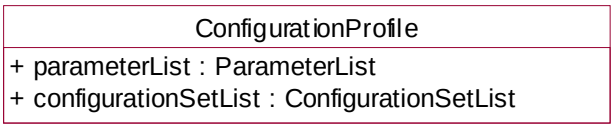


Status contains the current status of an SDO. It contains the name defining the kind of status, and a set of status described by a pair of name and value for each status values. The current availability (name) of an SDO with concrete status data (e.g., list of power on/off, activated/deactivated) is an example of status information.

<Attributes>

Attribute	Type	Description
name	**String**	Name of status.
statusList	**NVList**	A list containing status information.

2.2.10 ConfigurationProfile

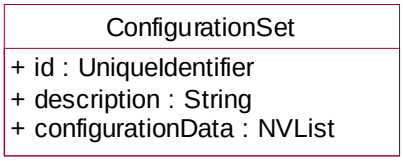


ConfigurationProfile contains a set of properties to configure an SDO.

*<**Attributes**>*

Attribute	Type	Description
parameters	**ParameterList**	A list of **Parameter** that represents the kinds of properties to configure an SDO.
configurationSets	**ConfigurationSetList**	A list of **configurationSet** (described below) objects that represents a set of properties with their values to configure an SDO.

- Data type definition: **ConfigurationSet**



Parameter defines the data type of a variable.

Attribute	Type	Description
id	**UniqueIdentifier**	Identifier of the set of configuration data stored in **ConfigurationProfile**. This can be used to activate the stored configuration.
description	**String**	Descriptive information for configuration data.
configurationData	**NVList**	A set of configuration data. This is used to configure an SDO.

2.2.11 Examples of resource data model

This section shows several examples of SDO resource data defined in this specification in order to demonstrate how it can be used. Two different types of SDOs (Thermometer SDO and Airconditioner SDO) are described as example SDOs.

The other SDO, for example TemperatureController, gets the temperature of the room from the Thermometer SDO and controls the Airconditioner SDO.

1. Thermometer SDO

The thermometer SDO illustrated below is a logical representation of a thermometer (device). The SDO keeps two attribute values that indicate its identifier and descriptive information, and has a **DeviceProfile** and **Status**.

DeviceProfile		
attribute	value	
deviceType	*Temperature Sensor*	
manufacturer	*Thermo Inc.*	
model	*TH310*	
version	*1*	
properties	name	value
	unit	*Celsius*
	rangeMin	*-50*
	rangeMax	*50*

SDO	
attribute	value
id	*abc_12345*
sdoType	*EN_TemperatureSensor*

Status		
attribute	value	
name	*deviceStatus*	
statusList	name	value
	thermoValue	*30*

Figure 2-2 Example: Thermometer SDO

The **DeviceProfile** contains three attributes in its **properties** attribute; `unit`, **rangeMin** and **rangeMax**. They specify the unit of the temperature, minimum and maximum degrees of temperature that the thermometer can sense, respectively.

Status contains a status information of the Thermometer SDO. The attribute *thermoValue* holds the current temperature (30 degrees) of the room where it is located. The value of this attribute can be monitored by other SDOs but it cannot be changed or configured.

Because thermometer SDO does not provide any software service (function), it does not have a **ServiceProfile**.

2. Airconditioner SDO

The airconditioner SDO described below is a logical representation of an air conditioner (device). The SDO keeps two attribute values that indicate its identifier and descriptive information, and has a **DeviceProfile**, two **Organization** objects, and two **ServiceProfile** objects.

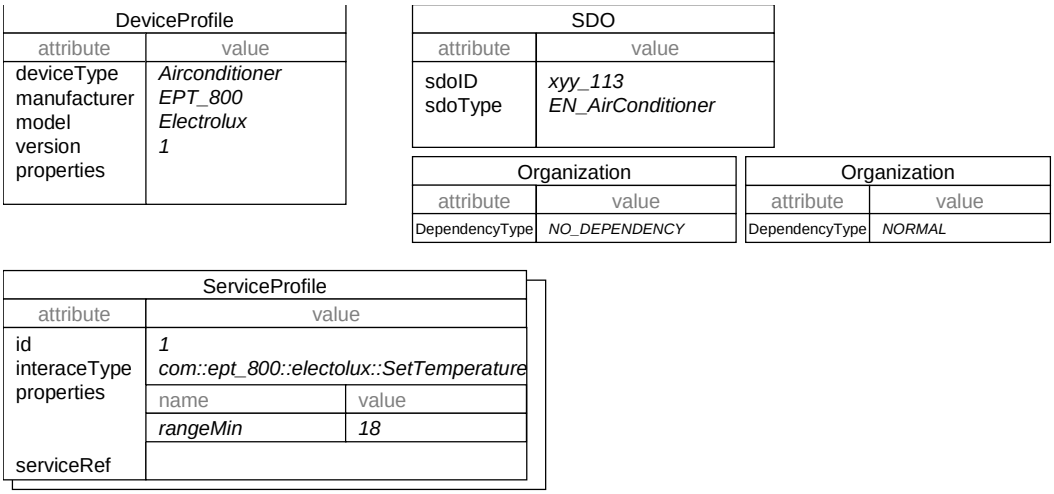


Figure 2-3 Example: Airconditioner SDO

Since the Airconditioner SDO provides two different software services (functions), it provide two **ServiceProfile** objects. One of them is the function to set a target degree of temperature. Its interface type is defined as "com::ept_800::electolux::SetTemperature". (In this example, **interfaceType** is described by repository ID in CORBA. Concrete values of **interfaceType** is dependent on implementing technologies.)

This airconditioner SDO has two **Organization** objects that connects with the other SDOs (e.g., surrounding devices, peripheral device SDOs). If the thermometer SDO and airconditioner SDO are located in the same room, they can be related with an **Organization** object that specifies "**NO_DEPENDENCY**" in its **DependencyType** attribute and a reference to a non-SDO representing the room in its **owner** attribute.

If the thermometer SDO is contained in the airconditioner, they can be related with an **Organization** object that specify "**NORMAL**" in its **DependencyType** attribute and references to the thermometer SDO (member) and airconditioner SDO (owner).

2.3 Interfaces

Two constructs defined in the resource data model runs as objects in SDO systems; **SDO** and **Organization**. This section describes their common interfaces.

2.3.1 Overview of Interfaces

This specification defines the following five interfaces that **SDO** and **Organization** implement.

- **SDOInterface**, which defines a series of operations to obtain SDO's properties and the other interfaces that the SDO implements (**MonitoringInterface**, **ConfigurationInterface**, and **SDOService**). All the SDOs must implement this interface.
- **MonitoringInterface**, which defines the operations to monitor changes in SDO's properties. Every SDO does not implement this interface. **NotificationCallbackInterface** is also defined as a call back interface of notification subscribed by using **MonitoringInterface**. This interface can be implemented by SDOs optionally.
- **ConfigurationInterface**, which defines the operations to configure SDO's profiles (e.g., device, service, and configuration profiles) and Organizations associated with the SDO. Every SDO does not implement this interface.
- **SDOService**, which abstracts the service provided by an SDO. Every SDO does not implement this interface.
- **OrganizationInterface**, which defines the operations to establish and maintain **Organizations**. All the **Organizations** must implement this interface.

2.3.2 Data Structures used by Interfaces

2.3.2.1 SDOException

SDOException encapsulates the exceptions that can be raised in SDO systems.

<Attributes>

Attribute	Type	Description
type	**String**	Name of a raised exception.
description	**String**	Descriptive information of a raised exception.

List of Exception types

SDONotExists

This exception is raised when a client of an SDO cannot reach the target SDO.

NotAvailable

This exception is thrown when there is no response from a target SDO. For example, it may be raised when a target SDO has already been stopped or down in an unusual state.

InvalidParameter

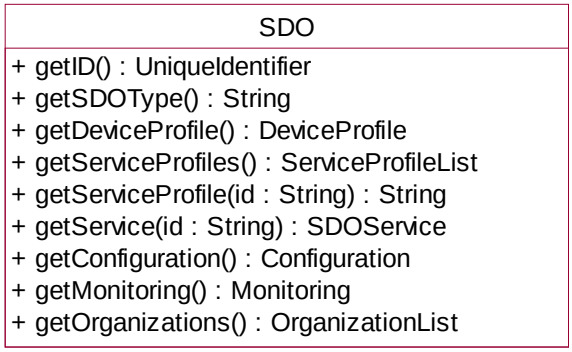
This exception is raised when a parameter(s) specified in an operation call is invalid. For example, it may be raised when null is assigned in a parameter that should contain a value.

InterfaceNotImplemented

This exception is raised when an interface that a client tries to access is not implemented. For example, it may be raised when a client tries to obtain a reference to **MonitoringInterface** through **getMonitoring()**, but it is not implemented.

2.3.3 SDO Interface

The **SDO** interface is used to manage elements of the SDO. All the other interfaces specified in this specification are navigated from **SDO** interface.



(1) *getID ( ) : String*

This operation returns **id** of the SDO .

Parameter	Type	Description
<return>	**UniqueIdentifier**	**id** of the SDO defined in the resource data model.

Exceptions

This operation throws **SDOException** with the following type:

- InvalidParameter type SDOException if the parameter **id** is null.
- SDONotExists type SDOException if the target SDO does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(2) *- + getSDOType ( ) : String*

This operation returns **sdoType** of the SDO.

Parameter	Type	Description
<return>	**String**	**sdoType** of the SDO defined in the resource data model.

Exceptions

This operation throws **SDOException** with the following type:

- InvalidParameter type SDOException if the parameter **sdoType** is null.
- SDONotExists type SDOException if the target SDO does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(3) + *getDeviceProfile ( ) : DeviceProfile*

This operation returns the **DeviceProfile** of the SDO

Parameter	Type	Description
<return>	**DeviceProfile**	Returns the **DeviceProfile** of the SDO if it exists, or NULL if it does not exist.

Exceptions

This operation throws **SDOException** with the following type:

- SDONotExists type SDOException if the target SDO does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(4) + *getServiceProfiles ( ) : ServiceProfileList*

This operation returns a list of **ServiceProfile**s which the SDO has.

Parameter	Type	Description
<return>	**ServiceProfilesList**	List of **ServiceProfile**s of all the services the SDOs' function providing.

Exceptions

This operation throws **SDOException** with the following type:

- SDONotExists type SDOException if the target SDO does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(5) + *getServiceProfile (id : String) : ServiceProfile*

This operation returns the **ServiceProfile** which is specified by the argument „**id**“.

Parameter	Type	Description
id	**String**	The identifier referring to one of the **ServiceProfile**s.
<return>	**ServiceProfile**	The profile of the specified service.

Exceptions

This operation throws **SDOException** with the following type.

- InvalidParameter type SDOException if the **ServiceProfile** which is specified by argument id does not exist.
- SDONotExists type SDOException if the target SDO does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(6) + *getServiceRef (id : String) : SDOService*

This operation returns a reference to a function which is specified by the argument **id**. Clients can invoke each function by using the returned reference.

Parameter	Type	Description
id	**String**	The identifier of the **ServiceProfile** referring to the requested service of an SDO.
<return>	**SDOService**	The service reference of the specified service.

Exceptions

This operation throws SDOException with the following type.

- InvalidParameter type SDOException if argument "**id**" is null, or if the **ServiceProfile** which is specified by argument "**id**" does not exist.
- SDONotExists type SDOException if the target SDO does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(7) + getConfiguration () : Configuration

This operation returns an object implementing the **Configuration** interface. The **Configuration** interface is one of the interfaces that each SDO maintains. The interface is used to configure the attributes defined in **DeviceProfile**, **ServiceProfile** and **Organization**. See Section 2.3.4, “Configuration Interface,” on page 2-22 for more details about the **Configuration** interface.

Parameter	Type	Description
<return>	**Configuration**	The **Configuration** interface of an SDO.

Exceptions

This operation throws **SDOException** with the following type.

- InterfaceNotImplemented type SDOException if the target SDO has no Configuration interface.
- SDONotExists type SDOException if the target SDO does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(8) + getMonitoring ( ) : Monitoring

This operation returns an object implementing the **Monitoring** interface. The **Monitoring** interface is one of the interfaces that each SDO maintains. The interface is used to monitor the properties of an SDO. See Section 2.3.6, “Monitoring Interface,” on page 2-29 for more details about the Monitoring interface.

Parameter	Type	Description
<return>	**Monitoring**	The **Monitoring** interface of an SDO.

Exceptions

This operation throws **SDOException** with the following type.

- InterfaceNotImplemented type SDOException if the target SDO has no **Monitoring** interface.
- SDONotExists type SDOException if the target SDO does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(9) + getOrganizations ( ) : OrganizationList

An SDO belongs to zero or more organizations. If the SDO belongs to one or more organizations, this operation returns the list of organizations that the SDO belongs to.

Parameter	Type	Description
<return>	**OrganizationList**	Returns the list of **Organizations** that the SDO belong to.

Exceptions

This operation throws **SDOException** with the following type.

- SDONotExists type SDOException if the target SDO does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

2.3.3.1 *Usage: SDO interface*

The section describes examples about how to use the operations of the SDO interface to get the SDO identifier.

As an example, the operation to get SDO parameter **id** is shown in Figure 2-4.

In Figure 2-4, sdo1 makes requests to sdo2 to acquire the parameter. The example of invocation of the operations that enable to get other interfaces by using SDO interface is described in Section 2.3.6, “Monitoring Interface,” on page 2-29.

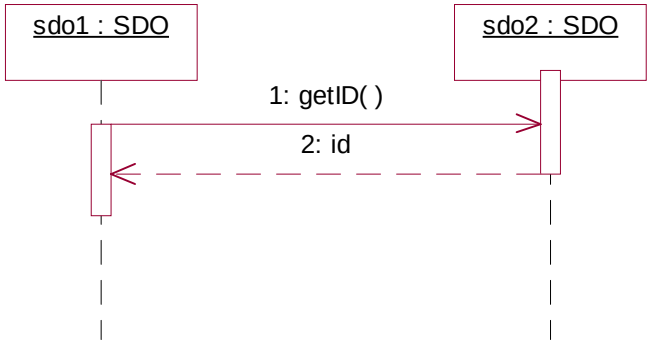


Figure 2-4 Sequence Chart: SDO operation concern with data resource

Message 1: sdo1 requests identifier of the sdo2. This identifier is described using String type.

Message 2: sdo2 returns the value of identifier.

2.3.4 Configuration Interface

Configuration interface provides operations to add or remove data specified in resource data model. These operations provide functions to change **DeviceProfile**, **ServiceProfile**, **ConfigurationProfile**, and **Organization**.

Configuration
+ setDeviceProfile(dProfile : DeviceProfile) : void + addServiceProfile(sProfile : ServiceProfile) : void + addOrganization(organization : Organization) : void + removeDeviceProfile() : void + removeServiceProfile(id : String) : void + removeOrganization(organization : Organization) : void + getConfigParameter() : ParameterList + getParameterValue(name : String) : any + setParameterValue(name : String, value : any) : void + getConfigurationSets() : ConfigurationSetList + addConfigurationSet(configurationSet : ConfigurationSet) : void + removeConfigurationSet(configID : String) : void + activateConfigurationSet(configID : String) : void

2.3.4.1 Operations

(1) + setDeviceProfile (dProfile: DeviceProfile) : void

This operation sets **DeviceProfile** object to the SDO that has this **Configuration** interface.

Parameter	Type	Description
dProfile	**DeviceProfile**	The device profile that is to be assigned to this SDO.

Exceptions

This operation throws **SDOException** with the following type:

- SDONotExists type SDOException if the target SDO does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.
- InvalidParameter type SDOException if argument "**dProfile**" is null, or if the object which is specified by argument "**dProfile**" does not exist.

(2) + addServiceProfile (sProfile : ServiceProfile) : void

This operation adds **ServiceProfile** to the target SDO that navigates this **Configuration** interface. If the id in argument **ServiceProfile** is null, new id is created and the **ServiceProfile** is stored. If the id is not null, the target SDO searches for **ServiceProfile** in it with the same id. It adds the **ServiceProfile** if not exist, or overwrites if exist.

Parameter	Type	Description
sProfile	**ServiceProfile**	**ServiceProfile** to be added.

Exceptions

This operation throws **SDOException** with the following type:

- SDONotExists type SDOException if the target SDO does not exist.
- InvalidParameter type SDOException if argument "sProfile" is null, or if the object which is specified by argument "sProfile" does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(3) + addOrganization (organization : Organization) : void

This operation adds **Organization** object to the SDO that has this **Configuration** interface. The **Organization** object to be added is specified by argument.

Parameter	Type	Description
organization	**Organization**	Organization to be added.

Exceptions

This operation throws **SDOException** with the following type:

- SDONotExists type SDOException if the target SDO does not exist.
- InvalidParameter type SDOException if argument "organization" is null, or if the object which is specified by argument "organization" does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(4) + removeDeviceProfile ( ) : void

This operation removes **DeviceProfile** object to the SDO that has this **Configuration** interface.

Parameter	Type	Description
none		

Exceptions

This operation throws **SDOException** with the following type.

- SDONotExists type SDOException if the target SDO does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.
- InvalidParameter type SDOException if argument "**dProfile**" is null, or if the object which is specified by argument "**dProfile**" does not exist.

(5) + removeServiceProfile (id : String) : void

This operation removes **ServiceProfile** object to the SDO that has this **Configuration** interface. The ServiceProfile object to be removed is specified by argument.

Parameter	Type	Description
id	**String**	**serviceID** of a **ServiceProfile** to be removed.

Exceptions

This operation throws **SDOException** with the following type:

- SDONotExists type SDOException if the target SDO does not exist.
- InvalidParameter type SDOException if argument "**sProfile**" is null, or if the object which is specified by argument "**sProfile**" does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(6) + remove Organization (organization: Organization) : void

This operation removes **Organization** object to the SDO that has this **Configuration** interface. The **Organization** object to be removed is specified by argument.

Parameter	Type	Description
organization	**Organization**	Organization to be removed.

Exceptions

This operation throws **SDOException** with the following type:

- SDONotExists type SDOException if the target SDO does not exist.
- InvalidParameter type SDOException if argument "**organization**" is null, or the object which is specified by argument "**organization**" does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(7) getConfigParameter () : ParameterList

This operation returns a list of Parameters.

Parameter	Type	Description
<return>	**ParameterList**	The list with definitions of parameters characterizing the configuration.

Exceptions

This operation throws **SDOException** with the following type:

- SDONotExists type SDOException if the target SDO does not exist.
- InvalidParameter type SDOException if the target SDO has no parameters to be configured.
- NotAvailable type SDOException if there is no response from the target SDO.

(8) + getParameterValue (name : String) : any

This operation returns a value of parameter that is specified by argument “name.”

Parameter	Type	Description
name	**String**	Name of the parameter whose value is requested.
<return>	**any**	The value of the specified parameter.

Exceptions

This operation throws **SDOException** with the following type.

- SDONotExists type SDOException if the target SDO does not exist.
- InvalidParameter type SDOException if the value of the argument "name" is empty String, or null, or if the parameter that is specified by argument "name" does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(9) + setConfigParameter(sdoID: String, name:String, value:any):void

This operation sets a parameter to a value that is specified by argument “**value**.” The parameter to be modified is specified by argument “**name**”.

Parameter	Type	Description
name	**String**	The name of parameter to be modified.
value	**any**	New value of the specified parameter.

Exceptions

This operation throws **SDOException** with the following type:

- SDONotExists type SDOException if the target SDO does not exist.
- InvalidParameter type SDOException if arguments ("name" and/or "value") is null, or if the parameter that is specified by the argument "name" does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(10) + getConfigurationSets () : ConfigurationSetList

This operation returns a list of **ConfigurationSets** that the **ConfigurationProfile** has.

Parameter	Type	Description
<return>	**ConfigurationSetList**	The list of stored configuration with their current values.

Exceptions

This operation throws **SDOException** with the following type.

- SDONotExists type SDOException if the target SDO does not exist.
- InvalidParameter type SDOException if the list of **ConfigurationSet** objects possessed by the SDO is empty.
- NotAvailable type SDOException if there is no response from the target SDO.

(11) + addConfigurationSet (configurationSet : ConfigurationSet) : void

This operation adds a **ConfigurationSet** to the **ConfigurationProfile**.

Parameter	Type	Description
configurationSet	**ConfigurationSet**	The **ConfigurationSet** that is added.

Exceptions

This operation throws **SDOException** with the following type:

- SDONotExists type SDOException if the target SDO does not exist.
- InvalidParameter type SDOException if the argument "configurationSet" is null, or if the object which is specified by this arguments does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(12) + removeConfigurationSet (configurationSetID : String) : void

This operation removes a **ConfigurationSet** from the **ConfigurationProfile**.

Parameter	Type	Description
configurationSetID	**String**	The **ConfigurationSet** that is removed.

Exceptions

This operation throws **SDOException** with the following type:

- SDONotExists type SDOException if the target SDO does not exist.
- InvalidParameter type SDOException if Arguments ("**configurationSet**") is null, or if the object that is specified by arguments ("**configurationSet**") does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(13) + activateConfigurationSet (configID : String) : void

This operation activates one of the stored **ConfigurationSets** in the **ConfigurationProfile**.

Parameter	Type	Description
configID	**String**	Identifier of **ConfigurationSet** to be activated.

Exceptions

This operation throws **SDOException** with the following type:

- SDONotExists type SDOException if the target SDO does not exist.
- InvalidParameter type SDOException if the argument ("**configID**") is null or there is no configuration set with identifier specified by the argument.
- NotAvailable type SDOException if there is no response from the target SDO.

2.3.4.2 *Usage: Configuration*

As an example, the message sequence chart for the case of profile acquisition is shown in Figure 2-5. And, as an example of parameter acquisition, the sequence of "**getConfigParameter()**" is shown in Figure 2-6. First, the configuring SDO (sdo1) gets the **Configuration** object by invocation of operations of the SDO which is configured (sdo2). And the sequences of the other operations which belong to the **Configuration** interface as shown in Figure 2-5and Figure 2-6.

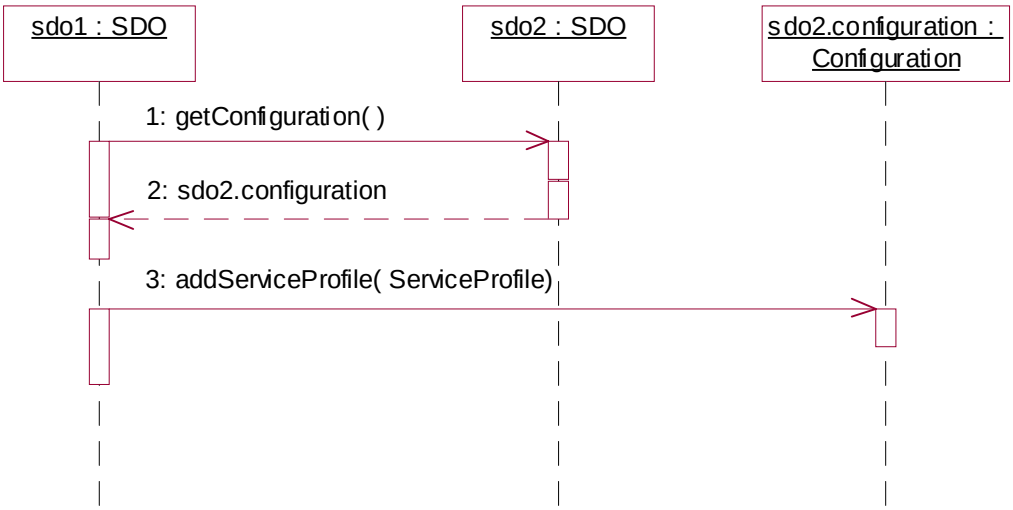


Figure 2-5 Sequence Chart: Configuration: Add Service Profile

Message 1: the configuring SDO (sdo1) gets the object implementing the **Configuration** of the SDO that is being configured (sdo2).

Message 2: sdo2 returns the object sdo2.configuration that implements the **Configuration**.

Message 3: sdo1 adds the ServiceProfile object to sdo2.

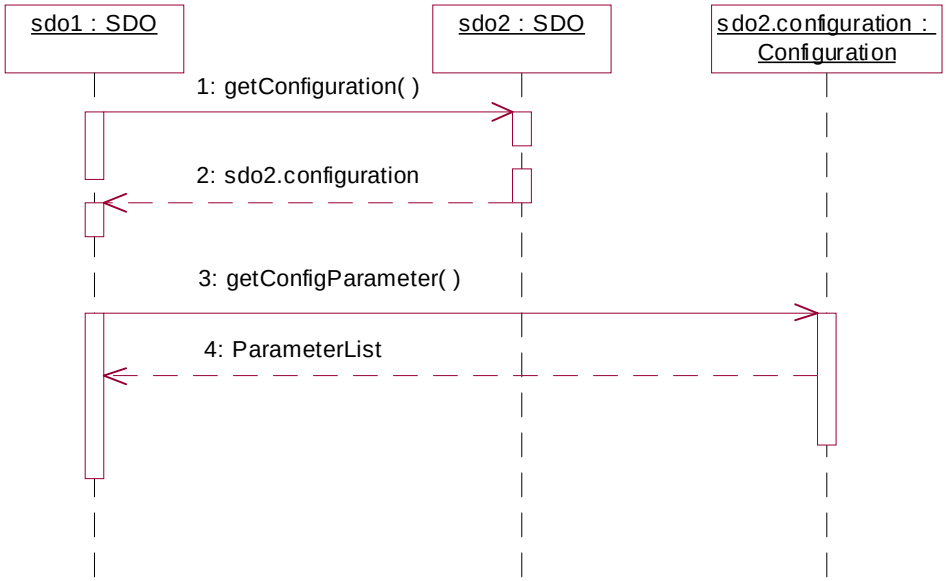


Figure 2-6 Sequence Chart: Configuration

Message 1: the configuring SDO (sdo1) gets the object implementing the **Configuration** of the SDO that is being configured (sdo2).

Message 2: sdo2 returns the object sdo2.configuration that implements the **Configuration**.

Message 3: sdo1 requests parameter list of sdo2.

Message 4: sdo2 returns the parameters as list of names and values of properties represented as NVList object.

2.3.5 SDOService Interface

SDOService interface provides operations for the services that are provided by functions of an SDO.

The interface of a service differs for every service. So, this interface does not have the fixed model.

2.3.6 Monitoring Interface

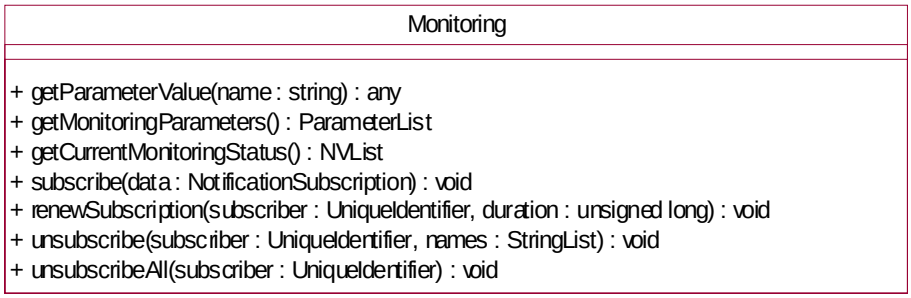
Each SDO is characterized by properties. These properties can depict the current state of an SDO (subject to monitoring) and can be used to control the SDO behavior (subject to configuration). Each SDO can have different number and different kinds of attributes. It is dependent upon the actual implementation which properties are provided by an SDO.

The **Monitoring** interface provides mechanisms to monitor the properties of an SDO. Each SDO implementing the **Monitoring** interface must specify the properties that can be monitored.

The properties of an SDO can be monitored mainly in two different ways: by *polling* and by *subscription*.

Polling is the simpler way of monitoring. The observer requests the current values of the properties it is interested in. The SDO that wants to monitor one or more properties must send a request message to the particular SDO. The Monitoring interface provide functions (**getCurrentMonitoringStatus()**, and **getParameterValue()**) that support the monitoring by polling. The monitoring by polling is described detailed in Section 2.3.6.4.1, “Monitoring by Polling,” on page 2-36.

Using *subscription* an observing SDO is notified about changes of monitored properties. The observing SDO has to subscribe to an SDO which is to be monitored. According to the subscription the monitored SDO notifies the subscriber using its Call-back interface (see Section 2.3.6.5, “NotificationCallback Interface,” on page 2-42). The Monitoring interface supports the subscription through appropriate functions (**subscribe()**, **renewSubscription()**, **unsubscribe(), unsubscribeAll()**). The monitoring by subscription is described in detail in Section 2.3.6.4.2, “Monitoring by Subscription,” on page 2-38.



The **Monitoring** interface is optional. As mentioned earlier each SDO specifies the properties that can be monitored. If an SDO does not want to provide any of its properties to be monitored, it can do so and in this case it does not need to implement the **Monitoring** interface.

2.3.6.1 *Data Structures Defined for Monitoring Interface*

Various data structures are defined to implement the **Monitoring** interface. This section defines all such data structures including the general data structures and data structures required only for the **Monitoring** interface. All such data structures are listed in Table 2-1 and later on described in detail individually.

Table 2-1 Detailed Description of Data Structures Required for **Monitoring** Interface

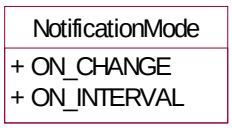
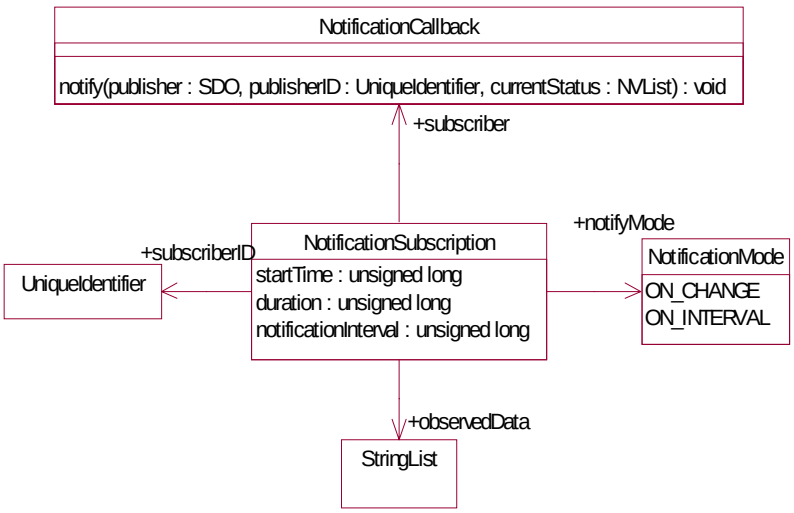
NotificationMode	Possible notification modes while subscribing to monitoring properties.  **ON_CHANGE** - To be notified of the subscribed monitoring property every time the value of the parameter changes. **ON_INTERVAL** - To be notified of the subscribed monitoring property on the specified interval of time. That is, if a subscription to some property is made in this mode, the notification is sent to the subscribing SDO only at the specified time with the current value.
NotificationSubscription	 This structure outlines the details of the subscription message. This message is received and evaluated by the SDO providing properties for monitoring. Depending on the notification mode, the subscriber (the message sender) will receive appropriate messages containing the property value periodically or if the property's value changes.

Table 2-1 Detailed Description of Data Structures Required for **Monitoring** Interface

	• subscriber - The address or reference of the object that will receive notification messages; the object referenced here must implement the interface **NotificationCallback**. The value of this field depends on basis technology of SDO system. Please read Section 2.3.6.5, "NotificationCallback Interface," on page 2-42 for more details. • **subscriberID** - The unique identifier of the SDO that subscribes to this property. • **notifyMode** - The mode of notification (**ON_CHANGE** or **ON_INTERVAL**). The notifications are sent either when the value of at least one of subscribed monitoring properties changes (notification on change), or periodically. In this case the frequency of notifications is specified with the attribute notificationInterval. • **observedData** - List of names of monitoring properties to be subscribed. • **startTime** - Defines the time period (in milliseconds) at which monitoring of the properties should start. If it is not specified, then the subscription will be activated right after receiving the subscription message. • **duration** - Indicates for how long (in milliseconds) the subscription should be last. • **notificationInterval** - Time interval (in milliseconds) in which the notification is sent to the subscribing SDO, if the **NotificationMode** of the subscription is **ON_INTERVAL**.

2.3.6.2 Operations provided by Monitoring Interface

This section describes all the operations provided by the **Monitoring** interface. All operations are initially listed in Table 2-2 and then each of them is described with examples.

Table 2-2 Operations provided by Monitoring Interface

Name	Short Description
getParameterValue (name : String)	To get the value of the specified property.
getMonitoringParameters () : ParameterList	To get the list of all monitoring properties.
getCurrentMonitoringStatus () : NVList	To get all the monitoring properties with their current values.
subscribe (data : NotificationSubscription) : void	This operation subscribes necessary monitoring properties with certain conditions.
renewSubscription (subscriber : UniqueIdentifier, duration : unsigned long) : void	This operation renews the already subscribed properties for the specified duration of time.

Table 2-2 Operations provided by Monitoring Interface

unsubscribe (subscriber : UniqueIdentifier, names : StringList) : void	This operation unsubscribes the specified list of already subscribed monitoring properties.
unsubscribeAll (subscriber : UniqueIdentifier) : void	This operation unsubscribes all the subscribed properties of the specified SDO.

2.3.6.3 Detailed description of all operations

In this section all the operations introduced above are described in detail.

(1) +getParameterValue (name : String) : any

This operation returns the current value of the specified monitoring property.

Parameter	Type	Description
name	**String**	Name of the property whose value is requested.
<return>	**any**	The current value of the property.

Exceptions

This operation throws **SDOException** with the following type:

- InvalidParameter type SDOException if the argument ‘name’ is null or a monitoring property named ‘name’ does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(2) +getMonitoringParameters () : ParameterList

This operation returns the list of monitoring properties defined for this SDO.

Parameter	Type	Description
<return>	**ParameterList**	List of containing names and types of monitoring properties of the SDO.

Exceptions

This operation throws **SDOException** with the following type.

- InvalidParameter type SDOException if the list of properties that can be monitored is empty.
- NotAvailable type SDOException if there is no response from the target SDO.

(3) +getCurrentMonitoringStatus () : NVList

This operation returns the current values of all the monitoring properties of the SDO.

Parameter	Type	Description
<return>	**NVList**	The list containing names and current values of all monitoring properties of the SDO.

Exceptions

This operation throws **SDOException** with the following type.

- InvalidParameter type SDOException if the list of properties that can be monitored is empty.
- NotAvailable type SDOException if there is no response from the target SDO.

(4) +subscribe (data : NotificationSubscription) : void

This operation subscribes necessary monitoring properties with certain conditions. The details of the notification are denoted in the argument **data**. When a subscription request arrives, the SDO may add the subscriber to its internal table of subscribers. The subscriptions in the table can be distinguished by the identifier of the subscriber and the name of subscribed property.

Parameter	Type	Description
data	**NotificationSubscription**	Properties being subscribed and the conditions for the subscription.

Exceptions

This operation throws **SDOException** with the following type.

- InvalidParameter type SDOException if the condition specified for the subscription is not valid. For example, if the mode of subscription is ON_INTERVAL and the attribute ‘notificationInterval’ is not defined in parameter NotificationSubscription. This exception arises also if the properties to be subscribed defined in NotificationSubscription.observedData do not exist. This exception is thrown even if one of the defined properties does not exist. In this case, the name of non-existing property must be specified in the exception data.
- NotAvailable type SDOException if there is no response from the target SDO.

(5) +renewSubscription (subscriber : UniqueIdentifier, duration : unsigned long) : void

This operation renews the already subscribed properties for the specified duration of time. The subscription time is extended for all properties that were subscribed previously by the specified SDO.

Parameter	Type	Description
subscriber	**UniqueIdentifier**	Unique ID of the SDO that is renewing the subscription.
duration	**unsigned long**	Time duration until which the subscriptions of the specified SDO should be renewed.

Exception

This operation throws **SDOException** with the following type.

- InvalidParameter type SDOException if there are no subscriptions from the observer SDO defined by the parameter *subscriber* to renew.
- NotAvailable type SDOException if there is no response from the target SDO.

(6) +unsubscribe (subscriber : UniqueIdentifier, names : StringList) : void

This operation unsubscribes the specified list of already subscribed monitoring properties.

Parameter	Type	Description
subscriber	**UniqueIdentifier**	Unique ID of the SDO that is unsubscribing.
names	**StringList**	List of names of the properties being unsubscribed.

Exception

This operation throws **SDOException** with the following type.

- InvalidParameter type SDOException if the stated properties (parameter **names**) to be unsubscribed does not exist or was not subscribed. This exception is thrown even if one of the defined properties does not exist or was not subscribed. The properties that do not exist or were not subscribed must be specified in the exception data.
- NotAvailable type SDOException if there is no response from the target SDO.

(7) +unsubscribeAll (subscriber : UniqueIdentifier) : void

This operation unsubscribes all the subscribed properties of the specified SDO.

Parameter	Type	Description
subscriber	**UniqueIdentifier**	Unique ID of the SDO that is unsubscribing all its subscriptions.

Exception

This operation throws **SDOException** with the following type.

- InvalidParameter type SDOException if there are no subscriptions from the observer SDO defined by the parameter **subscriber** to unsubscribe.
- NotAvailable type SDOException if there is no response from the target SDO.

2.3.6.4 Usage: Monitoring Interface

2.3.6.4.1 Monitoring by Polling

SDOs can get information on status of monitoring parameters of other SDO without subscription to event notifications as well. Their status can be obtained by requesting the SDO.

The operations that can be invoked to monitor the status of the SDO are shown in Figure 2-7. In the diagram shown in the picture, sdo1 makes requests to sdo2 to acquire its status.

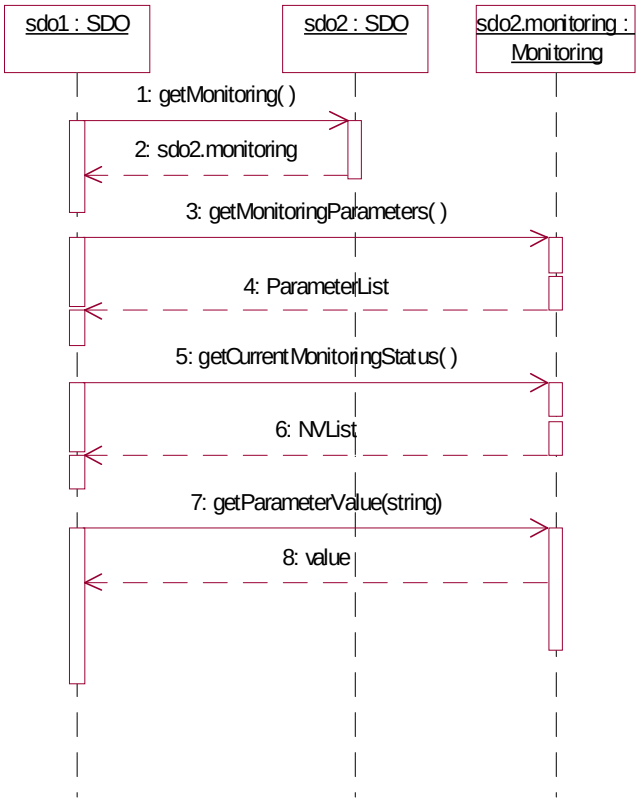


Figure 2-7 Sequence Chart: Monitoring

Message 1: the monitoring SDO (sdo1) gets the object implementing the **Monitoring** interface of the monitored SDO (sdo2).

Message 2: sdo2 returns the object sdo2.monitoring that implements the **Monitoring** interface.

Message 3: sdo1 requests the list of all monitoring properties of sdo2.

Message 4: sdo2 sends response, containing the list of monitoring properties specifying the sdo2.

Message 5: sdo1 requests the current values of monitoring parameters of the sdo2.

Message 6: sdo2 returns the data requested in message 5 as list of names and current values of properties represented as **NVList** object. Knowing the respective types of status properties, their values can be interpreted properly.

Message 7: sdo1 requests the current value of a particular monitoring property.

Message 8: sdo2 sends back reply containing the current property value. The value should be interpreted according to the type of the property.

2.3.6.4.2 Monitoring by Subscription

Monitoring by subscription uses time-limited subscriptions. When an observer SDO subscribes to certain data of a publisher SDO, the subscription message includes the validity (time period) of this subscription. The observer SDO knows when it will expire, and it just renews the subscription shortly before it expires. The publisher SDO checks every now and then if the subscription is still valid. If the subscription is already expired, it simply removes the subscription. It is implementation detail how long should be the time validity of the subscription. The time validity of the subscription can either be predefined by the system or defined by the subscriber SDO on its own. The shorter the time duration, the more often observer SDOs should send renewal messages. The time-limited subscriptions are advantageous in cases when for some reason observer SDOs are out of the system without being able to send notification that they are leaving.

Monitoring by subscription provides two different modes. These modes indicate when the monitoring SDOs should be notified:

- when the value of the monitored properties change (**ON_CHANGE** mode), or
- on certain time interval (**ON_INTERVAL** mode).

When subscribing, an SDO has to specify the mode and the properties it wants to monitor. Once subscribed the observing SDO (subscriber) is notified based upon the conditions specified. Since the subscriber is notified it must provide a callback interface, which is described in Section 2.3.6.5, “NotificationCallback Interface,” on page 2-42.

Subscribing with **ON_INTERVAL** notification mode may have advantage against subscribing **ON_CHANGE** if the values of monitoring properties of sdo2 change very often and are sent very frequently.

ON_CHANGE mode

This section describes in detail the monitoring by subscription using **ON_CHANGE** mode. Monitoring **ON_CHANGE** basis is useful if the subscribing SDO wants to be notified as soon as one of the subscribed property values has changed. For example if the Airconditioner SDO described in Section 2.2.11, “Examples of resource data model,” on page 2-13“ subscribes temperature value in the Thermometer SDO **ON_CHANGE** basis, it receives notification about the changes in the temperature value every time the temperature value changes in the room.

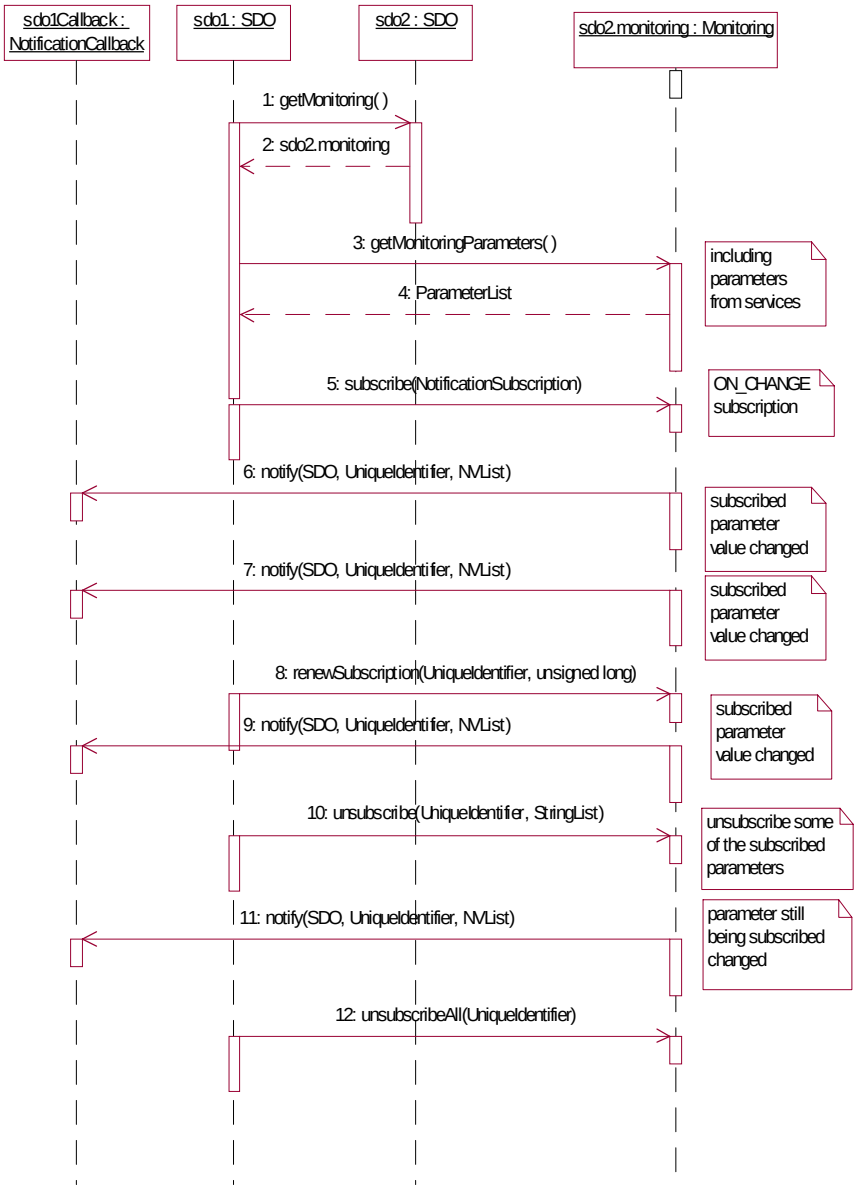


Figure 2-8 Subscription and Notification in ON_CHANGE mode

The subscription of properties is shown in the sequence diagram (Figure 2-8). The interaction depicted in Figure 2-8 occurs between two SDOs, sdo1 and sdo2. In the figure above, sdo1 intends to monitor the status of sdo2. sdo2 possesses an object that represents the **Monitoring** interface of sdo2. sdo1 implements the interface **NotificationCallback** to be able to receive notifications about status changes.

Message 1: the monitoring SDO (sdo1) gets the object implementing the **Monitoring** interface of the monitored SDO (sdo2).

Message 2: sdo2 returns the object sdo2.monitoring that implements the **Monitoring** interface.

Message 3: sdo1 requests the list of monitoring properties of sdo2.

Message 4: sdo2 sends response, containing the list of monitoring properties specifying the sdo2. The monitoring properties in list can represent the:

- properties of the SDO.
- properties of SDO services, offered for monitoring.

Message 5: sdo1 subscribes one or more properties of sdo2

Message 6, 7: when the values of one of these subscribed properties change, sdo1 gets a notification message from sdo2 with the name (or names) of the property that has changed along with its (or their) current values.

Message 8: shortly before the subscription expires, sdo1 sends a renewal request to extend the subscription time.

Message 9: since the subscription time is extended, the sdo1 continues receiving notifications about changes in status of sdo1.

Message 10: sdo1 unsubscribes one or more properties subscribed previously.

Message 11: sdo1 receives notification on changes in status of one of monitoring properties that remain in subscription.

Message 12: sdo1 unsubscribes all the properties it has subscribed at sdo2. Henceforth, it does not get any notifications on changes in status of monitoring properties of sdo2.

ON_INTERVAL mode

This section describes in detail the monitoring by subscription using **ON_INTERVAL** mode. Monitoring **ON_INTERVAL** mode is useful if the subscribing SDO wants to be notified periodically, because it may want to observe the development of a specific property over a longer period of time (even if the property's value does not change).

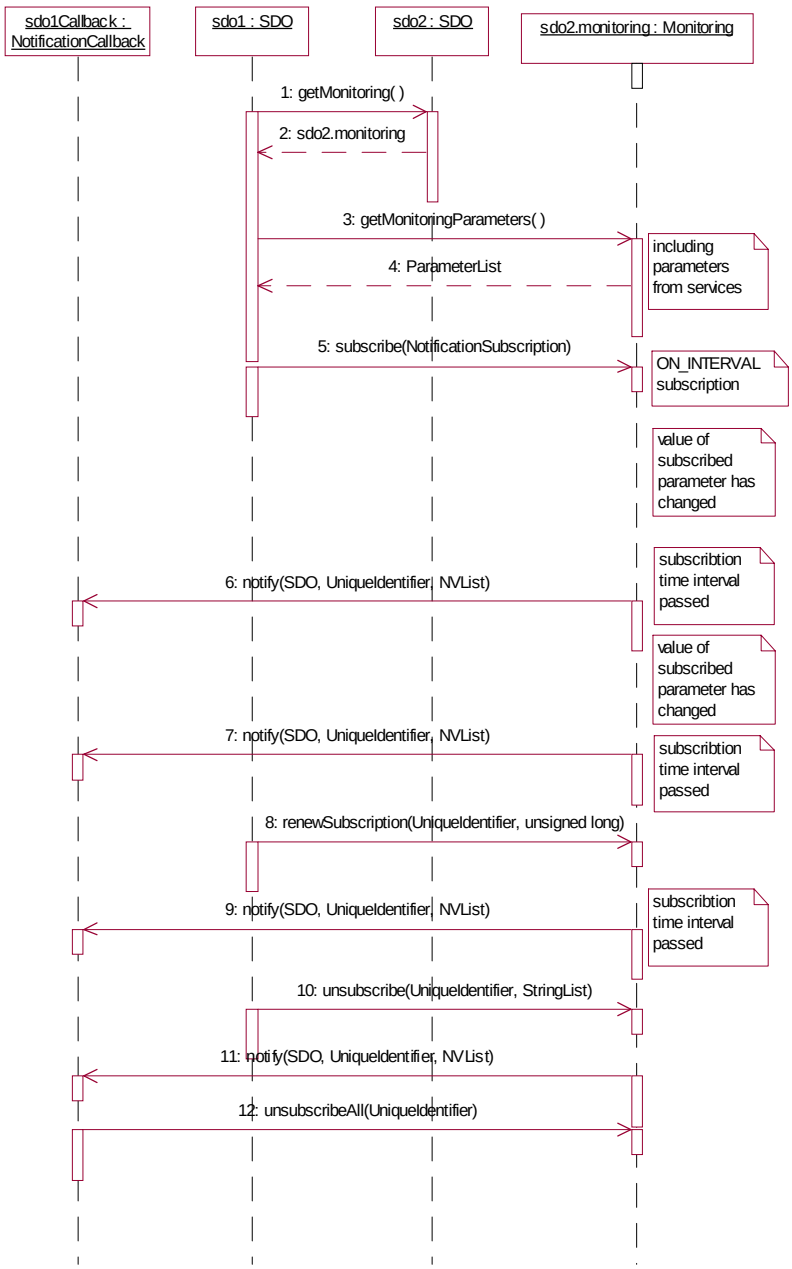


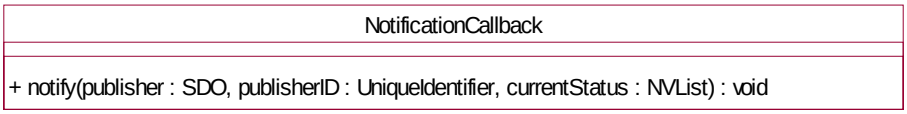
Figure 2-9 Subscription and Notification ON_INTERVAL

Figure 2-9 shows the notifications made in interval mode. In the diagram shown on the picture, sdo1 intends to monitor the status of sdo2.

In general, the same sequence of operations shown in Figure 2-8 is used to subscribe, renew, and cancel property notification. The difference is that notifications are sent not in the event when one of subscribed properties changes its value, but periodically in a specified time interval. Notifications contain the property names and their values at the moment when the notification was sent. Notifications are sent irrespectively of the fact whether the property values have changed or not since the last notification. So it can occur that between two notifications some properties have changed their values more than once or never at all. For example, in Figure 2-9 the property values change twice between notification messages 5 and 8. Furthermore, it can also occur that property values remain unchanged during several notification intervals, like in the sequence diagram in Figure 2-9 between notification messages 9 and 11.

2.3.6.5 NotificationCallback Interface

This interface provides call back mechanism for an SDO for the subscription notification. Monitoring properties of an SDO can be monitored by other SDOs by subscribing such properties. The changes in the monitored properties are subsequently notified to the subscribing SDOs. This call back interface will provide interface to notify subscribing SDOs.



2.3.6.6 Data Structures Defined for NotificationCallback Interface

Two general data structures (**UniqueIdentifier** and **NVList**) are used in the NotificationCallback interface. Please see Section 2.3.6.1, “Data Structures Defined for Monitoring Interface,” on page 2-31 for their detail description.

2.3.6.7 Operations provided by NotificationCallback Interface

(1) +notify(publisherID : UniqueIdentifier, currentStatus : NVList) : void

This operation notifies the subscriber that either the value of the property has changed or the notification interval has elapsed.

Parameter	Type	Description
publisherID	**UniqueIdentifier**	Unique identifier of the SDO that is sending the notification.

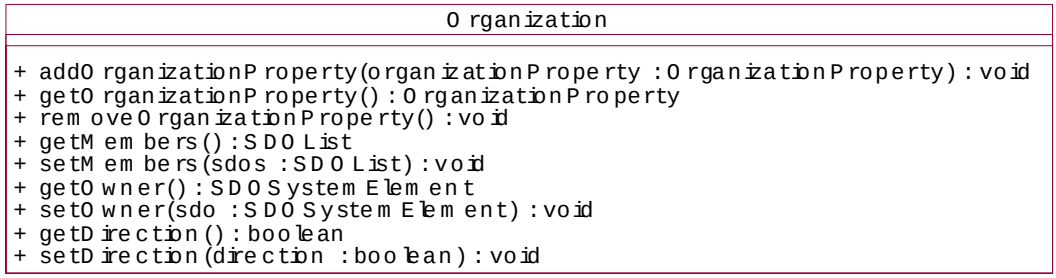
currentStatus	NVList	A list containing the properties and their current values. Please note that this list may not contain all the properties that an SDO has been subscribed to, probably because not all property has changed or because the notification interval of some properties has not elapsed yet.

2.3.6.8 *Usage: NotificationCallback Interface*

The examples of usage of operations of **NotificationCallback** interface are shown in Figure 2-8 and Figure 2-9. The operations are used to convey the changes in values of monitoring properties to notification subscriber.

2.3.7 *Organization Interface*

The **Organization** interface is used to manage the Organization attribute.



(1) *+ addOrganizationProperty (organizationProperty : OrganizationProperty) : void*

This operation adds the **OrganizationProperty** to an Organization. The **OrganizationProperty** is the property description of an Organization.

Parameter	Type	Description
organizationProperty	OrganizationProperty	The type of organization to be added.

Exception

This operation throws **SDOException** with the following type.

- InvalidParameter type SDOException if argument "**organizationProperty**" is null, or if the object which is specified by argument "**organizationProperty**" does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(2) + *removeOrganizationProperty () : void*

This operation removes the **OrganizationProperty** from an Organization.

Parameter	Type	Description
organizationProperty	**OrganizationProperty**	The type of organization to be removed.

Exception

This operation throws **SDOException** with the following type.

- InvalidParameter type SDOException if the Organization which the target SDO belong to has no organizationProperty.
- NotAvailable type SDOException if there is no response from the target SDO.

(3) + *getOrganizationProperty () : OrganizationProperty*

This operation returns the OrganizationProperty that an Organization has.

Parameter	Type	Description
<return>	**OrganizationPropertyList**	The list with properties of the organization.

Exception

This operation throws **SDOException** with the following type.

- InvalidParameter type SDOException if argument "**organizationProperty**" is null, or if the object which is specified by argument "**organizationProperty**" does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(4) + *getMembers ( ) : SDOList*

This operation returns a list of **SDO**s that are members of an Organization.

Parameter	Type	Description
<return>	**SDOList**	Member SDOs that are contained in the **Organization** object.

Exception

This operation throws **SDOException** with the following type.

- InvalidParameter type SDOException if the Organization which the target SDO belongs to has no members.
- NotAvailable type SDOException if there is no response from the target SDO.

(5) + setMembers (sdos : SDOList) : void

This operation sets **SDO**s as members of the organization. The **SDO**s to be set is specified by argument "**sdos**".

Parameter	Type	Description
sdos	**SDOList**	The **SDO**s to be added.

Exception

This operation throws **SDOException** with the following type.

- InvalidParameter type SDOException if argument "SDOList" is null, or if the object which is specified by the argument "sdos" does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(6) + getOwner ( ) : SDOSystemElement

This operation returns the **SDOSystemElement** that is owner of the Organization.

Parameter	Type	Description
<return>	**SDOSystemElement**	Reference of owner object.

Exception

This operation throws **SDOException** with the following type.

- InvalidParameter type SDOException if the target Organization has no owner.
- NotAvailable type SDOException if there is no response from the target SDO.

(7) + setOwner (sdo : SDOSystemElement) : void

This operation sets an **SDOSystemElement** to the owner of the Organization. The **SDOSystemElement** to be set is specified by argument "**sdo**".

Parameter	Type	Description
sdo	**SDOSystemElement**	Reference of owner object.

Exception

This operation throws **SDOException** with the following type.

- InvalidParameter type SDOException if argument "**sdo**" is null, or if the object which is specified by "**sdo**" in argument "**sdo**" does not exist.
- NotAvailable type SDOException if there is no response from the target SDO.

(8) + getDirection () : boolean

This operation gets the relationship direction of the Organization.

Parameter	Type	Description
<return>	**boolean**	direction.

Exception

This operation throws **SDOException** with the following type.

- InvalidParameter type SDOException if the value of "direction" is null.
- NotAvailable type SDOException if there is no response from the target SDO.

(9) + setDirection (direction : boolean) : void

This operation sets the relationship direction of the Organization. The value to be set is specified by argument "**direction**".

Parameter	Type	Description
direction	**boolean**	direction.

Exception

This operation throws **SDOException** with the following type.

- InvalidParameter type SDOException if argument "direction" is null.
- NotAvailable type SDOException if there is no response from the target SDO.

2.3.7.1 *Usage: Organization*

Figure 2-10 shows a sequence diagram to explain how to obtain a set of properties from an Organization. In this example, an SDO (sdo1) calls **getOrganizations()** on another SDO (sdo2) to obtain the Organization objects that sdo2 is associated with, chooses one of the obtained **Organization** objects (organization1), and then calls **getOrganizationProperty()** on organization1 in order to see its properties.

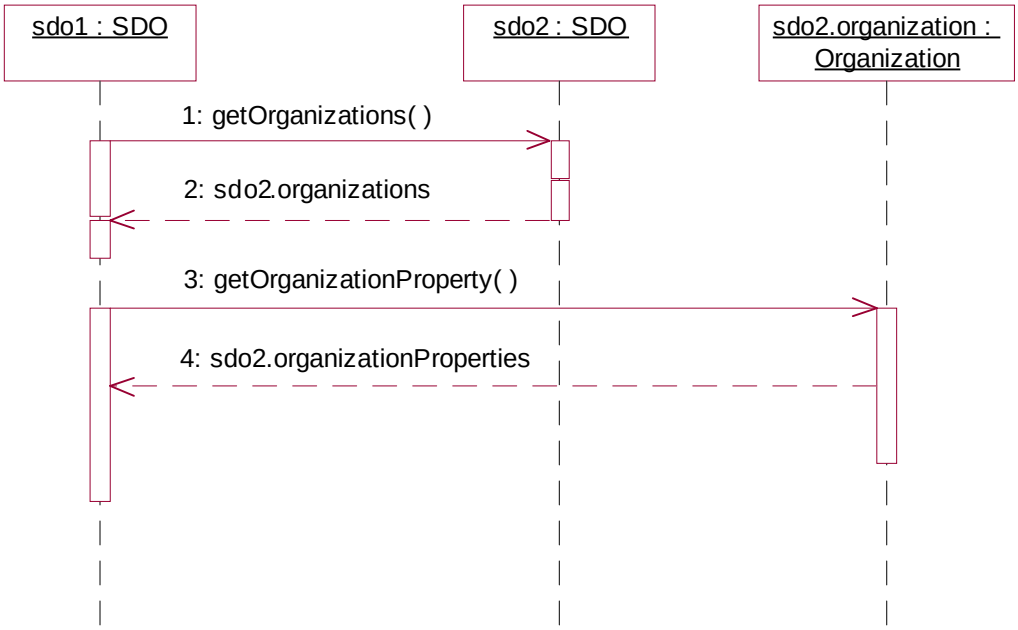


Figure 2-10 An example to obtain an OrganizationProperty

Message 1: An SDO (sdo1) asks another SDO (sdo2) to return the **Organization** objects that sdo2 is associated with.

Message 2: sdo2 returns sdo2.organizations, which is a list of **Organization** objects.

Message 3: sdo1 chooses one of the obtained **Organization** object (organization1) and requests its properties (i.e., organization1.property).

Message 4: organization1 returns organization1.property.

Platform Specific Model: Mapping to CORBA IDL 3

Contents

This chapter contains the following sections.

This chapter introduces a CORBA specific model for the SDO PIM defined in Chapter 2. The selected platform is CORBA version 3.0.

The SDO PIM defines the resource data model, interfaces, and necessary data structures for SDOs. In the PSM these interfaces and the data structures used in the individual methods are mapped to an according CORBA IDL specification. The complete IDL specification is presented in Chapter 4.

An interface defined in the SDO PIM is mapped to a CORBA interface. An operation in a PIM interface is mapped to an CORBA operation. A private attribute in a PIM interface is mapped to an operation named get_<attribute name>. A public attribute in an interface is mapped to two operations; get_<attribute name> and set_<attribute name>. An PIM exception is mapped to a CORBA exception. The other data types in the SDO PIM (e.g., resource data) are mapped to the non-interface types in CORBA IDL. The CORBA PSM is compliant with the IDL style guide [3].

3.1 SDO Module

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

3.2 Data types used in CORBA PSM

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

```
typedef sequence<string>            StringList;
typedef sequence<SDO>               SDOList;
typedef sequence<Organization>      OrganizationList;
typedef string                      UniqueIdentifier;
    struct NameValue {
        string name;
        any value;
     };
typedef sequence<NameValue>   NVList;
    enum NumericType {
        SHORT_TYPE,
        LONG_TYPE,
        FLOAT_TYPE,
        DOUBLE_TYPE};
    union Numeric switch (NumericType) {
        case SHORT_TYPE: short short_value;
        case LONG_TYPE: long long_value;
        case FLOAT_TYPE: float float_value;
        case DOUBLE_TYPE: double double_value;
    };
    struct EnumerationType {
        StringList enumerated_values;
    };
    struct RangeType {
        Numeric min;
        Numeric max;
        boolean min_inclusive;
        boolean max_inclusive;
    };
    struct IntervalType {
        Numeric min;
        Numeric max;
        boolean min_inclusive;
        boolean max_inclusive;
        Numeric step;
    };
    enum ComplexDataType {ENUMERATION, RANGE, INTERVAL};
    union AllowedValues switch (ComplexDataType) {
        case ENUMERATION: EnumerationType allowed_enum;
        case INTERVAL: IntervalType allowed_interval;
```

```
        case RANGE: RangeType   allowed_range;
    };
    struct Parameter {
        string        name;
        CORBA::TCKind  type;
        AllowedValues allowed_values;
    };
    typedef sequence<Parameter> ParameterList;
    struct OrganizationProperty {
        NVList properties;
    };
    enum DependencyType {
        NORMAL,
        REVERSE,
        NO_DEPENDENCY
    };
    struct DeviceProfile {
        string device_type;
        string manufacturer;
        string model;
        string version;
        NVList properties;
    };
    struct ServiceProfile {
        string id;
        string interface_type;
        NVList properties;
        SDOService service;
    };
    typedef sequence<ServiceProfile> ServiceProfileList;
    struct ConfigurationSet {
        string id;
        string description;
        NVList configuration_data;
};
typedef sequence<ConfigurationSet> ConfigurationSetList;
```

3.3 Exceptions

The methods of SDO interfaces can raise SDOException. The kind of exception is defined in the attribute *type* (see Section 2.3.2.1, “SDOException,” on page 2-16). This exception is mapped to several CORBA exceptions. All defined exceptions have structure specified by a macro **exception_body**. Five exceptions are defined in this specification: **NotAvailable**, **InterfaceNotImplemented**, **InvalidParameter**, and **NotFound**.

```
#define exception_body { string description; }
…
exception NotAvailable                exception_body;
exception InterfaceNotImplemented  exception_body;
```

exception InvalidParameter exception_body;

The exception SDONotExists defined in the PIM is mapped to CORBA standard system exception OBJECT_NOT_EXIST.

3.4 Interfaces

The SDO PIM defines several interfaces that can be implemented by an SDO. The SDO interface is a mandatory interface, whereas Configuration and Monitoring including NotificationCallback are optional interfaces. This means each SDO implementation has at least to implement the SDO interface and may additionally implement the other interfaces.

In the CORBA model all interfaces as defined in the SDO 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 SDO IDL specification includes following interfaces declarations:

- interface **SDOSystemElement**
- interface **SDO**
- interface **SDOService**
- interface **Configuration**
- interface **Monitoring**
- interface **Organization**

3.4.1 SDOSystemElement Interface

The **SDOSystemElement** interface is mapped to an CORBA interface. Interfaces of objects that represent elements of SDO system, such as SDOs, have to be derived from this interface. Therefore, the SDO interface inherits this interface. It is reserved for future extension to include further elements of SDO systems beside the actual SDOs.

The **SDOSystemElement** interface support an operation, **get_organizations**, which allows to get the list of organizations associated with the object implementing this interface.

```
interface SDOSystemElement {
    OrganizationList get_organizations()
        raises (NotAvailable);
};
```

3.4.2 SDO Interface

The **SDO** interface in the PIM is mapped directly to a CORBA interface. It inherits the **SDOSystemElement** interface.

```
interface SDO : SDOSystemElement {
    UniqueIdentifier get_id()
        raises (NotAvailable);
    string get_SDO_type()
        raises (NotAvailable);
    DeviceProfile get_device_profile ()
        raises (NotAvailable);
    ServiceProfileList get_service_profiles ()
        raises (NotAvailable);
    ServiceProfile get_service_profile (
        in string id
        ) raises (InvalidParameter, NotAvailable);
    SDOService get_service (
        in string id
        ) raises (InvalidParameter, NotAvailable);
    Configuration get_configuration ()
        raises (InterfaceNotImplemented, NotAvailable);
    Monitoring get_monitoring ()
        raises (InterfaceNotImplemented, NotAvailable);
};
```

3.4.3 Configuration Interface

The **Configuration** interface in the PIM is mapped directly to a CORBA interface:

```
interface Configuration {
    void set_device_profile (in DeviceProfile dProfile)
        raises (InvalidParameter, NotAvailable);
    void add_service_profile (in ServiceProfile sProfile)
        raises (InvalidParameter, NotAvailable);
    void add_organization (in Organization organization)
        raises (InvalidParameter, NotAvailable);
    void remove_device_profile ()
        raises (NotAvailable);
    void remove_service_profile (in string id)
        raises (InvalidParameter, NotAvailable);
    void remove_organization (in Organization organization)
        raises (InvalidParameter, NotAvailable);
    ParameterList get_config_parameters ()
        raises (NotAvailable);
    any get_parameter_value (in string name)
        raises (InvalidParameter, NotAvailable);
    void set_config_parameter (
        in string name,
        in any value)
        raises (InvalidParameter, NotAvailable);
    ConfigurationSetList get_configuration_sets ()
        raises (NotAvailable);
    void add_configuration_set (in ConfigurationSet configuration_set)
        raises (InvalidParameter, NotAvailable);
```

```
        void remove_configuration_set (in string config_id)
            raises (InvalidParameter, NotAvailable);
        void activate_configuration_set (in string config_id)
            raises (InvalidParameter, NotAvailable);
};
```

3.4.4 SDOService

In the PSM, SDO services are represented by CORBA objects. The class **SDOService** is mapped to an empty IDL interface. When implementing real services, their interfaces should be derived from the **SDOService** interface.

3.4.5 Monitoring Interface

The interface **Monitoring** in the PIM is mapped to a CORBA interface. The operations that enable to obtain the list of monitoring parameters supported by the SDO and their current values are mapped straight forward to operations of **Monitoring** Interface. The subscription and notification mechanisms described in Section 2.3.6, “Monitoring Interface,” on page 2-29 are implemented using the OMG Notification Service [4].

```
interface Monitoring : CosNotifyComm::StructuredPushConsumer,
                       CosNotifyComm::StructuredPushSupplier {
    any get_parameter_value (
        in string name
        ) raises (InvalidParameter, NotAvailable);
    ParameterList get_monitoring_parameters ()
        raises (NotAvailable);
    NVList get_current_monitoring_status ()
        raises (NotAvailable);
};
```

To use the mechanisms defined in Notification Service, the Monitoring interface inherits the interfaces **StructuredPushSupplier** and **StructuredPushConsumer**, defined in **CosNotifyComm** module.

The interface **StructuredPushSupplier** enables SDOs to publish event notifications to the Notification Service event channel (referred henceforth as the *notification channel*). This interface supports the behavior of objects that send Structured Events into the notification channel using push-style communication. (Models of event propagation are described in [5].) The operation **subscription_change** enables a notification consumer to inform an instance supporting this interface whenever there is a change to the types of events it is interested in receiving. The operation **disconnect_structured_push_supplier** is invoked to terminate a connection between the target **StructuredPushSupplier**, and its associated consumer. The operations of the interface **StructuredPushSupplier** cover the group of subscription operations defined for Monitoring interface in Section 2.3.6.2, “Operations provided by Monitoring Interface,” on page 2-32.

The interface **StructuredPushConsumer** enables the notification channel to send SDOs status notifications supplied by event suppliers as Structured Events by the push model, using the operation **push_structured_event**. The operation **offer_change** enables a notification supplier to inform an instance supporting this interface whenever there is a change to the types of events it intends to produce. The interface **StructuredPushConsumer** provides functionality that covers the functionality of NotificationCallback interface defined in Section 2.3.6.7, “Operations provided by NotificationCallback Interface,” on page 2-42.

3.4.6 Organization Interface

The **Organization** interface is mapped in the PSM to a CORBA interface. The class attributes are mapped to interface operations. For example, the attribute members is mapped to the operation pair **getMembers()** and **setMembers()**. It should also be noticed that both these operations work with lists of references of SDOs that belong to the organization. The operations **getOwner()** and **setOwner()** manipulate the reference of an object that owns the organization.

```
interface Organization {
    void add_organization_property (
        in OrganizationProperty organization_property
    ) raises (InvalidParameter, NotAvailable);
    void remove_organization_property ()
        raises (NotAvailable);
    OrganizationProperty get_organization_property ()
        raises (NotAvailable);
    SDOList get_members ()
        raises (NotAvailable);
    void set_members (
        in SDOList sdos
        ) raises (InvalidParameter, NotAvailable);
    SDOSystemElement get_owner ()
        raises (NotAvailable);
    void set_owner (
        in SDOSystemElement sdo
        ) raises (InvalidParameter, NotAvailable);
    boolean get_direction()
        raises (NotAvailable);
    void set_direction (
        in boolean direction
        ) raises (NotAvailable);
};
```

OMG IDL 4

4.1 SDO Package

```
// SDOPackage.idl

#ifndef    _SDO_PACKAGE_IDL_
#define    _SDO_PACKAGE_IDL_

#include <corba.idl>
#include <CosNotifyComm.idl>

/** CORBA specific model for SDOs */

#pragma prefix "org.omg"
#define exception_body { string description; }

module SDOPackage {
     interface SDO;
     interface SDOService;
     interface SDOSystemElement;
     interface Configuration;
     interface Monitoring;
     interface Organization;

     /** ------- Data Types -------*/
     typedef sequence<string>          StringList;
     typedef sequence<SDO>             SDOList;
     typedef sequence<Organization>  OrganizationList;
     typedef string                    UniqueIdentifier;
          struct NameValue {
             string name;
             any value;
  };
```

```
typedef sequence<NameValue>   NVList;
enum NumericType {
     SHORT_TYPE,
     LONG_TYPE,
     FLOAT_TYPE,
     DOUBLE_TYPE};
union Numeric switch (NumericType) {
     case SHORT_TYPE:  short short_value;
     case LONG_TYPE: long long_value;
     case FLOAT_TYPE: float float_value;
     case DOUBLE_TYPE: double double_value;
};
struct EnumerationType {
     StringList enumerated_values;
};
struct RangeType {
     Numeric min;
     Numeric max;
     boolean min_inclusive;
     boolean max_inclusive;
};
struct IntervalType {
     Numeric min;
     Numeric max;
     boolean min_inclusive;
     boolean max_inclusive;
     Numeric step;
};
enum ComplexDataType {ENUMERATION, RANGE, INTERVAL};
union AllowedValues switch (ComplexDataType) {
     case ENUMERATION:  EnumerationType allowed_enum;
     case INTERVAL:     IntervalType allowed_interval;
     case RANGE:        RangeType   allowed_range;
};
struct Parameter {
     string              name;
     CORBA::TCKind       type;
     AllowedValues allowed_values;
};
typedef sequence<Parameter> ParameterList;
struct OrganizationProperty {
     NVList properties;
};
enum DependencyType {
     NORMAL,
     REVERSE,
     NO_DEPENDENCY
};

struct DeviceProfile {
     string device_type;
```

```
        string manufacturer;
        string model;
        string version;
        NVList properties;
};
struct ServiceProfile {
        string id;
        string interface_type;
        NVList properties;
        SDOService service;
};
typedef sequence<ServiceProfile> ServiceProfileList;
struct ConfigurationSet {
        string id;
        string description;
        NVList configuration_data;
};
typedef sequence<ConfigurationSet> ConfigurationSetList;

/** ------- Exceptions -------*/
exception NotAvailable  exception_body;
exception InterfaceNotImplemented  exception_body;
exception InvalidParameter  exception_body;

/** ------- Interfaces -------*/
interface SDOSystemElement {
        OrganizationList get_organizations()
                raises (NotAvailable);
};

interface SDO : SDOSystemElement {
        UniqueIdentifier get_id()
                raises (NotAvailable);
        string get_SDO_type()
                raises (NotAvailable);
        DeviceProfile get_device_profile ()
                raises (NotAvailable);
        ServiceProfileList get_service_profiles ()
                raises (NotAvailable);
        ServiceProfile get_service_profile (
                in string id
                ) raises (InvalidParameter, NotAvailable);
        SDOService get_service (
                in string id
                ) raises (InvalidParameter, NotAvailable);
        Configuration get_configuration ()
                raises (InterfaceNotImplemented, NotAvailable);
        Monitoring get_monitoring ()
                raises (InterfaceNotImplemented, NotAvailable);
};
```

```
interface Configuration {
    void set_device_profile (in DeviceProfile dProfile)
        raises (InvalidParameter, NotAvailable);
    void add_service_profile (in ServiceProfile sProfile)
        raises (InvalidParameter, NotAvailable);
    void add_organization (in Organization organization)
        raises (InvalidParameter, NotAvailable);
    void remove_device_profile ()
        raises (NotAvailable);
    void remove_service_profile (in string id)
        raises (InvalidParameter, NotAvailable);
    void remove_organization (in Organization organization)
        raises (InvalidParameter, NotAvailable);
    ParameterList get_config_parameters ()
        raises (NotAvailable);
    any get_parameter_value (in string name)
        raises (InvalidParameter, NotAvailable);
    void set_config_parameter (
        in string name,
        in any value)
        raises (InvalidParameter, NotAvailable);
    ConfigurationSetList get_configuration_sets ()
        raises (NotAvailable);
    void add_configuration_set (in ConfigurationSet configuration_set)
        raises (InvalidParameter, NotAvailable);
    void remove_configuration_set (in string config_id)
        raises (InvalidParameter, NotAvailable);
    void activate_configuration_set (in string config_id)
        raises (InvalidParameter, NotAvailable);
};

interface Monitoring : CosNotifyComm::StructuredPushConsumer,
    CosNotifyComm::StructuredPushSupplier {
    any get_parameter_value (
        in string name
        ) raises (InvalidParameter, NotAvailable);
    ParameterList get_monitoring_parameters ()
        raises (NotAvailable);
    NVList get_current_monitoring_status ()
        raises (NotAvailable);
};

interface SDOService {};

interface Organization {
    void add_organization_property (
        in OrganizationProperty organization_property
        ) raises (InvalidParameter, NotAvailable);
    void remove_organization_property ()
        raises (NotAvailable);
    OrganizationProperty get_organization_property ()
```

```
            raises (NotAvailable);
        SDOList get_members ()
            raises (NotAvailable);
        void set_members (
            in SDOList sdos
            ) raises (InvalidParameter, NotAvailable);
        SDOSystemElement get_owner ()
            raises (NotAvailable);
        void set_owner (
            in SDOSystemElement sdo
            ) raises (InvalidParameter, NotAvailable);
        boolean get_direction()
            raises (NotAvailable);
        void set_direction (
            in boolean direction
            ) raises (NotAvailable);
    };
};
#endif //_SDO_PACKAGE_IDL_
```

References *A*

[1] SDO Whitepaper, http://www.omg.org/cgi-bin/doc?sdo/01-07-05

[2] "Information technology - Open Systems Interconnection - Remote Procedure Call (RPC)", February 2003

[3] OMG IDL Style Guide, ab/98-06-03

[4] Notification Service Specification, formal/02-08-04

[5] Event Service Specification, formal/01-03-01

A

Complete UML Diagram B

B.1 Complete UML Diagram of SDO resource data model

The complete UML diagram of SDO is as follows.

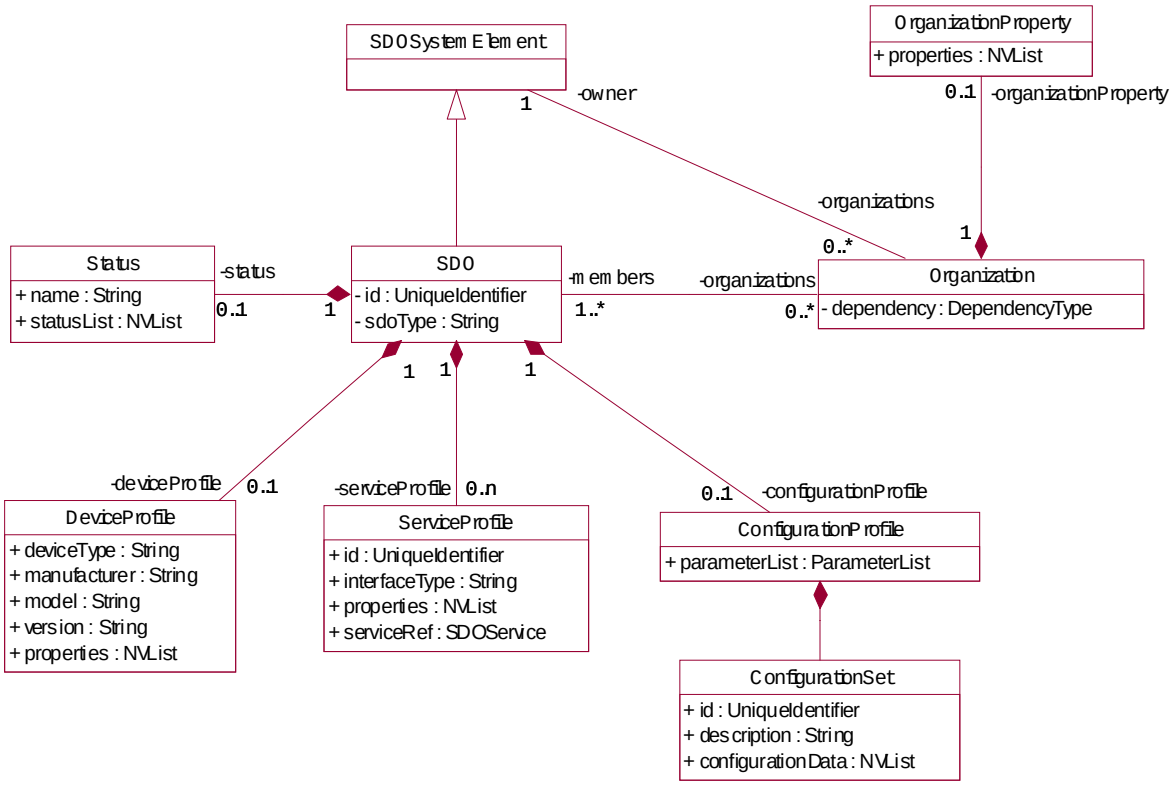


Figure B-1 Complete UML Diagram of the SDO resource data model

B.2 Mapping to ECHONET

B.2.1 What is ECHONET

(ref. http://www.echonet.gr.jp/english/index.htm)

The ECHONET Consortium was inaugurated in 1997 to shape an affluent society in the 21st century that was compatible with both the human being and the environment.

The ECHONET Consortium has since developed key software and hardware to support a home network that is committed to energy conservation, boosting security, enhancing home health care, etc. The network we develop uses power lines, radio frequency and infra-red to provide a low-cost implementation of data transmission without requiring additional wiring.

The consortium plans a validation test to evaluate the validity of the systems and software developed and to drive publicity in and outside Japan. It also expects to stage efforts to enhance security, strengthen interworking with the Internet, and develop new application middleware.

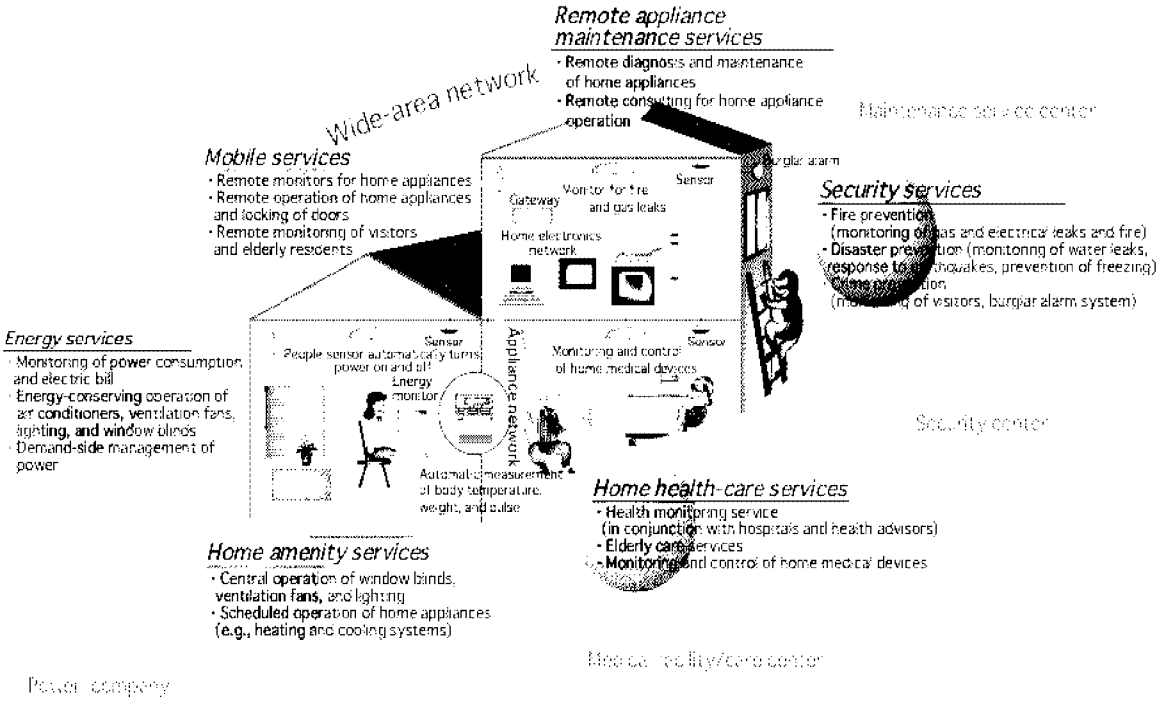


Figure B-2 Application Areas of ECHONET

B.2.2 ECHONET architecture

(ref. http://www.echonet.gr.jp/english/1_echo/index.htm)

- Designed for detached homes, collective housing, shops, and small office buildings.

- Open disclosure of APIs and protocol standards will promote applications development and result in an open system architecture that allows external expansion and new entries.
- The physical layer will be designed to accept other transmission media as well.
- Upper-level compatibility will be maintained by using HBS as a platform for development.

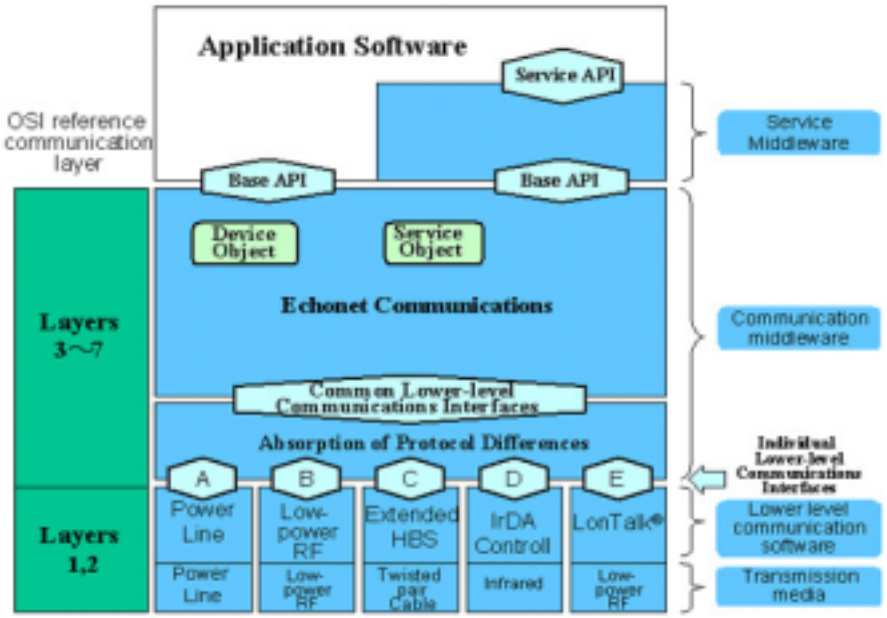


Figure B-3 ECHONET architecture

*1) API (Application Program Interface): An interface that makes it possible to call efficiently on functions provided by the network or OS. The presence of an API greatly facilitates programming efforts.

*2) HBS (Home Bus System): Japanese standard for home networks. Established in 1988 by the Electronic Industries Association of Japan.

*3) Lon Talk : Lon Talk is a registered trademark of Echelon Corporation in USA and other countries.

B.3 Mapping SDO to ECHONET

B.3.1 Resource data structure

In ECHONET, several standard objects have been specified to model home appliances. Typical ones are node profile object and device object. A node profile object describes an addressable device, and a device object describes common attributes of home

appliances as well as appliance specific attributes. As a device may have multiple functions and separated hardware (e.g., an air-conditioner may have indoor units, and an outdoor unit), a node object can contain multiple device objects.

In addition, a gateway object has been specified to mediate the communication between application programs outside of a home and ECHONET devices in a home. A gateway object provides interfaces of some devices that can be accessed from those applications.

Composite SDO structure of SDOs can be mapped to this structure and enable unified management of hardware device and software components. An organization represents composite devices and a gateway object.

B.3.2 Property mapping from ECHONET to SDO

In ECHONET, properties of the objects are defined in detail for each type of devices. Common properties of them are as follows,

Unique identifier data, Operating status, Fault status, Fault content, Version data, Manufacturer code, Place of business code, Product code, Serial number, Date of manufacture, SetM property map, GetM property map, Status change announcement property map, Set property map, Get property map, Installation location

For more detail, please refer to ECHONET specification (ref. http://www.echonet.gr.jp/english/8_kikaku/index.htm)

Properties defined in SDO resource data can represent these ECHONET properties as follows,

- SDO.id

  SDO.id is mapped to "Unique identifier data".

- DeviceProfile

  The properties specified in DeviceProfile are mapped to some properties of “Device object” and “Profile object” in ECHONET that contain "Version data", "Manufacturer code", "Place of business code", "Product code", "Serial number" and "Date of manufacture".

- ServiceProfile

  In ECHONET, functions of a device are described by property map holding an array of a code unique to each function. The properties specified in ServiceProfile classes are mapped to Property Maps("SetM property map", "GetM property map", "Status change announcement property map", "Set property map" and "Get property map") of “Device object” and “Node profile object” defined in ECHONET.

- Status

  ECHONET specifies properties representing status of an object. "Operating status", "Fault status" and "Fault content" are defined in the “Device object”. These properties are represented as named value sets in Status.statusList.

- Location

The "Installation location" property is specified in each Device Object in ECHONET to describe the location of each device. (e.g., outdoor unit, indoor unit) These properties are represented to, for example, a newly inherited class of SDOSystemElement.

B.3.3 Common interfaces

ECHONET specifies simple APIs named setProperty and getProperty. These APIs are used to handle properties of a device. SDO common interfaces proposed in this document are easily mapped to these APIs and special properties of ECHONET object corresponding to the SDO. Configuration interface is used to wrap “setProperty” API. “getProperty” is wrapped by monitoring interface or other operations in SDO defined to set SDO profiles.