



# Platform Independent Model and Platform Specific Model for Super Distributed Object

*Version 1.1*

---

OMG Document Number: formal/2008-10-01  
Standard document URL: <http://www.omg.org/spec/SDO/1.1/PDF>  
Associated File(s)\*: <http://www.omg.org/spec/SDO/20080301>  
<http://www.omg.org/spec/SDO/20080201>

---

Source document: PIM and PSM for SDO, Beta 1 (dtc/2008-02-25 and 2008-02-26)

\* Original files: IDL (dtc/2008-03-05) and XMI (dtc/2008-02-27)

Copyright © 2003, Fraunhofer FOKUS  
Copyright © 2003, Hitachi Ltd.  
Copyright © 2008, Object Management Group

## USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

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

## LICENSES

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

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

## PATENTS

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

## GENERAL USE RESTRICTIONS

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

## DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE.

IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

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

## RESTRICTED RIGHTS LEGEND

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

## TRADEMARKS

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

## COMPLIANCE

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

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the

software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

## **OMG's Issue Reporting Procedure**

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



# Table of Contents

Preface .....	iii
1 Scope .....	1
1.1 Objectives .....	1
2 Compliance .....	1
3 Normative References .....	2
4 Terms and Definitions .....	2
5 Symbols .....	2
6 Additional Information .....	2
6.1 Acknowledgements .....	2
7 Platform Independent Model .....	3
7.1 Overview .....	3
7.2 Overview of Platform Independent Model (PIM) for Super Distributed Objects (SDO) .....	3
7.3 Resource Data Model .....	3
7.3.1 Overview of Resource Data Model .....	3
7.3.2 Data Structures Used by Resource Data Model .....	4
7.3.3 SDOSystemElement .....	8
7.3.4 SDO .....	9
7.3.5 Organization .....	10
7.3.6 OrganizationProperty .....	12
7.3.7 DeviceProfile .....	12
7.3.8 ServiceProfile .....	13
7.3.9 Status .....	14
7.3.10 ConfigurationProfile.....	14
7.3.11 Examples of resource data model .....	15
7.4 Interfaces .....	17
7.4.1 Overview of Interfaces .....	17
7.4.2 Data Structures used by Interfaces .....	18
7.4.3 SDOSystemElement Interface .....	19
7.4.4 SDO Interface .....	19
7.4.5 Configuration Interface .....	24
7.4.6 SDOService Interface .....	34

7.4.7 Monitoring Interface .....	34
7.4.8 Organization Interface .....	47
<b>8 Platform Specific Model: Mapping to CORBA IDL .....</b>	<b>55</b>
8.1 Overview .....	55
8.2 SDO Module .....	55
8.3 Data Types used in CORBA PSM.....	55
8.4 Exceptions .....	57
8.5 Interfaces .....	57
8.5.1 SDOSystemElement Interface .....	58
8.5.2 SDO Interface .....	58
8.5.3 Configuration Interface .....	58
8.5.4 SDOService .....	59
8.5.5 Monitoring Interface .....	59
8.5.6 Organization Interface .....	60
<b>9 OMG IDL .....</b>	<b>63</b>
<b>Annex A - References .....</b>	<b>69</b>
<b>Annex B - Complete UML Diagram .....</b>	<b>71</b>
<b>Index .....</b>	<b>75</b>



# Preface

## About the Object Management Group

### OMG

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

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

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

## OMG Specifications

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

[http://www.omg.org/technology/documents/spec\\_catalog.htm](http://www.omg.org/technology/documents/spec_catalog.htm)

Specifications within the Catalog are organized by the following categories:

### OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications.

### OMG Middleware Specifications

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

### Platform Specific Model and Interface Specifications

- CORBA services

- CORBA facilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications.

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. (as of January 16, 2006) at:

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

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

## Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to <http://www.omg.org/technology/agreement.htm>.

# 1 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).

## 1.1 Objectives

A Super Distributed Object (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 to 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. This specification does not address access control or security aspects.

# 2 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 7.3.3, “SDOSystemElement,” on page 8 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.

### **3 Normative References**

There are no references that pertain to this specification.

### **4 Terms and Definitions**

There are no terms and definitions that pertain to this specification.

### **5 Symbols**

There are no symbols that pertain to this specification.

### **6 Additional Information**

#### **6.1 Acknowledgements**

The following companies submitted and/or supported this specification:

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

# 7 Platform Independent Model

## 7.1 Overview

This section specifies the PIM from information and computational viewpoints of SDOs. Section 7.3, “Resource Data Model,” on page 3 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 7.4, “Interfaces,” on page 17 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.

## 7.2 Overview of Platform Independent Model (PIM) for Super Distributed Objects (SDO)

An SDO represents a hardware device or software component, and provides a set of interfaces through which other SDOs or applications access it. 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.

## 7.3 Resource Data Model

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

### 7.3.1 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 7.3.7, “DeviceProfile,” on page 12).
  - Service profile, which defines a set of software specific properties (see Section 7.3.8, “ServiceProfile,” on page 13).
  - Configuration profile, which defines a set of properties to configure SDOs (see Section 7.3.10, “ConfigurationProfile,” on page 14).

- Organization, which defines a relationship between/among the objects running in SDO system (see Section 7.3.5, “Organization,” on page 10).
- Status, which indicates the current status of SDOs (see Section 7.3.9, “Status,” on page 14).

Figure 7.1 shows a simplified UML notation of the resource data. The complete diagram is depicted in Annex B.

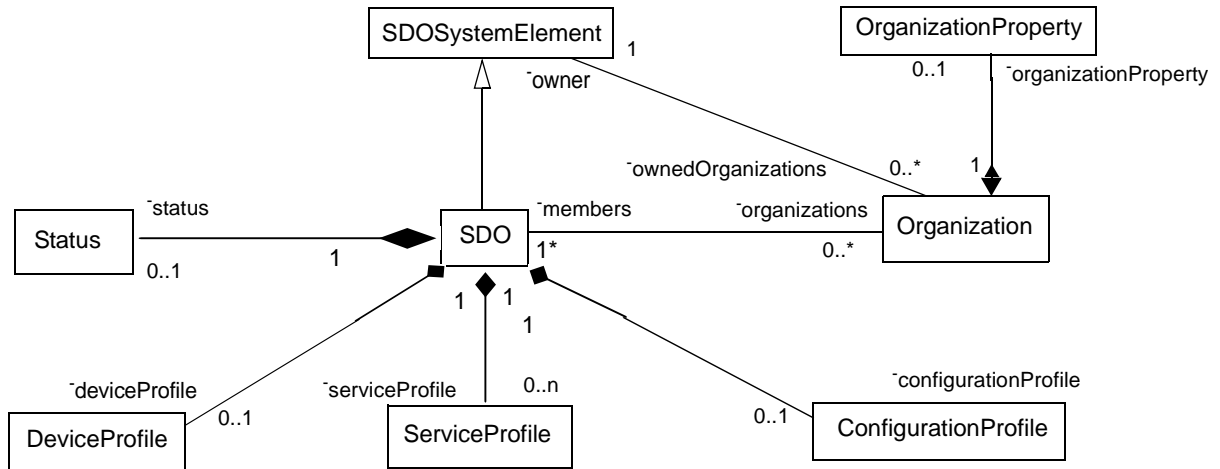
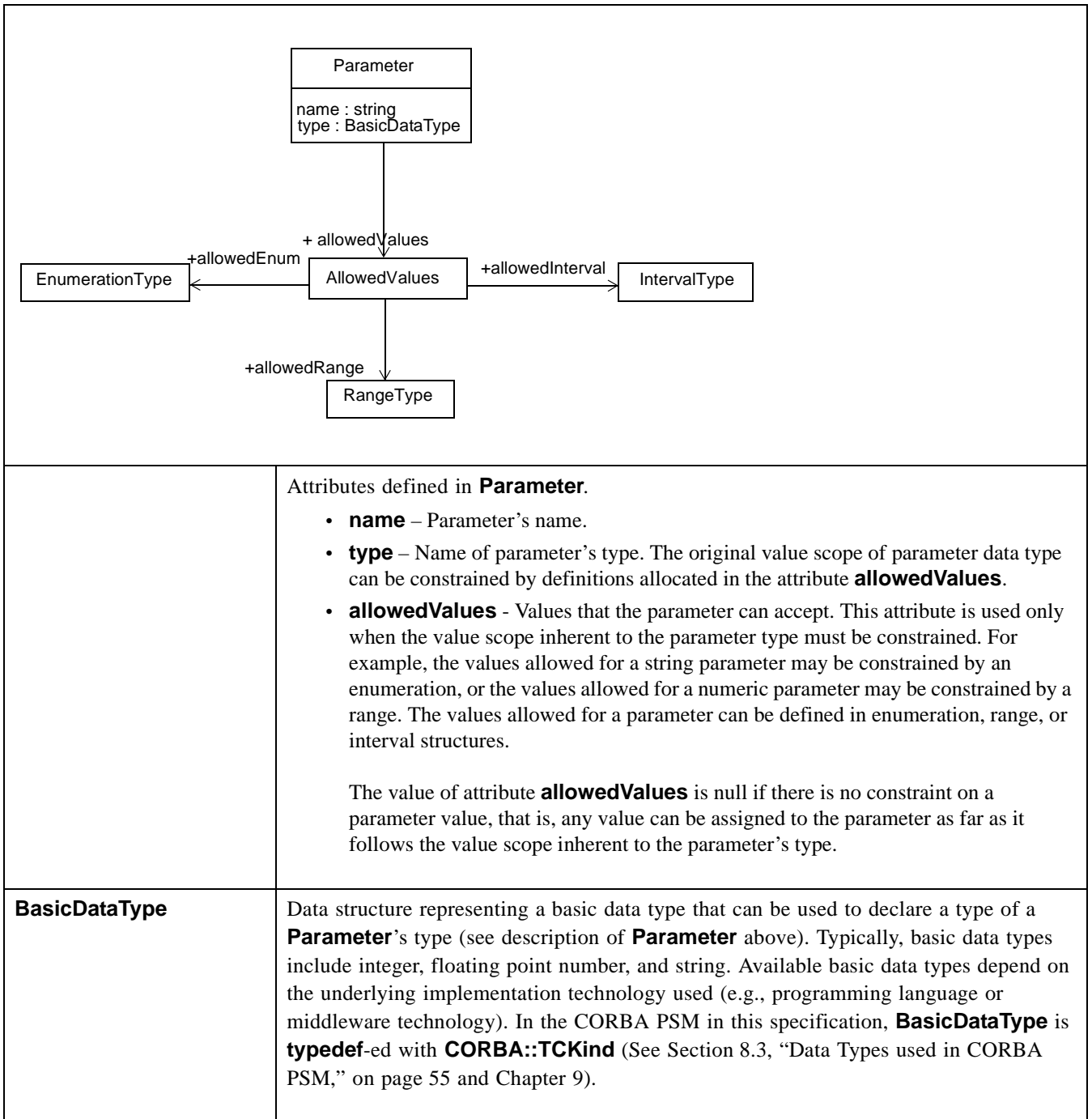


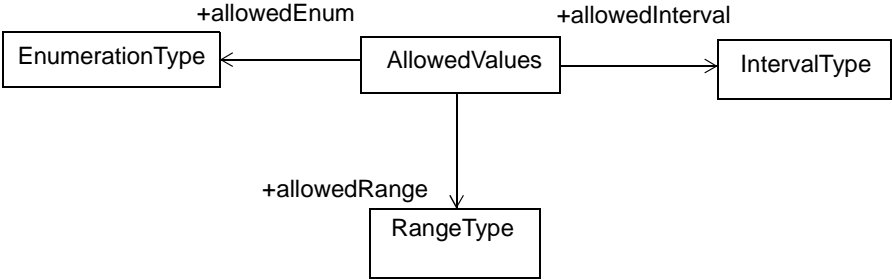
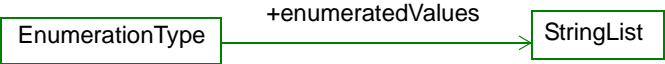
Figure 7.1 - SDO resource data model

### 7.3.2 Data Structures Used by Resource Data Model

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

Name	Description
<b>StringList</b>	A list of strings.
<b>UniquelIdentifier</b>	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. See the Note below this table.
<b>NameValue</b>	A pair of a name and its value.  <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 10px auto;"> <pre> NameValue + name : string + value : any           </pre> </div>
<b>NVList</b>	A list of <b>NameValue</b> pairs.
<b>Parameter</b>	Data structure to define a variable (parameter) independently of implementation technologies. The <b>Parameter</b> structure defines the name and type of a variable.



<b>AllowedValues</b>	
	 <pre> classDiagram     class AllowedValues     class EnumerationType     class RangeType     class IntervalType     AllowedValues &lt; -- EnumerationType     AllowedValues &lt; -- RangeType     AllowedValues &lt; -- IntervalType     </pre>
	<p>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 that is defined with <b>EnumerationType</b>, <b>RangeType</b>, or <b>IntervalType</b> structures, respectively (see below).</p>
<b>ComplexDataType</b>	<div data-bbox="507 898 770 1066" style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <pre> &lt;&lt;Enumeration&gt;&gt; ComplexDataType ENUMERATION RANGE INTERVAL </pre> </div> <p><b>ComplexDataType</b> enumerates three types of structures, each of which can be used in the attribute <b>allowedValues</b> to define a value scope of constraint. The structures that can be used to specify enumeration, range, or interval are defined below.</p>
<b>EnumerationType</b>	 <pre> classDiagram     class EnumerationType     class StringList     EnumerationType --&gt; StringList : +enumeratedValues     </pre> <p>The enumeration of values that can be assigned to the attribute AllowedValues of <b>Parameter</b>. The enumerated values are always of type string.</p> <ul style="list-style-type: none"> <li>• <b>enumerated_values</b> – a list of string values that a <b>Parameter</b> can contain. Example: {"red," "blue," "white"}.</li> </ul>
<b>NumericType</b>	<div data-bbox="611 1514 951 1682" style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <pre> N u m e r i c T y p e S H O R T _ T Y P E L O N G _ T Y P E F L O A T _ T Y P E D O U B L E _ T Y P E </pre> </div> <p>This enumeration is used by structures <b>RangeType</b> and <b>IntervalType</b>. It defines numeric types used to specify the bounds of intervals or ranges, and the step between interval values.</p>



<p><b>RangeType</b></p>	<div data-bbox="502 346 1145 520" data-label="Diagram"> <pre> classDiagram     class RangeType {         minInclusive : boolean         maxInclusive : boolean     }     class Numeric {         abortValue : abort         longValue : long         floatValue : float         doubleValue : double     }     RangeType --&gt; Numeric : +min     RangeType --&gt; Numeric : +max     </pre> </div> <p>Data structure representing a range of values that can be assigned to the attribute AllowedValues of <b>Parameter</b>.</p> <ul style="list-style-type: none"> <li>• <b>min</b> – The lower bound of the range.</li> <li>• <b>max</b> – The upper bound of the range.</li> <li>• <b>minInclusive</b> – A boolean value showing if the lower bound value is included in the range.</li> <li>• <b>maxInclusive</b> – A boolean value showing if the upper bound value is included in the range.</li> </ul> <p>Example: The range ((int) 0, (int) 20, false, true) defines values {1,2,...20}.</p>
<p><b>IntervalType</b></p>	<div data-bbox="502 961 1161 1136" data-label="Diagram"> <pre> classDiagram     class IntervalType {         minInclusive : boolean         maxInclusive : boolean     }     class Numeric {         shortValue : short         longValue : long         floatValue : float         doubleValue : double     }     IntervalType --&gt; Numeric : +min     IntervalType --&gt; Numeric : +max     IntervalType --&gt; Numeric : +step     </pre> </div> <p>Data structure representing an interval of values that can be assigned to the attribute AllowedValues of <b>Parameter</b> defined as interval.</p> <ul style="list-style-type: none"> <li>• <b>min</b> – The lower bound of the interval.</li> <li>• <b>max</b> – The upper bound of the interval.</li> <li>• <b>minInclusive</b> – A boolean value showing if the lower bound value is included in the interval.</li> <li>• <b>maxInclusive</b> – A boolean value showing if the upper bound value is included in the interval.</li> <li>• <b>step</b> – The step between the values in the interval.</li> </ul> <p>Example: The interval ((int) 0, (int) 20, false, true, 5) defines values {5,10,15,20}.</p>

<b>ParameterList</b>	A list of <b>Parameter</b> structures.
<b>DependencyType</b>	<p>Data type used to specify relation between elements in an organization. The value indicates if one side of an <b>Organization</b> depends on the other side. If the <b>Organization</b> represents dependency relationship, it also indicates which side depends on which side. Enumeration <b>DependencyType</b> includes three possible forms of dependency.</p> <div style="border: 1px solid green; padding: 5px; margin: 10px auto; width: fit-content;"> <p style="text-align: center;">&lt;Enumeration&gt; DependencyType</p> <hr style="border: 0.5px solid green;"/> <p>+NORMAL +REVERSE +NO_DEPENDENCY</p> </div>

**Note** – 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.

### 7.3.3 SDOSystemElement

SDOSystemElement
- organizations : OrganizationList

**SDOSystemElement** is the base class of the classes that represent SDO system 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 7.3.4, "SDO," on page 9. And, there are system elements that are running on the SDO environment but are not SDOs as specified in this document. Examples of those elements that are not SDO include human users and locations. It is left to future specifications to define those elements as additional subclasses of **SDOSystemElement**.

## Attributes

Attribute	Type	Description
<b>ownedOrganizations</b>	<b>OrganizationList</b>	A list of <b>Organization</b> objects that <b>SDOSystemElement</b> has.

### 7.3.4 SDO

SDO
- id : UniquelIdentifier - sdoType : String - deviceProfile : DeviceProfile - serviceProfiles : ServiceProfileList - configurationProfile : ConfigurationProfile - organizations : OrganizationList - status : Status

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

SDOs are characterized by properties. Due to the nature of the SDO different SDOs can have different properties. Which properties are monitorable and which are configurable is implementation dependent. Properties that can be monitored are provided through the monitoring interface. Properties that can be configured are provided through the configuration. A property can be both configurable and monitorable. Where and how these properties are stored is implementation dependent.

Services provided by the SDO can have their own properties that define the behavior of the service. These properties can also be provided to be configured. Some sort of mechanism must be provided to indicate the difference between service and the SDO properties.

## Attributes

Attribute	Type	Description
<b>id</b>	<b>UniquelIdentifier</b>	Unique identifier for an SDO.
<b>sdoType</b>	<b>String</b>	Textual description of the SDO. <b>sdoType</b> contains short description of the SDO's functionality.
<b>deviceProfile</b>	<b>DeviceProfile</b>	A device profile that an SDO has. See Section 7.3.7, "DeviceProfile," on page 12 for more details about device profile.
<b>serviceProfiles</b>	<b>ServiceProfileList</b>	A list of service profiles that an SDO has. See Section 7.3.8, "ServiceProfile," on page 13 for more details about service profile.
<b>configurationProfile</b>	<b>ConfigurationProfile</b>	A configuration profile that an SDO has. See Section 7.3.10, "ConfigurationProfile," on page 14 for more details about configuration profile.
<b>status</b>	<b>Status</b>	A status information of an SDO. See Section 7.3.9, "Status," on page 14 for more details about status.
<b>organizations</b>	<b>OrganizationList</b>	A list of references to the list of <b>Organization</b> objects that an SDO keeps with other SDOs or objects of type <b>SDOSystemElement</b> . See Section 7.3.5, "Organization," on page 10 for more details about class <b>Organization</b> .

### 7.3.5 Organization

Organization
<ul style="list-style-type: none"> <li>- id : UniquelIdentifier</li> <li>- members : SDOList</li> <li>- owner : SDOSystemElement</li> <li>- dependency : DependencyType</li> <li>- property : OrganizationProperty</li> </ul>

**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 7.3.6, "OrganizationProperty," on page 12).

## Attributes

Attribute	Type	Description
<b>id</b>	<b>UniquelIdentifier</b>	The identifier of the <b>Organization</b> .
<b>members</b>	<b>SDOList</b>	A list of reference to SDOs that are the members associated with the <b>Organization</b> .
<b>owner</b>	<b>SDOSystemElement</b>	The owner of the <b>Organization</b> . It can be an SDO or another subclass of <b>SDOSystemElement</b> (See the text under this table for more detail).
<b>dependency</b>	<b>DependencyType</b>	This attribute specifies the dependency relation between the owner and members of the organization. Further details are discussed in text under this table.
<b>property</b>	<b>OrganizationProperty</b>	Property of the <b>Organization</b> . <b>OrganizationProperty</b> is described in Section 7.3.6, “OrganizationProperty,” on page 12.

**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 **OWN** value (see description of DependencyType on previous pages).
2. Reversely hierarchical organization, which indicates **members** supervise **owner**. In this case, **DependencyType** should hold **OWNED** value (see description of DependencyType on previous pages).
3. Flat organization, which indicates no dependency exists. In this case, **DependencyType** should hold **NO\_DEPENDENCY** value (see description of DependencyType on previous pages).

Both an SDO and another subclass of **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 subclass of **SDOSystemElement**, which is not an 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 the same place (e.g., a room) have equal relationships among them to communicate with each other.

### 7.3.6 OrganizationProperty

OrganizationProperty
+ properties : NVList

**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 <b>Organization</b> . The property values contained in this attribute are implementation dependent. Examples include the identifier of an <b>Organization</b> and the time when an <b>Organization</b> is established.

### 7.3.7 DeviceProfile

DeviceProfile
+ deviceType : String + manufacturer : String + model : String + version : String +properties : NVList

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

## Attributes

Attribute	Type	Description
<b>deviceType</b>	<b>String</b>	General type name of a device. This attribute describes general kind of devices to categorize them and specify fundamental capability of the device.
<b>manufacturer</b>	<b>String</b>	Identifier for the manufacturer of the device.
<b>model</b>	<b>String</b>	Model name of the device.
<b>version</b>	<b>String</b>	Version number of the device.
<b>properties</b>	<b>NVList</b>	Device specific properties in addition to the above four. These properties are statically assigned and immutable at runtime. They are not configured and monitored through the configuration and monitoring interfaces, respectively. The property values contained in this attribute are implementation dependent.

### 7.3.8 ServiceProfile

ServiceProfile
+ id : UniquelIdentifier + interfaceType : String + properties : NVList + serviceRef : SDOService

**ServiceProfile** defines a set of properties of the **service** provided by a device or software component that an SDO represents. The **service** is implemented by another object that **service** refers to (see also Section 7.4.6, “SDOService Interface,” on page 34). An SDO maintains zero or more **ServiceProfile** objects, each of which maintains a reference to the object implementing the service.

For example, an air conditioner SDO provides a service to stabilize room temperature. This service may have a control interface (referred by Service) that defines the operations to set a room temperature, set an operation mode (e.g., heating, cooling, or dehumidifying) and power on/off.

## Attributes

Attribute	Type	Description
<b>id</b>	<b>UniquelIdentifier</b>	Identifier for a service (or function) that an SDO provides. An SDO can provide one or more services, and id is used to distinguish different services. An SDO provides zero or more services, and 'id' is used to distinguish different services.
<b>interfaceType</b>	<b>String</b>	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 CORBA PSM, this attribute contains the repository ID of the IDL interface through which an SDO's service (function) is provided.
<b>properties</b>	<b>NVList</b>	The list of properties describing the SDOService itself (ontology description). The property values contained in this attribute are specific to each SDOService and implementation dependent.
<b>serviceRef</b>	<b>SDOService</b>	Reference to the object that provides the service (function) represented by this profile.

### 7.3.9 Status

Status
+statusList : NVList

**Status** contains the current status of an SDO. It contains a set of status described by a pair of name and value for each status value. 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
<b>statusList</b>	<b>NVList</b>	A list containing status information.

### 7.3.10 ConfigurationProfile

ConfigurationProfile
+ parameterList : ParameterList
+ configurationSetList : ConfigurationSetList



**ConfigurationProfile** contains a set of properties to configure an SDO.

### Attributes

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

- Data type definition: **ConfigurationSet**

ConfigurationSet
+ id : UniqueIdentifier + description : String + configurationData : NVList

Parameter defines the data type of a variable.

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

### 7.3.11 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	<i>Temperature Sensor</i>	
manufacturer	<i>Thermo Inc.</i>	
model	<i>TH310</i>	
version	<i>1</i>	
properties	name	value
	<i>unit</i>	<i>Celsius</i>
	<i>rangeMin</i>	<i>-50</i>
	<i>rangeMax</i>	<i>50</i>

SDO	
attribute	value
id	<i>abc_12345</i>
sdoType	<i>EN_TemperatureSensor</i>

Status		
attribute	value	
name	<i>deviceStatus</i>	
statusList	name	value
	<i>thermoValue</i>	<i>30</i>

**Figure 7.2 - 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.

DeviceProfile	
attribute	value
deviceType	<i>Airconditioner</i>
manufacturer	<i>EPT_800</i>
model	<i>Electrolux</i>
version	<i>1</i>
properties	

SDO	
attribute	value
sdoid	<i>xyy_113</i>
sdoType	<i>EN_AirConditioner</i>

Organization	
attribute	value
DependencyType	<i>NO_DEPENDENCY</i>

Organization	
attribute	value
DependencyType	<i>NORMAL</i>

ServiceProfile		
attribute	value	
id	<i>1</i>	
interaceType	<i>com::ept_800::electolux::SetTemperature</i>	
properties	name	value
	<i>rangeMin</i>	<i>18</i>
serviceRef		

**Figure 7.3 - Example: Airconditioner SDO**

Since the Airconditioner SDO provides two different software services (functions), it provides 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 connect with 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 subclass of **SDOSystemElement** 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 “**OWN**” in its **DependencyType** attribute and references to the thermometer SDO (member) and airconditioner SDO (owner).

## 7.4 Interfaces

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

### 7.4.1 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. Which properties are monitorable is implementation dependent. **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. Which properties are configurable is implementation dependent.
- **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.

## 7.4.2 Data Structures used by Interfaces

### 7.4.2.1 SDOException

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

#### Attributes

Attribute	Type	Description
<b>type</b>	<b>String</b>	Name of a raised exception.
<b>description</b>	<b>String</b>	Descriptive information of a raised exception.

#### 7.4.2.1.1 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 the target SDO is reachable but cannot respond. 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 if at least one argument specified in an operation call is invalid.

##### **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.

##### **InternalError**

SDO is reachable and can respond but cannot execute the operation completely due to some internal error.

### 7.4.3 SDOSystemElement Interface

The **SDOSystemElement** interface is used to manage the subclass of **SDOSystemElement** itself.

(1) + *getOwnedOrganizations ( ) : OrganizationList*

**SDOSystemElement** can be the owner of zero or more organizations. If the **SDOSystemElement** owns one or more **Organizations**, this operation returns the list of **Organizations** that the **SDOSystemElement** owns. If it does not own any **Organization**, it returns empty list.

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

#### Exceptions

This operation throws **SDOException** with one of the following types:

- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

### 7.4.4 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.

SDO
+getSDOID() : UniqueIdentifier +getSDOType() : String +getStatus(name : String) : any +getStatusList() : NVList +getDeviceProfile() : DeviceProfile +getServiceProfiles() : ServiceProfileList +getServiceProfile(id : UniqueIdentifier) : ServiceProfile +getSDOService(id : UniqueIdentifier) : SDOService +getConfiguration() : Configuration +getMonitoring() : Monitoring +getOrganizations() : OrganizationList

(1) *getSDOID ( ) : String*

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

Parameter	Type	Description
<return>	<b>UniquelIdentifier</b>	<b>id</b> of the SDO defined in the resource data model.

### Exceptions

This operation throws **SDOException** with one of the following types:

- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(2) + *getSDOType ( ) : String*

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

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

### Exceptions

This operation throws **SDOException** with one of the following types:

- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(3) + *getStatus ( name : String ) : any*

This operation returns the value of the specified status parameter.

Parameter	Type	Description
<b>Name</b>	<b>String</b>	One of the parameters defining the 'status' of an SDO.
<return>	<b>Any</b>	The value of the specified status parameter.

### Exceptions

This operation throws **SDOException** with one of the following types:

- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InvalidParameter** - if the parameter defined by 'name' is null or does not exist.

- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(4) + *getStatusList( ) : NVList*

This operation returns an NVlist describing the status of an SDO.

Parameter	Type	Description
<return>	<b>NVList</b>	The actual status of an SDO.

### Exceptions

This operation throws **SDOException** with one of the following types:

- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(5) + *getDeviceProfile( ) : DeviceProfile*

This operation returns the **DeviceProfile** of the SDO. If the SDO does not represent any hardware device, then a **DeviceProfile** with empty values are returned.

Parameter	Type	Description
<return>	<b>DeviceProfile</b>	The <b>DeviceProfile</b> of the SDO.

### Exceptions

This operation throws **SDOException** with one of the following types:

- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(6) + *getServiceProfiles( ) : ServiceProfileList*

This operation returns a list of **ServiceProfiles** that the SDO has. If the SDO does not provide any service, then an empty list is returned.

Parameter	Type	Description
<return>	<b>ServiceProfilesList</b>	List of <b>ServiceProfiles</b> of all the services the SDO is providing.

### Exceptions

This operation throws **SDOException** with one of the following types:

- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(7) + *getServiceProfile ( id : UniqueIdentifier ) : ServiceProfile*

This operation returns the **ServiceProfile** that is specified by the argument “**id**.”

Parameter	Type	Description
<b>id</b>	<b>UniqueIdentifier</b>	The identifier referring to one of the <b>ServiceProfiles</b> .
<return>	<b>ServiceProfile</b>	The profile of the specified service.

### Exceptions

This operation throws **SDOException** with one of the following types:

- **InvalidParameter** - if the **ServiceProfile** that is specified by the argument 'id' does not exist or if 'id' is 'null.'
- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(8) + *getSDOService ( id : UniqueIdentifier ) : SDOService*

This operation returns an object implementing an SDO’s service that is identified by the identifier specified as an argument. Different services provided by an SDO are distinguished with different identifiers. See Section 7.3.8, “ServiceProfile,” on page 13 for more details.

Parameter	Type	Description
<b>id</b>	<b>UniqueIdentifier</b>	The identifier of the requested service.
<return>	<b>SDOService</b>	The object implementing the requested service.

### Exceptions

This operation throws **SDOException** with one of the following types:

- **InvalidParameter** - if argument “**id**” is null, or if the **ServiceProfile** that is specified by argument “**id**” does not exist.
- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.



(9) + *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 7.4.5, “Configuration Interface,” on page 24 for more details about the **Configuration** interface.

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

**Exceptions**

This operation throws **SDOException** with one of the following types:

- **InterfaceNotImplemented** - if the target SDO has no Configuration interface.
- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(10) + *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 7.4.7, “Monitoring Interface,” on page 34 for more details about the **Monitoring** interface.

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

**Exceptions**

This operation throws **SDOException** with one of the following types:

- **InterfaceNotImplemented** - if the target SDO has no **Monitoring** interface.
- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(11) + *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. An empty list is returned if the SDO does not belong to any Organizations.

Parameter	Type	Description
<return>	<b>OrganizationList</b>	The list of <b>Organizations</b> that the SDO belong to.

### Exceptions

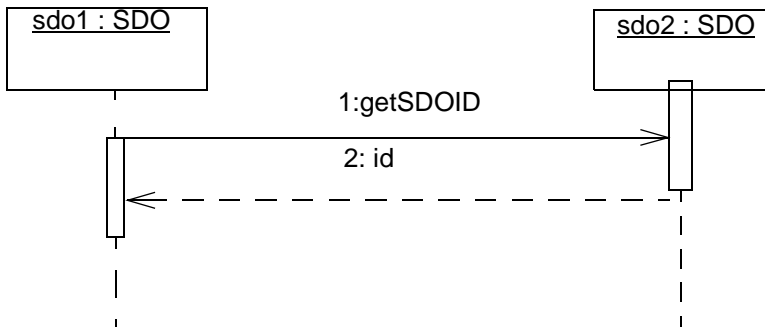
This operation throws **SDOException** with one of the following types:

- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

#### 7.4.4.1 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 7.4.

In Figure 7.4, 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 7.4.7, “Monitoring Interface,” on page 34.



**Figure 7.4 - 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.

### 7.4.5 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**. This specification does not address access control or security aspects. Access to operations that modify or remove profiles should be controlled depending upon the application.

Different configurations can be stored for simple and quick activation. Different predefined configurations are stored as different **ConfigurationSets** or configuration profile. A **ConfigurationSet** stores the value of all properties assigned for the particular configuration along with its unique id and description to identify and describe the configuration respectively.

Operations in the configuration interface help manage these **ConfigurationSets**.

**getConfigurationSets ()** - This operation returns a list of all **ConfigurationSets** of the SDO. If no predefined **ConfigurationSets** exist, then empty list is returned.

**addConfigurationSet(..)** - This operation adds a new **ConfigurationSet**. The given **ConfigurationSet** contains the property names and their desired configuration values. This **ConfigurationSet** may not provide desired value for all configurable properties. If so, then the values of the configuration properties that are not given are the current values of these properties.

**removeConfigurationSet(..)** - This operation removes the given **ConfigurationSet**. Once the configuration set is removed, this configuration set cannot be activated.

**activateConfigurationSet(..)** - This operation activates the specified stored **ConfigurationSets**. This means that the configuration properties of the SDO are changed as the values of these properties specified in the stored **ConfigurationSet**. In other words, values of the specified **ConfigurationSet** are now copied to the active configuration.

**getConfigurationSet(..)** - This operation gets the **ConfigurationSet** specified by the parameter.

**getActiveConfigurationSet()** - This operation gets the current active **ConfigurationSet** if any.

Configuration
+setDeviceProfile(dProfile : DeviceProfile) : Boolean
+addServiceProfile(sProfile : ServiceProfile) : Boolean
+addOrganization(organization : Organization) : Boolean
+removeServiceProfile(id : UniqueIdentifier) : Boolean
+removeOrganization(organizationID : UniqueIdentifier) : Boolean
+getConfigurationParameters() : ParameterList
+getConfigurationParameterValues() : NVList
+getConfigurationParameterValue(name : String) : any
+setConfigurationParameter(name : String, value : any) : Boolean
+getConfigurationSets() : ConfigurationSetList
+getConfigurationSet(configurationSetID : UniqueIdentifier) : ConfigurationSet
+getActiveConfigurationSet() : ConfigurationSet
+addConfigurationSet(configurationSet : ConfigurationSet) : Boolean
+setConfigurationSetValues(configurationSet : ConfigurationSet) : Boolean
+removeConfigurationSet(configurationSetID : UniqueIdentifier) : Boolean
+activateConfigurationSet(configurationSetID : UniqueIdentifier) : Boolean

### 7.4.5.1 Operations

(1) + *setDeviceProfile* ( *dProfile*: *DeviceProfile* ) : *Boolean*

This operation sets the **DeviceProfile** of an SDO. If the SDO does not have **DeviceProfile**, the operation will create a new **DeviceProfile**, otherwise it will replace the existing **DeviceProfile**.

Parameter	Type	Description
<b>dProfile</b>	<b>DeviceProfile</b>	The device profile that is to be assigned to this SDO.
<return>	Boolean	If the operation was successfully completed.

#### Exceptions

This operation throws **SDOException** with one of the following exception types:

- **SDOExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InvalidParameter** - if argument “**dProfile**” is null.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(2) + *addServiceProfile* ( *sProfile* : *ServiceProfile* ) : *Boolean*

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
<b>sProfile</b>	<b>ServiceProfile</b>	<b>ServiceProfile</b> to be added.
<return>	<b>Boolean</b>	If the operation was successfully completed.

#### Exceptions

This operation throws **SDOException** with one of the following exception types:

- **SDOExists** - if the target SDO does not exist.
- **InvalidParameter** - if argument “**sProfile**” is null.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(3) + *addOrganization (organization : Organization) : Boolean*

This operation adds reference of an **Organization** object.

Parameter	Type	Description
<b>organization</b>	<b>Organization</b>	Organization to be added.
<return>	<b>Boolean</b>	If the operation was successfully completed.

**Exceptions**

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.
- **InvalidParameter** - if argument “organization” is null.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(4) + *removeServiceProfile (id : UniqueIdentifier) : Boolean*

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
<b>id</b>	<b>UniqueIdentifier</b>	<b>serviceID</b> of a <b>ServiceProfile</b> to be removed.
<return>	Boolean	If the operation was successfully completed.

**Exceptions**

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.
- **InvalidParameter** - if argument “sProfile” is null, or if the object that is specified by argument “sProfile” does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(5) + *remove Organization (organizationID: UniqueIdentifier) : Boolean*

This operation removes the reference of an Organization object.

Parameter	Type	Description
<b>organizationID</b>	<b>UniquelIdentifier</b>	<b>UniquelIdentifier</b> of the organization to be removed.
<return>	<b>Boolean</b>	If the operation was successfully completed.

### Exceptions

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.
- **InvalidParameter** - if argument “**organizationID**” is null, or the object which is specified by argument “**organizationID**” does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

### (6) *getConfigurationParameters () : ParameterList*

This operation returns a list of Parameters. An empty list is returned if the SDO does not have any configurable parameter.

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

### Exceptions

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

### (7) *+ getConfigurationParameterValues(): NVList*

This operation returns all configuration parameters and their values.

Parameter	Type	Description
<return>	<b>NVList</b>	List of all configuration parameters and their values.

### Exceptions

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.

- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(8) + *getConfigurationParameterValue ( name : String ) : any*

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

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

### Exceptions

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.
- **InvalidParameter** - 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** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(9) + *setConfigurationParameter(name:String, value:any):Boolean*

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
<b>name</b>	<b>String</b>	The name of parameter to be modified.
<b>value</b>	<b>any</b>	New value of the specified parameter.
<return>	Boolean	If the operation was successfully completed.

### Exceptions

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.
- **InvalidParameter** - if arguments (“name” and/or “value”) is null, or if the parameter that is specified by the argument “name” does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(10) + *getConfigurationSets () : ConfigurationSetList*

This operation returns a list of **ConfigurationSets** that the **ConfigurationProfile** has. An empty list is returned if the SDO does not have any **ConfigurationSets**.

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

**Exceptions**

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(11) + *getConfigurationSet( configurationSetID: UniqueIdentifier ) : ConfigurationSet*

This operation returns the **ConfigurationSet** specified by the parameter **configurationSetID**.

Parameter	Type	Description
<b>configurationSetID</b>	<b>UniqueIdentifier</b>	Identifier of <b>ConfigurationSet</b> requested.
<return>	<b>ConfigurationSet</b>	The configuration set specified by the parameter <b>configurationSetID</b> .

**Exceptions**

This operation throws **SDOException** with one of the following exception types:

- **InvalidParameter** - if the parameter '**configurationSetID**' is null or if there are no **ConfigurationSets** stored with such id.
- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(12) + *getActiveConfigurationSet(): ConfigurationSet*

This operation returns the current active **ConfigurationSet** of an SDO (i.e., if the current configuration of the SDO was set using predefined configuration set). **ConfigurationSet** cannot be considered active if the:

- current configuration of the SDO was not set using any predefined **ConfigurationSet**, or
- configuration of the SDO was changed after it has been active, or



- **ConfigurationSet** that was used to configure the SDO was modified.

Empty **ConfigurationSet** is returned in these cases.

Parameter	Type	Description
<return>	<b>ConfigurationSet</b>	The active <b>ConfigurationSet</b> .

### Exceptions

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(13) + *addConfigurationSet ( configurationSet : ConfigurationSet ) : Boolean*

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

Parameter	Type	Description
<b>configurationSet</b>	<b>ConfigurationSet</b>	The <b>ConfigurationSet</b> that is added.
<return>	Boolean	If the operation was successfully completed.

### Exceptions

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.
- **InvalidParameter** - if the argument “**configurationSet**” is null, or if one of the attributes defining “**configurationSet**” is invalid, or if the specified identifier of the configuration set already exists.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(14) + *setConfigurationSetValues(configurationSet: ConfigurationSet): Boolean*

This operation modifies the specified **ConfigurationSet** of an SDO.

Parameter	Type	Description
<return>	<b>Boolean</b>	A flag indicating if the <b>ConfigurationSet</b> was modified successfully. ‘true’ - The <b>ConfigurationSet</b> was modified successfully. ‘false’ - The <b>ConfigurationSet</b> could not be modified successfully.

## Exceptions

This operation throws **SDOException** with one of the following exception types:

- **InvalidParameter** - if the parameter '**configurationSetID**' is null or if there is no **ConfigurationSet** stored with such id. This exception is also raised if one of the attributes defining **ConfigurationSet** is not valid.
- **SDOExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(15) + *removeConfigurationSet ( configurationSetID : UniqueIdentifier ) : Boolean*

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

Parameter	Type	Description
<b>configurationSetID</b>	<b>UniqueIdentifier</b>	The <b>ConfigurationSet</b> which is removed.
<return>	<b>Boolean</b>	If the operation was successfully completed.

## Exceptions

This operation throws **SDOException** with one of the following exception types:

- **SDOExists** - if the target SDO does not exist.
- **InvalidParameter** - if the arguments “**configurationSetID**” is null, or if the object specified by the argument “**configurationSetID**” does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(16) + *activateConfigurationSet ( configID : UniqueIdentifier ) : Boolean*

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

Parameter	Type	Description
<b>configID</b>	<b>UniqueIdentifier</b>	Identifier of <b>ConfigurationSet</b> to be activated.
<return>	<b>Boolean</b>	If the operation was successfully completed.

## Exceptions

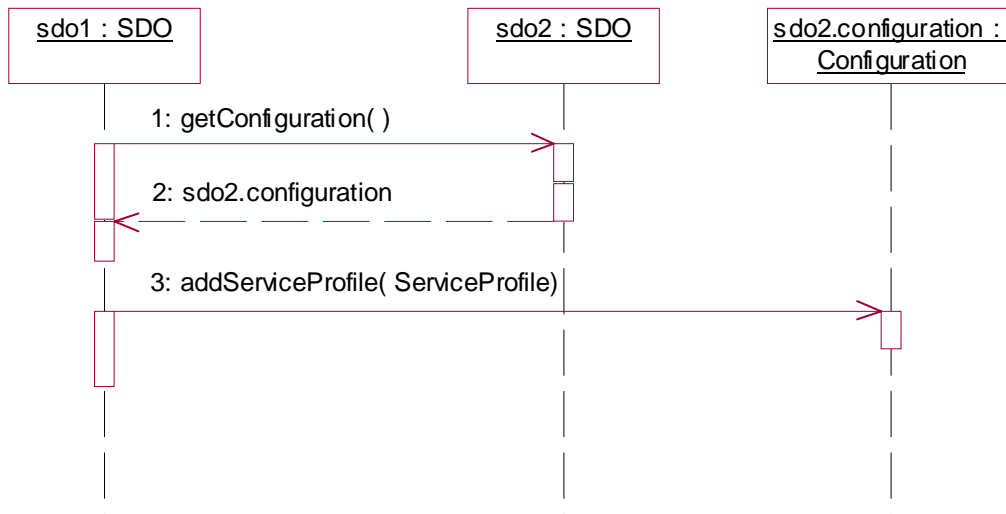
This operation throws **SDOException** with one of the following exception types:

- **SDOExists** - if the target SDO does not exist.
- **InvalidParameter** - if the argument (“**configID**”) is null or there is no configuration set with identifier specified by the argument.

- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

### 7.4.5.2 Usage: Configuration

As an example, the message sequence chart for the case of profile acquisition is shown in Figure 7.5. And, as an example of parameter acquisition, the sequence of “**getConfigurationParameters()**” is shown in Figure 7.6. First, the configuring SDO (sdo1) gets the **Configuration** object by invocation of operations of the SDO that is configured (sdo2). And the sequences of the other operations that belong to the **Configuration** interface as shown in Figure 7.5 and Figure 7.6.

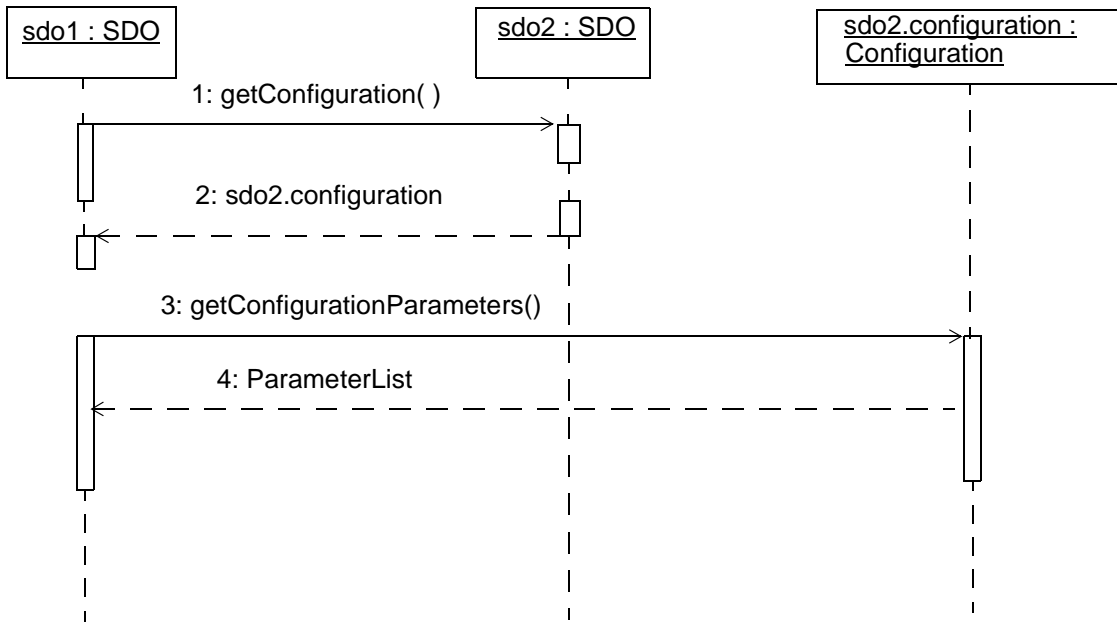


**Figure 7.5 - Sequence Chart: Configuration: Add Service Profile**

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 7.6 - 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.

## 7.4.6 SDOService Interface

**SDOService** is the interface used to define an operation(s) through which an SDO provides its service(s). An **SDOService** object may define an operation for a single service or may define multiple operations for different services, each of which is represented by a **ServiceProfile** object. It is implementation dependent how to design **SDOService** objects.

## 7.4.7 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 numbers 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 provides the functions (**getMonitoringParameterValues()**, and **getMonitoringParameterValue()**) that support the monitoring by polling. The monitoring by polling is described in detail in Section 7.4.7.4.1, “Monitoring by Polling,” on page 41.

Using *subscription* an observing SDO is notified about changes of monitored properties. The observing SDO has to be an SDO that is to be monitored. According to the subscription the monitored SDO notifies the subscriber using its **Callback** interface (see Section 7.4.7.5, “NotificationCallback Interface,” on page 46). The Monitoring interface supports the subscription through appropriate functions (**subscribe()**, **renewSubscription()**, **unsubscribe()**, **unsubscribeAll()**). The monitoring by subscription is described in detail in Section 7.4.7.4.2, “Monitoring by Subscription,” on page 42.

Monitoring
<pre> +getMonitoringParameterValue(name : String) : any +getMonitoringParameters() : ParameterList +getMointoringParameterValues() : NVList +subscribe(data : NotificationSubscription) : Boolean +renewSubscription(subscriber:UniquelIdentifier, duration:unsigned long) : Boolean +unsubscribe(subscriber : UniquelIdentifier, names : StringList) : Boolean +unsubscribe(subscriber : UniquelIdentifier) : Boolean </pre>

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.

### 7.4.7.1 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 7.1 and described in detail individually later on.

**Table 7.1 - Detailed Description of Data Structures Required for Monitoring Interface**

<p><b>NotificationMode</b></p>	<p>Possible notification modes while subscribing to monitoring properties.</p> <div style="border: 1px solid black; padding: 5px; margin: 10px auto; width: fit-content;"> <p style="text-align: center;">NotificationMode</p> <hr/> <p>+ON_CHANGE +ON_INTERVAL</p> </div> <p><b>ON_CHANGE</b> - To be notified of the subscribed monitoring property every time the value of the parameter changes.</p> <p><b>ON_INTERVAL</b> - 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.</p>
<p><b>NotificationSubscription</b></p>	
<pre> classDiagram     class NotificationSubscription {         startime : unsigned long         duration : unsigned long         notificationInterval : unsigned long     }     class NotificationCallback {         notify(publisher : SDO, publisherID : UniquelIdentifier, currentStatus : NVList) : void     }     class UniquelIdentifier     class NotificationMode {         ON_CHANGE         ON_INTERVAL     }     class StringList      NotificationSubscription --&gt; NotificationCallback : +subscriber     NotificationSubscription --&gt; UniquelIdentifier : +subscriberID     NotificationSubscription --&gt; NotificationMode : +notifyMode     NotificationSubscription --&gt; StringList : +observedData     </pre>	
	<p>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.</p>

	<ul style="list-style-type: none"> <li>• <b>subscriber</b> - The address or reference of the object that will receive notification messages; the object referenced here must implement the interface <b>NotificationCallback</b>. The value of this field depends on basis technology of SDO system. Please read Section 7.4.7.5, “NotificationCallback Interface,” on page 46 for more details.</li> <li>• <b>subscriberID</b> - The unique identifier of the SDO that subscribes to this property.</li> <li>• <b>notifyMode</b> - The mode of notification (<b>ON_CHANGE</b> or <b>ON_INTERVAL</b>). 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 <b>notificationInterval</b>.</li> <li>• <b>observedData</b> - List of names of monitoring properties to be subscribed.</li> <li>• <b>startTime</b> - 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.</li> <li>• <b>duration</b> - Indicates for how long (in milliseconds) the subscription should last.</li> <li>• <b>notificationInterval</b> - Time interval (in milliseconds) in which the notification is sent to the subscribing SDO, if the <b>NotificationMode</b> of the subscription is <b>ON_INTERVAL</b>.</li> </ul>
--	---

### 7.4.7.2 Operations provided by Monitoring Interface

This section describes all the operations provided by the **Monitoring** interface. All operations are initially listed and then each is described with examples.

Name	Short Description
<b>getMonitoringParameterValue (name : String)</b>	To get the value of the specified property.
<b>getMonitoringParameters () : ParameterList</b>	To get the list of all monitoring properties.
<b>getMonitoringParameterValues () : NVList</b>	To get all the monitoring properties with their current values.
<b>subscribe (data : NotificationSubscription) : Boolean</b>	This operation subscribes necessary monitoring properties with certain conditions.
<b>renewSubscription (subscriber : UniquelIdentifier, duration : unsigned long) : Boolean</b>	This operation renews the already subscribed properties for the specified duration (in milliseconds) of time.
<b>unsubscribe (subscriber : UniquelIdentifier, names : StringList) : Boolean</b>	This operation unsubscribes the specified list of already subscribed monitoring properties.
<b>unsubscribeAll (subscriber : UniquelIdentifier) : Boolean</b>	This operation unsubscribes all the subscribed properties of the specified SDO.

### 7.4.7.3 Detailed description of all operations

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

(1) *+getMonitoringParameterValue (name : String) : any*

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

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

### Exceptions

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.
- **InvalidParameter** - if the argument 'name' is null or a monitoring property named 'name' does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(2) *+getMonitoringParameters () : ParameterList*

This operation returns the list of monitoring properties defined for this SDO. An empty list is returned if the SDO does not have any monitoring parameters.

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

### Exceptions

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(3) *+getMonitoringParameterValues () : NVList*

This operation returns the current values of all the monitoring properties of the SDO. An empty list is returned if the SDO does not have any monitoring parameters.

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



## Exceptions

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

### (4) *+subscribe (data : NotificationSubscription) : Boolean*

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
<b>data</b>	<b>NotificationSubscription</b>	Properties being subscribed and the conditions for the subscription.
<return>	Boolean	If the operation was successfully completed.

## Exceptions

This operation throws **SDOException** with one of the following exception types:

- **InvalidParameter** - 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 the non-existing property must be specified in the exception data.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.
- **SDONotExists** - if the target SDO does not exist.

### (5) *+renewSubscription (subscriber : UniqueIdentifier, duration : unsigned long) : Boolean*

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

Parameter	Type	Description
<b>subscriber</b>	<b>UniqueIdentifier</b>	Unique ID of the SDO that is renewing the subscription.
<b>duration</b>	<b>unsigned long</b>	Time duration (in milliseconds) until which the subscriptions of the specified SDO should be renewed.
<return>	<b>Boolean</b>	If the operation was successfully completed.

## Exception

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.
- **InvalidParameter** - if one or more properties specified by the attribute 'names' have not been subscribed before or does not exist. This exception is also raised if the specified 'duration' is outside the predefined limit.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(6) *+unsubscribe (subscriber : UniqueIdentifier, names : StringList) : Boolean*

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

Parameter	Type	Description
<b>subscriber</b>	<b>UniqueIdentifier</b>	Unique ID of the SDO that is unsubscribing.
<b>names</b>	<b>StringList</b>	List of names of the properties being unsubscribed.
<return>	<b>Boolean</b>	If the operation was successfully completed.

## Exception

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** if the target SDO does not exist.
- **InvalidParameter** if the stated properties (parameter names) to be unsubscribed does not exist or was not subscribed.
- **NotAvailable** if the target SDO is reachable but cannot respond.
- **InternalError** if the target SDO cannot execute the operation completely due to some internal error.

(7) *+unsubscribeAll (subscriber : UniqueIdentifier) : Boolean*

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

Parameter	Type	Description
<b>subscriber</b>	<b>UniqueIdentifier</b>	Unique ID of the SDO that is unsubscribing all its subscriptions.
<return>	<b>Boolean</b>	If the operation was successfully completed.

## Exception

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.

- **InvalidParameter** - if there are no subscriptions from the observer SDO defined by the parameter **subscriber** to unsubscribe.
- **NotAvailable** - if the target SDO is reachable but cannot respond.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

#### 7.4.7.4 Usage: Monitoring Interface

##### 7.4.7.4.1 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 7.7. In the diagram shown in the picture, sdo1 makes requests to sdo2 to acquire its status.

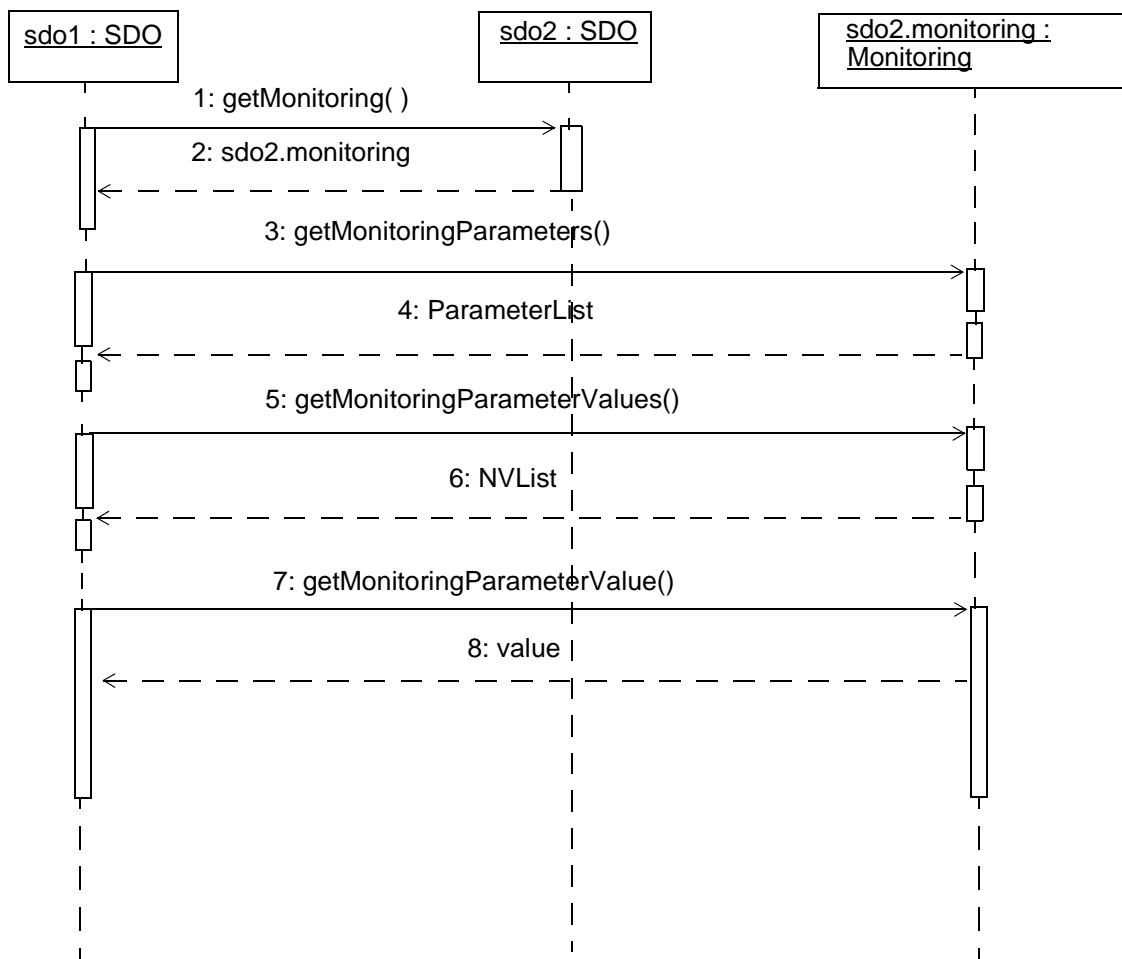


Figure 7.7 - Sequence Chart: Monitoring

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.

#### 7.4.7.4.2 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 the time validity of the subscription should be. 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:

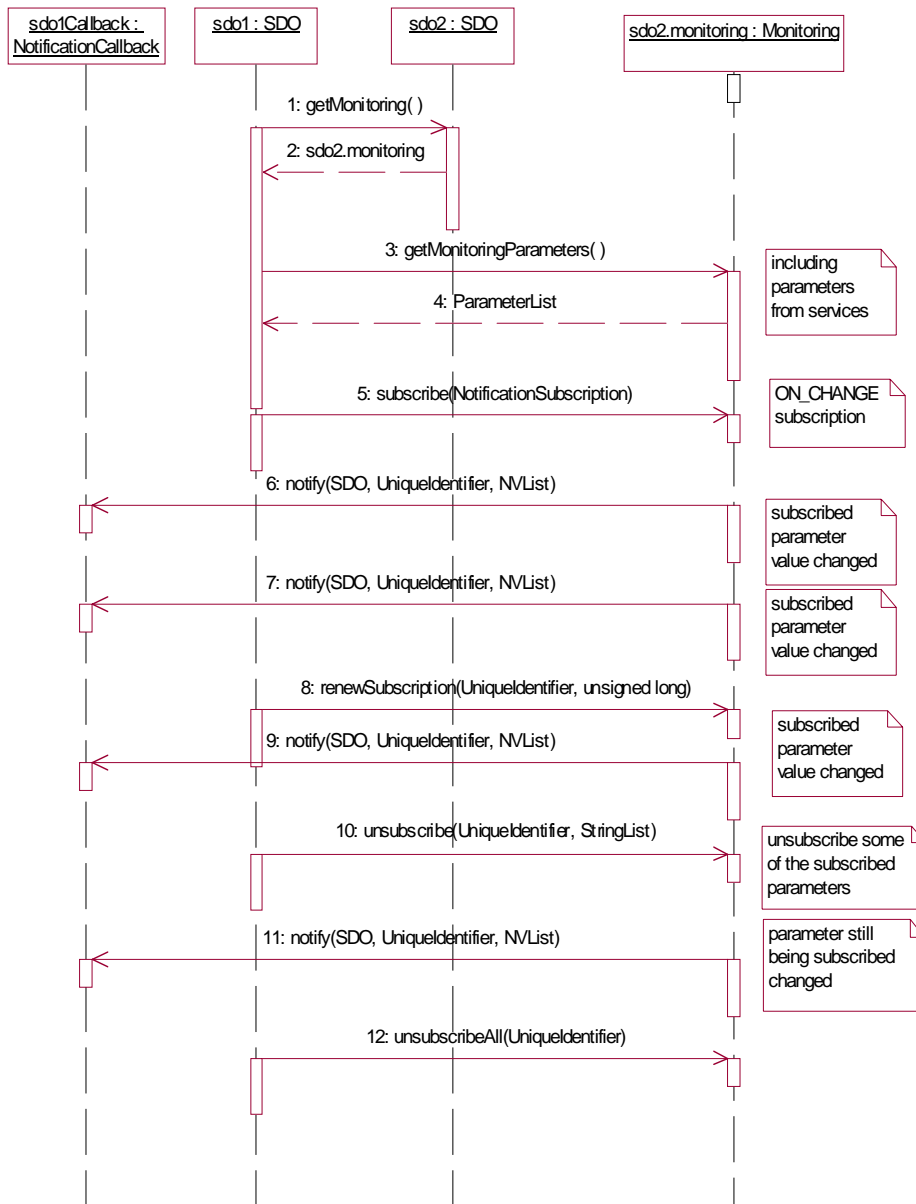
- when the value of the monitored properties change (**ON\_CHANGE** mode), or
- on certain time interval (**ON\_INTERVAL** mode).

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 7.4.7.5, “NotificationCallback Interface,” on page 46.

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 7.3.11, ‘Examples of resource data model’ 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 7.8 - Subscription and Notification in ON\_CHANGE mode**

The subscription of properties is shown in the sequence diagram (Figure 7.8). The interaction depicted in Figure 7.8 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.

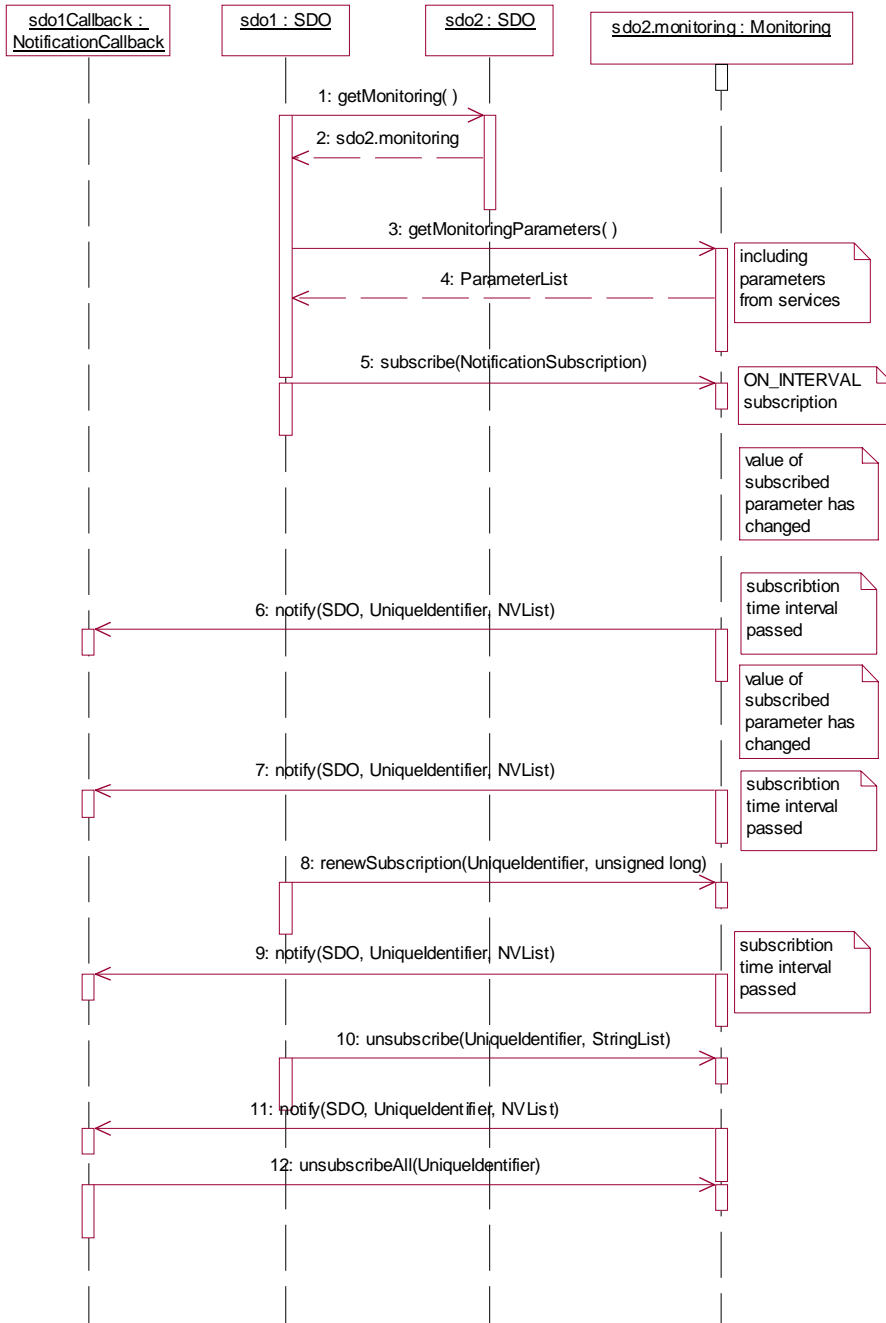
Message 10: sdo1 unsubscribes one or more properties subscribed previously.

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 7.9 - Subscription and Notification ON\_INTERVAL**

Figure 7.9 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 7.8 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 7.9 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 7.9 between notification messages 9 and 11.

#### 7.4.7.5 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.

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

#### 7.4.7.6 Data Structures Defined for NotificationCallback Interface

Two general data structures (**UniqueIdentifier** and **NVList**) are used in the **NotificationCallback** interface. Please see Section 7.4.7.1, “Data Structures Defined for Monitoring Interface,” on page 35 for their detail description.

#### 7.4.7.7 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
<b>publisherID</b>	<b>UniqueIdentifier</b>	Unique identifier of the SDO that is sending the notification.
<b>currentStatus</b>	<b>NVList</b>	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.

#### 7.4.7.8 Usage: NotificationCallback Interface

The examples of usage of operations of **NotificationCallback** interface are shown in Figure 7.8 and Figure 7.9. The operations are used to convey the changes in values of monitoring properties to notification subscriber.



## 7.4.8 Organization Interface

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

Organization
<pre> +getOrganizationID():UniqueIdentifier +addOrganizationProperty(organizationProperty:OrganizationProperty):Boolean +getOrganizationProperty():OrganizationProperty +getOrganizationPropertyValue(name:String):any +setOrganizationPropertyValue(name:String,value:any):any +removeOrganizationProperty(name:String):Boolean +addMembers(sdoList:SDOList):Boolean +getMembers():SDOList +setMembers(sdos:SDOList):Boolean +removeMembers(sdoID:UniqueIdentifier):Boolean +getOwner():SDOSystemElement +setOwner(sdo:SDOSystemElement : Boolean +getDependency():DependencyType +setDependencyType(dependency:DependencyType):Boolean </pre>

### (1) + *getOrganizationID() : UniqueIdentifier*

This operation returns the 'id' of the **Organization**.

Parameter	Type	Description
<return>	<b>UniqueIdentifier</b>	The identifier of the Organization defined in the resource data model.

### Exception

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if there is no response from the target object.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

### (2) + *addOrganizationProperty (organizationProperty : OrganizationProperty) : Boolean*

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

Parameter	Type	Description
<b>organizationProperty</b>	<b>OrganizationProperty</b>	The type of organization to be added.
<return>	Boolean	If the operation was successfully completed.

## Exception

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.
- **InvalidParameter** - if the argument '**organizationProperty**' is null.
- **NotAvailable** - if there is no response from the target object.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

### (3) + *getOrganizationProperty()*: *OrganizationProperty*

This operation returns the **OrganizationProperty** that an Organization has. An empty **OrganizationProperty** is returned if the **Organization** does not have any properties.

Parameter	Type	Description
<return>	<b>OrganizationProperty</b>	The list with properties of the organization.

## Exception

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.
- **NotAvailable** - if there is no response from the target object.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

### (4) + *getOrganizationPropertyValue(name : String)*: *any*

This operation returns a value in the **OrganizationProperty**. The value to be returned is specified by argument '**name**.'

Parameter	Type	Description
<b>Name</b>	<b>String</b>	The name of the value to be returned.
<return>	<b>Any</b>	The value of property which is specified by argument ' <b>name</b> .'

## Exception

This operation throws **SDOException** with one of the following exception types:

- **InvalidParameter** - if the property which is specified by argument "name" does not exist.
- **NotAvailable** - if there is no response from the target object.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.
- **SDONotExists** - if the target SDO does not exist.

(5) + *setOrganizationPropertyValue* (name : String, value : any): Boolean

This operation adds or updates a pair of name and value as a property of **Organization** to/in **NVList** of the **OrganizationProperty**. The name and the value to be added/updated are specified by argument ‘name’ and ‘value.’

Parameter	Type	Description
Name	String	The name of the property to be added/updated.
Value	Any	The value of the property to be added/updated.

**Exception**

This operation throws **SDOException** with one of the following exception types:

- **NotAvailable** - if there is no response from the target object.
- **SDONotExists** - if the target SDO does not exist.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.
- **InvalidParameter** - if the property that is specified by argument “name” does not exist.

(6) + *removeOrganizationProperty* (name : String): Boolean

This operation removes a property of **Organization** from **NVList** of the **OrganizationProperty**. The property to be removed is specified by argument ‘name.’

Parameter	Value	Description
Name	String	The name of the property to be removed.

**Exception**

This operation throws **SDOException** with one of the following exception types:

- **InvalidParameter** - if the property that is specified by argument “name” does not exist.
- **NotAvailable** - if there is no response from the target object.
- **SDONotExists** - if the target SDO does not exist.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(7) + *addMembers*(sdoList : SDOList) : Boolean

This operation adds a member that is an SDO to the organization. The member to be added is specified by argument ‘sdo.’

Parameter	Value	Description
sdo	SDO	The member to be added to the organization.

## Exception

This operation throws **SDOException** with one of the following exception types:

- **InvalidParameter** - if argument “sdo” is null.
- **NotAvailable** - if there is no response from the target object.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.
- **SDONotExists** - if the target SDO does not exist.

### (8) + *getMembers ( ) : SDOList*

This operation returns a list of **SDOs** that are members of an **Organization**. An empty list is returned if the **Organization** does not have any members.

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

## Exception

This operation throws **SDOException** with one of the following exception types:

- **NotAvailable** - if there is no response from the target object.
- **SDONotExists** - if the target SDO does not exist.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

### (9) + *setMembers (sdos : SDOList) : Boolean*

This operation assigns a list of **SDOs** to an **Organization** as its members. If the **Organization** has already maintained a member **SDO(s)** when it is called, the operation replaces the member(s) with specified list of **SDOs**.

Parameter	Type	Description
<b>sdos</b>	<b>SDOList</b>	Member <b>SDOs</b> to be assigned.
<return>	<b>Boolean</b>	If the operation was successfully completed.

## Exception

This operation throws **SDOException** with one of the following exception types:

- **InvalidParameter** - if argument “SDOList” is null, or if the object that is specified by the argument “sdos” does not exist.
- **NotAvailable** - if there is no response from the target object.
- **SDONotExists** - if the target SDO does not exist.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(10) + *removeMember(sdoID : UniqueIdentifier) : Boolean*

This operation removes a member from the organization. The member to be removed is specified by argument 'id.'

Parameter	Value	Description
<b>sdoID</b>	<b>UniqueIdentifier</b>	Id of the SDO to be removed from the organization.

**Exception**

This operation throws **SDOException** with one of the following exception types:

- **SDONotExists** - if the target SDO does not exist.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.
- **InvalidParameter** - if argument "id" is null or does not exist.
- **NotAvailable** - if there is no response from the target object.

(11) + *getOwner() : SDOSystemElement*

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

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

**Exception**

This operation throws **SDOException** with one of the following exception types:

- **NotAvailable** - if there is no response from the target object.
- **SDONotExists** - if the target SDO does not exist.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(12) + *setOwner(sdo : SDOSystemElement) : Boolean*

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

Parameter	Type	Description
<b>sdo</b>	<b>SDOSystemElement</b>	Reference of owner object.
<return>	<b>Boolean</b>	If the operation was successfully completed.

**Exception**

This operation throws **SDOException** with one of the following exception types:

- **InvalidParameter** - if argument “**sdo**” is null, or if the object that is specified by “**sdo**” in argument “**sdo**” does not exist.
- **NotAvailable** - if there is no response from the target object.
- **SDONotExists** - if the target SDO does not exist.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(13) + *getDependency() : DependencyType*

This operation gets the relationship ‘**DependencyType**’ of the **Organization**.

Parameter	Type	Description
<return>	<b>DependencyType</b>	The relationship of the <b>Organization</b> as <b>DependencyType</b> . <b>DependencyType</b> is defined in Section 7.3.2, “Data Structures Used by Resource Data Model,” on page 4.

**Exception**

This operation throws **SDOException** with one of the following exception types:

- **NotAvailable** - if there is no response from the target object.
- **SDONotExists** - if the target SDO does not exist.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

(14) + *setDependency(dependency : DependencyType) : Boolean*

This operation sets the relationship ‘**DependencyType**’ of the **Organization**. The value to be set is specified by argument ‘**dependency**.’

Parameter	Type	Description
<b>dependency</b>	<b>DependencyType</b>	The relationship of the <b>Organization</b> as <b>DependencyType</b> . <b>DependencyType</b> is defined in Section 7.3.2, “Data Structures Used by Resource Data Model,” on page 4.
<return>	<b>Boolean</b>	If the <b>DependencyType</b> is set successfully.

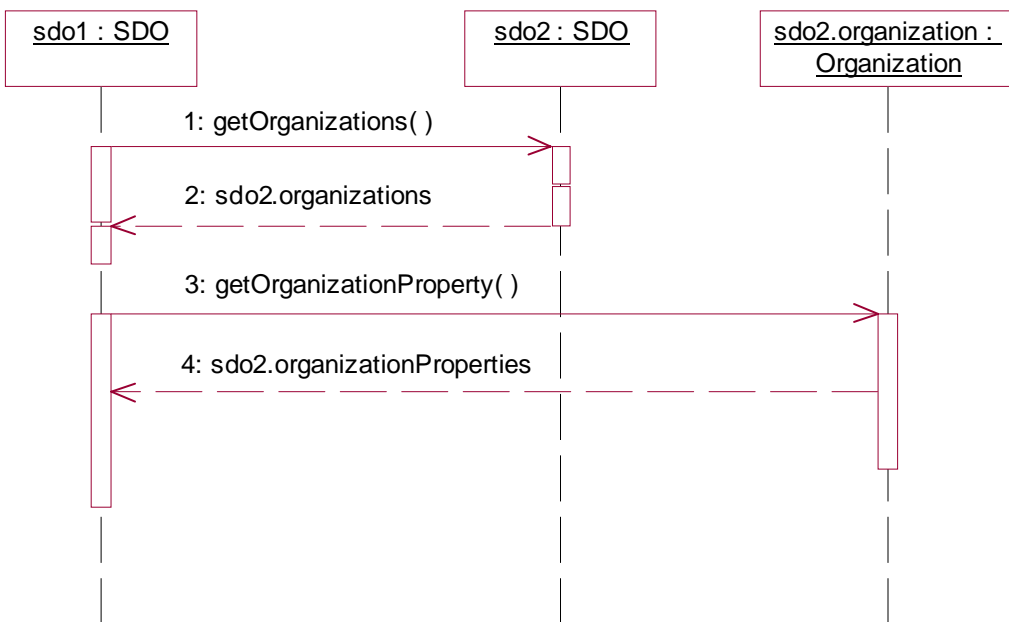
**Exception**

This operation throws **SDOException** with one of the following exception types:

- **InvalidParameter** - if argument “**dependency**” is null.
- **NotAvailable** - if there is no response from the target object.
- **SDONotExists** - if the target SDO does not exist.
- **InternalError** - if the target SDO cannot execute the operation completely due to some internal error.

### 7.4.8.1 Usage: Organization

Figure 7.10 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 7.10 - 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.





## 8 Platform Specific Model: Mapping to CORBA IDL

### 8.1 Overview

This chapter introduces a CORBA specific model for the SDO PIM defined in Chapter 7. The selected platform is CORBA version 3.0.

The Super Distributed Objects (SDO) Platform Independent Model (PIM) defines the resource data model, interfaces, and necessary data structures for SDOs. In the Platform Specific Model (PSM) these interfaces and the data structures used in the individual methods are mapped according to a CORBA IDL specification. The complete IDL specification is presented in Chapter 9.

An interface defined in the SDO PIM is mapped to a CORBA interface. An operation in a PIM interface is mapped to a 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>`. A 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].

### 8.2 SDO Module

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

### 8.3 Data Types used in CORBA PSM

In 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 {
    OWN,
    OWNED,
    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;

```

## 8.4 Exceptions

The methods of SDO interfaces can raise **SDOException**. The kind of exception is defined in the attribute *type* (see Section 7.4.2.1, “SDOException,” on page 18). 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**, **NotFound**, and **InternalError**.

```

#define exception_body { string description; }
...
exception NotAvailable           exception_body;
exception InterfaceNotImplemented exception_body;
exception InvalidParameter       exception_body;
exception InternalError          exception_body;

```

The exception **SDONotExists** defined in the PIM is mapped to CORBA standard system exception **OBJECT\_NOT\_EXIST**.

## 8.5 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 that each SDO implementation at least must 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 the following interface declarations:

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

### 8.5.1 SDOSystemElement Interface

The **SDOSystemElement** interface is mapped to a 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 supports an operation, **get\_owned\_organizations**, which allows getting the list of organizations associated with the object implementing this interface.

```
interface SDOSystemElement {
    OrganizationList get_owned_organizations()
        raises (NotAvailable, InternalError);
};
```

### 8.5.2 SDO Interface

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

```
interface SDO : SDOSystemElement {
    UniquelIdentifier get_sdo_id()
        raises (NotAvailable, InternalError);
    string get_sdo_type()
        raises (NotAvailable, InternalError);
    DeviceProfile get_device_profile ()
        raises (NotAvailable, InternalError);
    ServiceProfileList get_service_profiles ()
        raises (NotAvailable, InternalError);
    ServiceProfile get_service_profile (in UniquelIdentifier id)
        raises (InvalidParameter, NotAvailable, InternalError);
    SDOService get_sdo_service (in UniquelIdentifier id)
        raises (InvalidParameter, NotAvailable, InternalError);
    Configuration get_configuration ()
        raises (InterfaceNotImplemented, NotAvailable, InternalError);
    Monitoring get_monitoring ()
        raises (InterfaceNotImplemented, NotAvailable, InternalError);
    OrganizationList get_organizations ()
        raises (NotAvailable, InternalError);
    NVList get_status_list ()
        raises (NotAvailable, InternalError);
    any get_status ()
        raises (InvalidParameter, NotAvailable, InternalError);
};
```

### 8.5.3 Configuration Interface

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

```
interface Configuration {
    boolean set_device_profile (in DeviceProfile dProfile)
        raises (InvalidParameter, NotAvailable, InternalError);
    boolean add_service_profile (in ServiceProfile sProfile)
```

```

        raises (InvalidParameter, NotAvailable, InternalError);
    boolean add_organization (in Organization organization)
        raises (InvalidParameter, NotAvailable, InternalError);
    boolean remove_service_profile (in UniqueIdentifier id)
        raises (InvalidParameter, NotAvailable, InternalError);
    boolean remove_organization (in UniqueIdentifier organization_id)
        raises (InvalidParameter, NotAvailable, InternalError);
    ParameterList get_configuration_parameters ()
        raises (NotAvailable, InternalError);
    NVList get_configuration_parameter_values ()
        raises (NotAvailable, InternalError);
    any get_configuration_parameter_value (in string name)
        raises (InvalidParameter, NotAvailable, InternalError);
    boolean set_configuration_parameter (
        in string name,
        in any value)
        raises (InvalidParameter, NotAvailable, InternalError);
    ConfigurationSetList get_configuration_sets ()
        raises (NotAvailable, InternalError);
    ConfigurationSet get_configuration_set (in UniqueIdentifier config_id)
        raises (NotAvailable, InternalError);
    boolean set_configuration_set_values (
        in ConfigurationSet configuration_set)
        raises (InvalidParameter, NotAvailable, InternalError);
    ConfigurationSet get_active_configuration_set ()
        raises (NotAvailable, InternalError);
    boolean add_configuration_set (in ConfigurationSet configuration_set)
        raises (InvalidParameter, NotAvailable, InternalError);
    boolean remove_configuration_set (in UniqueIdentifier config_id)
        raises (InvalidParameter, NotAvailable, InternalError);
    boolean activate_configuration_set (in UniqueIdentifier config_id)
        raises (InvalidParameter, NotAvailable, InternalError);
};

```

#### 8.5.4 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.

#### 8.5.5 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 7.4.7, “Monitoring Interface,” on page 34 are implemented using the OMG Notification Service [4].

```

interface Monitoring : CosNotifyComm::StructuredPushConsumer,
    CosNotifyComm::StructuredPushSupplier {
    any get_monitoring_parameter_value (
        in string name

```

```

        ) raises (InvalidParameter, NotAvailable, InternalError);
    ParameterList get_monitoring_parameters ()
        raises (NotAvailable, InternalError);
    NVList get_monitoring_parameter_values ()
        raises (NotAvailable, InternalError);
};

```

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 7.4.7.2, “Operations provided by Monitoring Interface,” on page 37.

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 7.4.7.7, “Operations provided by NotificationCallback Interface,” on page 46.

## 8.5.6 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 {
    UniqueIdentifier get_organization_id ()
        raises (InvalidParameter, NotAvailable, InternalError);
    OrganizationProperty get_organization_property ()
        raises (NotAvailable, InternalError);
    any get_organization_property_value (in string name)
        raises (InvalidParameter, NotAvailable, InternalError);
    boolean add_organization_property (
        in OrganizationProperty organization_property
    ) raises (InvalidParameter, NotAvailable, InternalError);
    boolean set_organization_property_value (
        in string name,
        in any value
    ) raises (InvalidParameter, NotAvailable, InternalError);
    boolean remove_organization_property ( in string name )
        raises (InvalidParameter, NotAvailable, InternalError);
    SDOSystemElement get_owner ()

```

```
        raises (NotAvailable, InternalError);
boolean set_owner (in SDOSystemElement sdo)
    raises (InvalidParameter, NotAvailable, InternalError);
SDOList get_members ()
    raises (NotAvailable, InternalError);
boolean set_members (in SDOList sdos)
    raises (InvalidParameter, NotAvailable, InternalError);
boolean add_members ( in SDOList sdo_list)
    raises (InvalidParameter, NotAvailable, InternalError);
boolean remove_member (in UniqueIdentifier id)
    raises (InvalidParameter, NotAvailable, InternalError);
DependencyType get_dependency()
    raises (NotAvailable, InternalError);
boolean set_dependency (in DependencyType dependency)
    raises (NotAvailable, InternalError);
};
```





## 9 OMG IDL

### 9.1 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 {
    OWN,
    OWNED,
    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;
exception InternalError exception_body;

/** ----- Interfaces -----*/
interface SDOSystemElement {
    OrganizationList get_owned_organizations()
        raises (NotAvailable);
};
interface SDO : SDOSystemElement {
    UniqueIdentifier get_sdo_id()
        raises (NotAvailable, InternalError);
    string get_sdo_type()
        raises (NotAvailable, InternalError);
    DeviceProfile get_device_profile ()
        raises (NotAvailable, InternalError);
    ServiceProfileList get_service_profiles ()
        raises (NotAvailable, InternalError);
    ServiceProfile get_service_profile (in UniqueIdentifier id)
        raises (InvalidParameter, NotAvailable, InternalError);
    SDOService get_sdo_service (in UniqueIdentifier id)
        raises (InvalidParameter, NotAvailable, InternalError);
    Configuration get_configuration ()
        raises (InterfaceNotImplemented, NotAvailable, InternalError);
    Monitoring get_monitoring ()
        raises (InterfaceNotImplemented, NotAvailable, InternalError);
    OrganizationList get_organizations ()
        raises (NotAvailable, InternalError);
    NVList get_status_list ()
        raises (NotAvailable, InternalError);
    any get_status (in string name)
        raises (InvalidParameter, NotAvailable, InternalError);
};

interface Configuration {
    boolean set_device_profile (in DeviceProfile dProfile)
        raises (InvalidParameter, NotAvailable, InternalError);
    boolean set_service_profile (in ServiceProfile sProfile)
        raises (InvalidParameter, NotAvailable, InternalError);
    boolean add_organization (in Organization organization_object)
        raises (InvalidParameter, NotAvailable, InternalError);
    boolean remove_service_profile (in UniqueIdentifier id)
        raises (InvalidParameter, NotAvailable, InternalError);
    boolean remove_organization (in UniqueIdentifier organization_id)

```

```

        raises (InvalidParameter, NotAvailable, InternalError);
ParameterList get_configuration_parameters ()
    raises (NotAvailable, InternalError);
NVList get_configuration_parameter_values ()
    raises (NotAvailable, InternalError);
any get_configuration_parameter_value (in string name)
    raises (InvalidParameter, NotAvailable, InternalError);
boolean set_configuration_parameter (
    in string name,
    in any value)
    raises (InvalidParameter, NotAvailable, InternalError);
ConfigurationSetList get_configuration_sets ()
    raises (NotAvailable, InternalError);
ConfigurationSet get_configuration_set
    (in UniquelIdentifier config_id)
    raises (NotAvailable, InternalError);
boolean set_configuration_set_values (
    in UniquelIdentifier config_id,
    in ConfigurationSet configuration_set)
    raises (InvalidParameter, NotAvailable, InternalError);
ConfigurationSet get_active_configuration_set ()
    raises (NotAvailable, InternalError);
boolean add_configuration_set
    (in ConfigurationSet configuration_set)
    raises (InvalidParameter, NotAvailable, InternalError);
boolean remove_configuration_set (in UniquelIdentifier config_id)
    raises (InvalidParameter, NotAvailable, InternalError);
boolean activate_configuration_set (in UniquelIdentifier config_id)
    raises (InvalidParameter, NotAvailable, InternalError);
};

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

interface SDOService {};

interface Organization {
    UniquelIdentifier get_organization_id ()
        raises (InvalidParameter, NotAvailable, InternalError);
OrganizationProperty get_organization_property ()
    raises (NotAvailable, InternalError);
any get_organization_property_value (in string name)
    raises (InvalidParameter, NotAvailable, InternalError);
};

```

```

    boolean set_organization_property (
        in OrganizationProperty organization_property
    ) raises (InvalidParameter, NotAvailable, InternalError);
    boolean set_organization_property_value (
        in string name,
        in any value
    ) raises (InvalidParameter, NotAvailable, InternalError);
    boolean remove_organization_property ( in string name )
        raises (InvalidParameter, NotAvailable, InternalError);
    SDOSystemElement get_owner ()
        raises (NotAvailable, InternalError);
    boolean set_owner (in SDOSystemElement sdo)
        raises (InvalidParameter, NotAvailable, InternalError);
    SDOList get_members ()
        raises (NotAvailable, InternalError);
    boolean set_members (in SDOList sdos)
        raises (InvalidParameter, NotAvailable, InternalError);
    boolean add_members ( in SDOList sdo_list)
        raises (InvalidParameter, NotAvailable, InternalError);
    boolean remove_member (in UniqueIdentifier id)
        raises (InvalidParameter, NotAvailable, InternalError);
    DependencyType get_dependency()
        raises (NotAvailable, InternalError);
    boolean set_dependency (in DependencyType dependency)
        raises (NotAvailable, InternalError);
};
};
#endif // _SDO_PACKAGE_IDL_

```



## Annex A: References

- [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





# Annex B: Complete UML Diagram

## B.1 Complete UML Diagram of SDO Resource Data Model

The complete UML diagram of SDO is as follows.

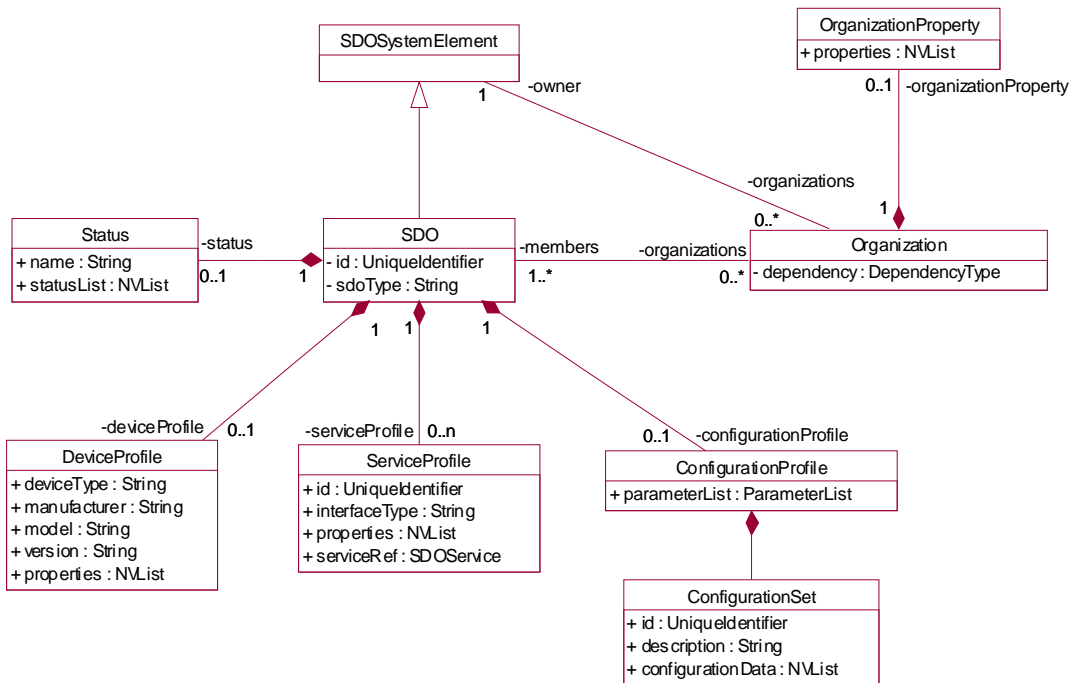


Figure B.1 - Complete UML Diagram of the SDO resource data model

## B.1 Mapping to ECHONET

### B.1.1 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.

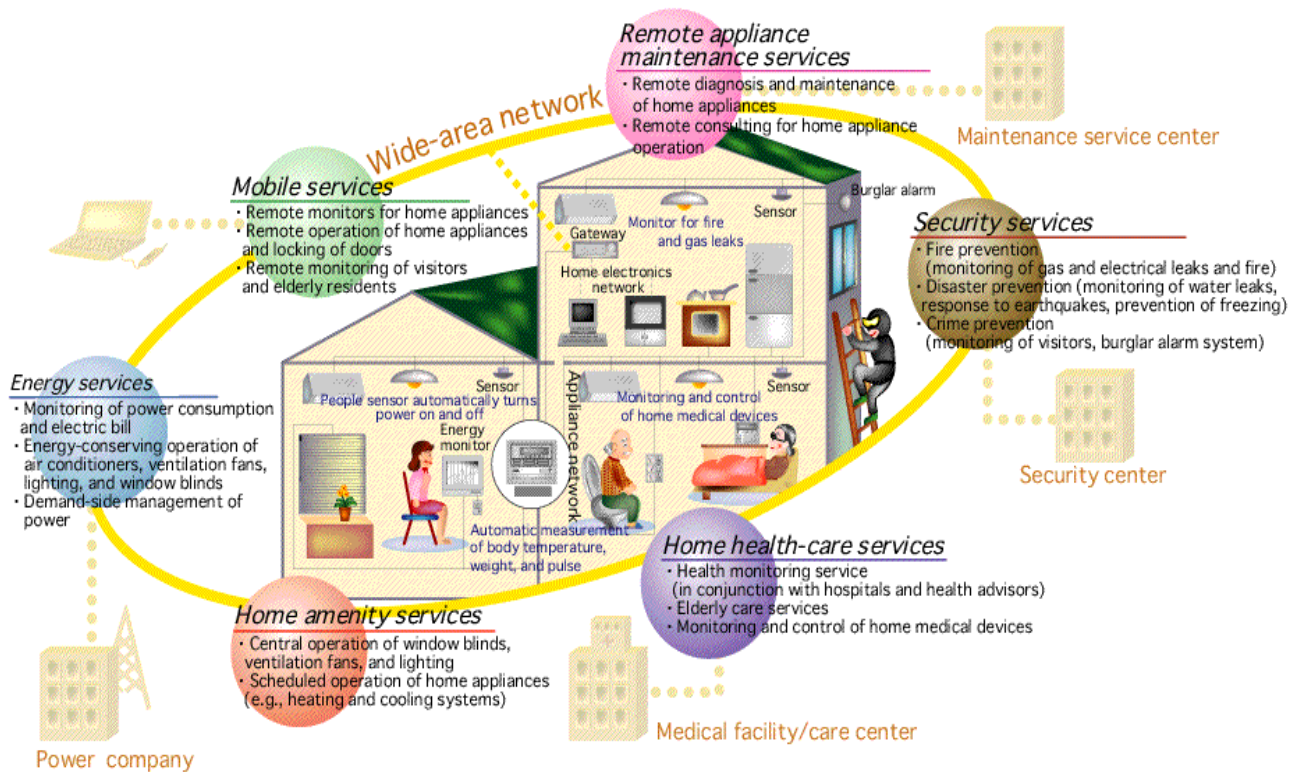


Figure B.2 - Application Areas of ECHONET

### B.1.2 ECHONET Architecture

(ref. [http://www.echonet.gr.jp/english/1\\_echo/index.htm](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 (Application Program Interface) 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 (Home Business System) as a platform for development.

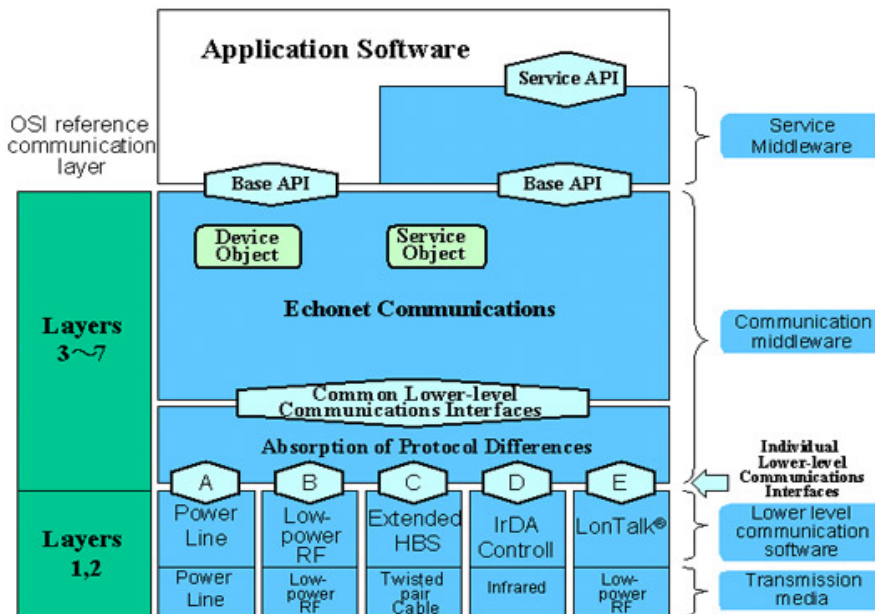


Figure B.3 - ECHONET Architecture

- \*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.1 Mapping SDO to ECHONET

### B.1.1 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.1.2 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](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.1.3 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.

# INDEX

## A

Acknowledgements 2  
Additional Information 2

## C

Common interfaces 74  
Configuration interface 24, 58  
ConfigurationProfile 15  
ConfigurationSet 15  
Conformance 1  
CORBA IDL specification 55  
CORBA PSM 55  
CORBA specific model 55

## D

data structures 4, 18  
Definitions 2  
DeviceProfile 12

## E

ECHONET 71  
ECHONET architecture 72

## F

Flat organization 11

## H

Hierarchical organization 11

## I

Interfaces 17

## M

mapping SDO to ECHONET 73  
monitoring by polling 41  
monitoring by subscription 42  
Monitoring interface 34, 59

## N

Normative References 2  
NotificationCallback interface 46

## O

ON\_CHANGE mode 42  
ON\_INTERVAL mode 44  
Organization 10  
Organization interface 47, 60  
OrganizationProperty 12

## P

Platform Independent Model (PIM) 1, 3  
Platform Specific Model (PSM) 1, 55

properties 9  
property mapping from ECHONET to SDO 74

## R

References 2  
resource data model 3, 15  
Reversely hierarchical organization 11

## S

Scope 1  
SDO 9  
SDO interface 19, 58  
SDOException 18, 57  
SDOService interface 34, 59  
SDOSystemElement 8, 19  
SDOSystemElement interface 58  
ServiceProfile 13  
Status 14  
Super Distributed Objects (SDO) 1, 3  
Symbols 2

## T

Terms and definitions 2

## U

UML diagram of SDO 71  
UML specification 3  
Usage  
    Configuration 33  
    Organization 53

