Revised Submission to Telecom Domain Taskforce RFP

Platform Independent Model (PIM) and Platform Specific Model (PSM) for SDO

Ver. 1.0

sdo/03-03-01

Submitted by
Hitachi Ltd.
Fraunhofer Fokus

March 3, 2003

Content

1 Preface

This document contains a revised joint submission to a Request for Proposal titled *PIM and PSM for SDO* (sdo / 02-01-04).

1.1 Submitters

This initial submission in response to the PIM and PSM for SDO RFP is made by Hitachi Ltd. and Fraunhofer Fokus. This submission is supported by University of California Irvine.

1.2 Contact points

Questions about this submission should be addressed to:

- **Submitter**

Shigetoshi Sameshima

Hitachi Ltd., System Development Laboratory

1009, Ohzenji, Asao-ku, Kawasaki-shi,

Kanagawa-ken, 215-0013 Japan

Phone: +81-44-966-9111

Fax: +81-44-959-0851

E-Mail: samesima@sdl.hitachi.co.jp

Stephan Steglich

Fraunhofer Institute FOKUS

Kaiserin-Augusta-Allee 31

10589 Berlin, Germany

Phone: +49 30 34637373

E-mail: steglich@fokus.fhg.de

- **Supporter**

Junichi Suzuki

University of California, Irvine

Department of Information and Computer Science

Irvine, CA 92697-3425, USA

Phone: +1-949-824-3097

E-mail: jsuzuki@ics.uci.edu

1.3 Guide to the Submission

An SDO (Super Distributed Object) is a logical representation of a hardware device or software component operating on highly distributed system. This specification addresses the information and computational aspects for SDOs, and describes a PIM (Platform Independent Model) and CORBA PSM (Platform Specific Model) for each aspect.

This specification defines the SDO resource data model, which is used to describe the capabilities and properties of SDOs, from an information viewpoint of SDOs. It also defines the interfaces to access and manipulate the resource data from a computational viewpoint of SDOs.

The PIM described in this specification is defined with Unified Modeling Language (UML), and the CORBA PSM is described with CORBA Interface Definition Language (IDL).

1.4 How to read this document

This specification is organized as follows. Chapter 1 contains the information required to accompany a submission. This section identifies the submitter of this specification and the information required to satisfy the business requirements incumbent upon all adopted technology. Chapter 2 provides an overview of the technology advances related to SDO systems (i.e., highly distributed systems, see Section 2) and describes motivations to this specification. Chapter 3 specifies the PIM for the resource data model and the interfaces to access and manipulate resource data. Chapter 4 defines the CORBA PSM for them. Chapter 5 lists the CORBA IDL definition. Chapter 6 summarizes this submission. Appendix A and Appendix B describe the complete UML diagram and examples on how this specification can be used in cooperation with an existing standard for device level network.

1.5 Proof of concept

The PIM and PSM described in this specification were reverse engineered and generalized from several existing standards as well as actual application systems. An example of these standards is described in

Appendix B of this document. The proposed model and interface are based on the systems that submitters have implemented or prototyped.

1.6 Relation to OMG Specification

This specification (re)uses two interfaces defined in the COS notification service for defining the SDO monitoring interface in the CORBA PSM (see Sections 4.4.5 and 5).

The XML Metadata Interchange (XMI) specification could be used to encode the SDO resource data model in XML and interchange the data in SDO systems. It is left to implementations how to use XMI in SDO systems.

The COS relationship service might be used to specify the Organization interface defined in the proposed CORBA PSM. The submitters decided not to use the service because its scope is too broad for our purpose (see Section 3.3.7); it defines a lot of constructs the submitters do not need to define Organization.

1.7 Relation to Pending OMG Specification

The PIM and PSM for SDO address the needs of basic data model and common interfaces for logical representation of hardware devices and software components. Since the proposed PIM is described with UML, the UML version 2.0 revision work, which has been conducted by the Analysis and Design Taskforce, is closely related to this specification.

1.8 Commercial availability

The Letter of Intents states companies' intentions regarding commercial availability of this submission.

2 Overview

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

2.1 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).

2.2 Objectives

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

Today, there are several resource interconnection technologies such as Universal Plug and Play, HAVi, OSGi, ECHONET and Jini. They are, however, restricted to specific platforms, network protocols and

programming languages, or they focus on limited application domains. No common model-based standards exist to handle various resources in a unified manner independently of underlying technologies and application domains. The objectives of this specification are to abstract the existing resource interconnection technologies into a higher layer, define their information and computational models in the layer, and make objects defined the models interoperable.

2.3 Resolution of RFP requirements and requests

2.3.1 Mandatory requirements

- *Proposals shall specify a resource data model for SDOs, which describes their capabilities and properties. This model shall identify all the necessary and relevant data to describe them and contain the corresponding data structures and relationships.*

This submission defines the data structure and their relationships describing SDOs. Each data structure defines relevant data to describe a SDO that represents hardware device or software component logically.

- *Proposals shall specify interfaces that are common to all SDOs to monitor and configure the resource data of SDOs.*

This submission defines the following five interfaces to monitor and configure the resource data of SDOs;

SDO interface

Configuration interface

Monitoring interface

SDOService interface

Organization interface

2.3.2 Optional requirements

- *Proposals may specify a model and interface for dynamic discovery of SDOs.*

This submission do not define a model nor an interface for dynamic discovery of SDOs

- *Proposals may specify a common interface for reserving an SDO's utilization in order to gain an exclusive access to it.*

This submission do not define an interface for reserving an SDO's utilization.

2.4 Responses to RFP issues to be discussed

This part will be completed in the final version.

- *Proposals should address how the OMG Trader Object Service is applicable in their CORBA platform specific models and other standard in each PSM, or at least shall justify precisely why the they are not used and describe their evaluation criteria.*

OMG Trader service is not discussed in this submission because this do not define a model nor an interface for dynamic discovery of SDOs.

2.5 Compliance

This specification consists of two parts, a Platform Independent Model (PIM) described by UML and a Platform Specific Model (PSM), specifying a realization of the PIM in the terms of CORBA IDL. Both parts have mandatory pieces (SDO interface and the data structure used by the SDO interface) and optional pieces. Conformant implementations must either provide an implementation which represents a complete mapping of the PIM into the selected target technology; or it must provide a complete implementation of the CORBA IDL PSM described in this document.

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

3 Platform Independent Model

This section specifies the PIM from information and computational viewpoint of SDOs. Section 3.2 describes the PIM for the resource data model, which is used to describe the capabilities and properties of SDOs (i.e., an information aspect of SDOs). Section 3.3 defines the interfaces to access and manipulate resource data (i.e., a computational aspect of SDOs). The proposed PIM is build based on the UML specification 1.4.

3.1 Overview of PIM for SDO

A SDO represents a hardware devices that may be accessed through existing standards or software component and provides information and interfaces for dynamic access by other applications. As described above, the proposed 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.

3.2 Resource Data Model

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

3.2.1 Overview of Resource Data Model

The resource data model includes the following constructs:

- Profiles
  - Device profile, which defines a set of hardware specific properties, (see Section 3.2.7),
  - Service profile, which defines a set of software specific properties, (see Section 3.2.8),
  - Configuration profile, which defines a set of properties to configure SDOs, (see Section 3.2.10),
- Status, which indicates the current status of SDOs (see Section 3.2.9), and
- Organization, which defines a relationship between/among the objects running in SDO system (see Section 3.2.5).

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

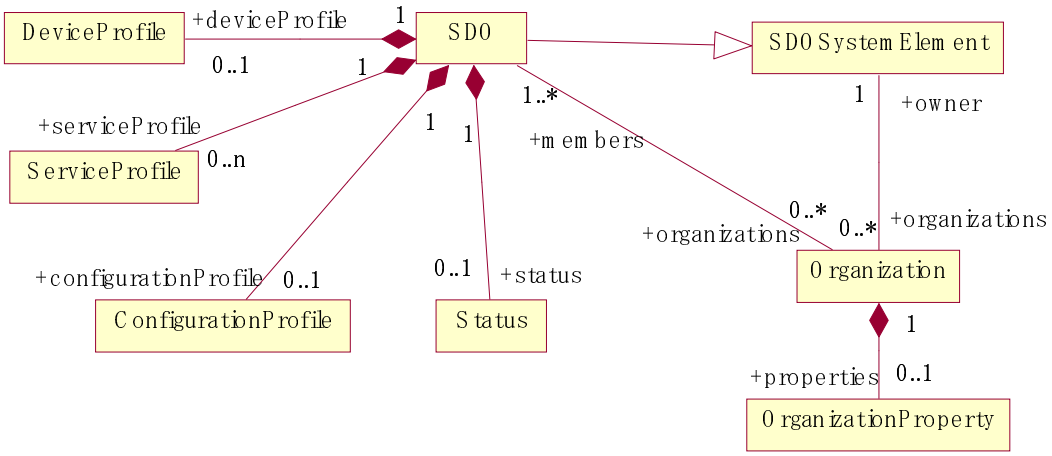
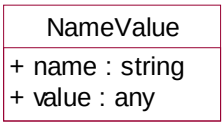
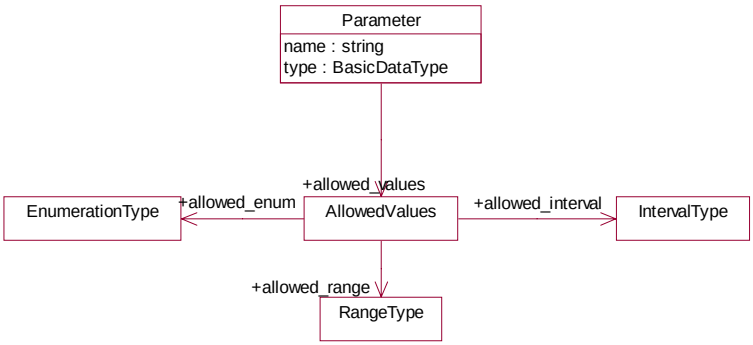


Figure 1. SDO resource data model

3.2.2 Data Structures used by Resource Data Model

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

Name	Description
StringList	A list of strings.
`UniqueIdentifier`	Identifier for the constructs in the resource data model (e.g. SDO). Each identifier is typed as a string and must be unique in a given domain of application deployment.[1]
`NameValue`	A pair of a name and its value.  **Figure 2. NameValue**
`NVList`	A list of `NameValues`.
Parameter	Data structure to define a variable (parameter) independently of implementation technologies. The `Parameter` structure defines the name of variable and the type of data contained in it.  **Figure 3. Parameter** Attributes defined in `Parameter`. ▪ name – parameter's name ▪ type – name of parameter's type. The original

[1] It is beyond the scope of this specification to define the format of identifiers and the algorithm to generate them, because they are implementation dependent. For example, some applications may use standardized schemes such as the IETF UUID [UUID00], others may use proprietary ones. Different SDO systems need to follow an agreed scheme for identifiers to maintain the interoperability between SDOs.

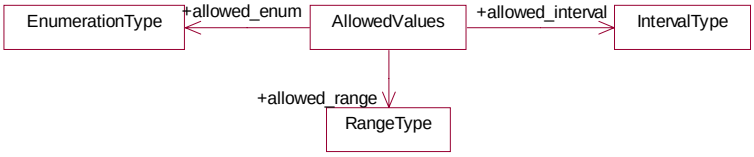
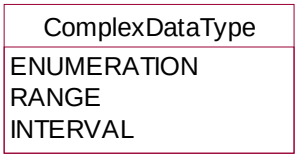
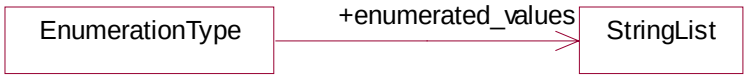
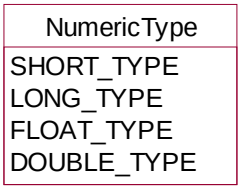
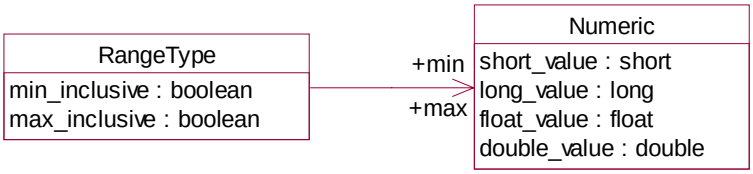
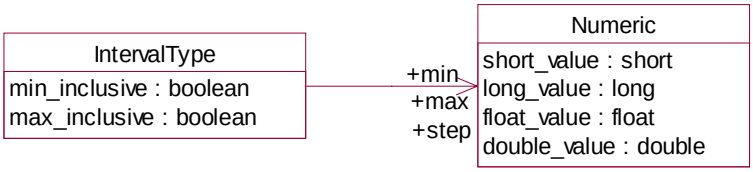
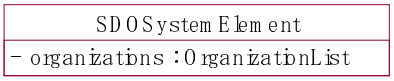
	value scope of parameter data type can be constrained by definitions allocated in the attribute allowedValues. ▪ allowedValues - Values that the parameter can accept. This attribute is used only when the original scope of the parameter's data type must be constrained. For example, values allowed for a string parameter can be defined by an enumeration, or values allowed for a parameter of numeric type can be constrained by a range. The values allowed for a parameter can be defined in enumeration, range, or interval structures. The value of attribute allowedValues is null if there are no constraints on parameter values, that is, any value from the original scope of parameter's type can be assigned to the parameter.
BasicDataType	Data structure representing the basic data types that can be used to declare a `Parameter`'s type (see Figure 3). Typically, basic data types include integer, floating point numbers and string. Available basic data types depend on the underlying implementation technology used (e.g., programming language or middleware technology). In the CORBA PSM in this specification, `BasicDataType` is `typedef`-ed with `CORBA::TCKind` (See Sections 4.2 and 5).
AllowedValues	 **Figure 4. Structure AllowedValues** Data structure that specifies the constraint applied to the original scope of parameter's data type. Depending on data type chosen for this structure, an enumeration, range, or interval can be specified, using EnumerationType, RangeType, or IntervalType structures, respectively (see below).
ComplexDataType	

	Figure 5. ComplexDataType ComplexDataType enumerates types of structures that can be used in the attribute allowedValues to define constraints on original data type of parameter. The structures that can be used to specify enumeration, range or interval are defined below.
EnumerationType	 **Figure 6. EnumerationType** The enumeration of values that can be assigned to a `Parameter`. The enumerated values are always of string type. ▪ enumeratedValues – a list of string values that a `Parameter` can contain. Example: {“red”, “blue”, “white”}
NumericType	 **Figure 7. NumericType** This enumeration is used by structures RangeType and IntervalType. It defines numeric types allowed to be used to specify bounds of intervals or ranges, and step between interval values.
RangeType	 **Figure 8. RangeType** Data structure representing the values that can be assigned to a `Parameter` defined as range. ▪ min – the lower bound of the range

	▪ max – the upper bound of the range ▪ minInclusive – a boolean value showing if the lower bound value is included in the range ▪ maxInclusive – a boolean value showing if the upper bound value is included in the range Example: the range ((int) 0, (int) 20, false, true) defines values {1,2,…20}.
IntervalType	 **Figure 9. IntervalType** Data structure representing the values that can be assigned to a `Parameter` defined as interval. ▪ min – the lower bound of the interval ▪ max – the upper bound of the interval ▪ minInclusive – a boolean value showing if the lower bound value is included in the range ▪ maxInclusive – a boolean value showing if the upper bound value is included in the range ▪ step – the step between the values in the interval. Example: the interval ((int) 0, (int) 20, false, true, 5) defines values {5,10,15,,20}.
ParameterList	A list of `Parameter` structures.

3.2.3 SDOSystemElement



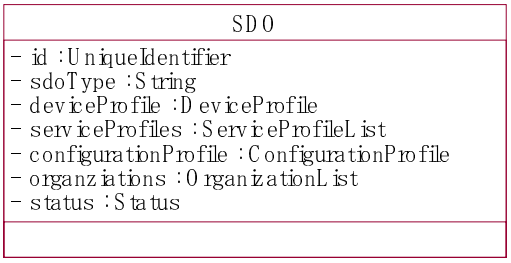
`SDOSystemElement` is the base class of the classes that represent SDO system's elements. The examples of the system's elements include SDOs and non-SDOs (e.g., user and location). SDO (a subclass of `SDOSystemElement`) is described in Secton 3.2.3. An example of non-SDOs is described in Section 3.2.11. `SDOSystemElement` is used to indicate that its subclasses represent SDO system's elements.

<Attributes>

Attribute	Type	Description
organizations	OrganizationList	A list of Organizations that SDOSystemElement has.

3.2.4 SDO

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



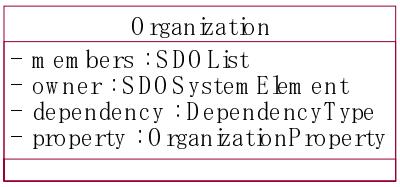
<Attributes>

Attribute	Type	Description
id	UniqueIdentifier	Unique identifier for an SDO uniquely.
sdoType	String	Descriptive information of an SDO.
deviceProfile	DeviceProfile	A device profile that an SDO has. Null is assigned when an SDO does not have its device profile. See Section 3.2.7 for more details about device profile.
serviceProfile	ServiceProfile	A list of service profiles that an SDO has. Null is assigned when an SDO does not have any service profile. See Section 3.2.8 for more details about service profile.
configurationProfile	CongigurationProfile	A configuration profile that an SDO has. Null is assigned when an SDO does not have its configuration profile.See Section 3.2.10 for more details about configuration profile.
status	Status	A status information of

		an SDO. Null is assigned when no status information is available. See Section 3.2.9 for more details about status.
`organizations`	`OrganizationList`	A list of references to the `Organization`s that an SDO keeps with other `SDO`s or `SDOSystemElement`. Null is assigned when an SDO has no relationship with other `SDO`s or `SDOSystemElement`. See Section 3.2.5 for more details about `organization`.

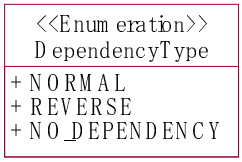
3.2.5 Organization

`Organization` represents a relationship between/among `SDOSystemElement`s. An Organization can be established among different SDOs, and 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 3.2.6).



<Attributes>

Attribute	Type	Description
`members`	`SDOList`	A list of SDOs that are the members associated with the Organization.
`owner`	`SDOSytemElement`	The owner of the Organization. It can be an SDO or non-SDO (see below for more detail).
`dependency`	`DependencyType`	Indicates if one side of an `Organization` depends on the other side. If the `Organization` represents dependency relationship, it

		also indicates which side depends on which side. This attribute is typed as an enumeration. `DependencyType` is an enumeration of three forms of dependency (Figure 10).  **Figure 10. DependencyType** See more details below.
`property`	`OrganizationProperty`	Property of the `Organization`. `OrganizationProperty` is described in Section 3.2.6.

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

(1) Hierarchical organization, which indicates `owner` supervises `members`. In this case, `DependencyType` should hold `NORMAL` value (see Figure 10).

(2) Reversely hierarchical organization, which indicates `members` supervise `owner`. In this case, `DependencyType` should hold `REVERSE` value (see Figure 10).

(3) Flat organization, which indicates no dependency exists. In this case, `DependencyType` should hold `NO_DEPENDENCY` value (see Figure 10).

Both an SDO and an non-SDO (i.e. any instance of the subclasses specializing `SDOSystemElement)` can be assigned to `owner` of an `Orgnaization`. When an SDO is an `owner`, Organization can represent any of the above three topology patterns.

- When an Organization represent topology pattern (1), an SDO (`owner`) composes one or more SDOs (`members`). For example, air conditioner (owner) composes a temperature sensor (member), humidity sensor (member) and wind flow controller (member).

- When an Organization represent topology pattern (2), multiple SDOs(members) share an SDO (owner). For example, an amplifier (owner) is shared by several AV components (members) in an AV stereo.

When a non-SDO is an `owner`, examples of the topology are as follows.

- User (owner)-SDO (members): When a user (owner) supervises one or more SDOs (members), the `organization` represents topology pattern (1).

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

3.2.6 OrganizationProperty

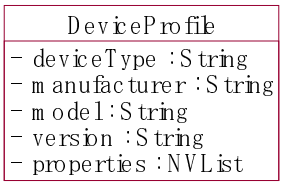
`OrganizationProperty` contains the properties of an `Organization`.

<Attributes>

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

3.2.7 DeviceProfile

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

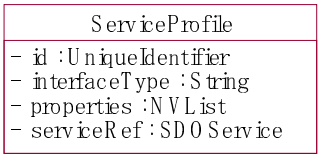


<Attributes>

Attribute	Type	Description
`deviceType`	`String`	General type name of a device. This attribute describes general kind of devices to categorize them and specify

		fundamental capability of the device.
`manufacturer`	`String`	Identifier for the manufacturer of a device.
`model`	`String`	Model name of a device.
`version`	`String`	Version number of a device.
`properties`	`NVList`	Device specific properties in addition to the above four ones. The property values contained in this attribute are implementation dependent.

3.2.8 ServiceProfile



`ServiceProfile` defines a set of properties of the function provided by a device or software component that an SDO represents. The function is implemented by another object that `serviceRef` referes to (see also Section 3.3.5).

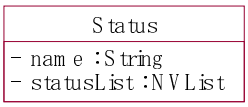
For example, an air conditioner has a function (a capability of service) for "fixing room temperature", and has control interfaces (referred by `ServiceRef`) for operations like "set a room's temperature", "set operation mode (heating/cooling/dehumidifying)", "turn on/off power", and so on.

<Attributes>

Attribute	Type	Description
`id`	`UniqueIdentifier`	Identifier for a service (or function) that an SDO provides. An SDO can provide one or more services (functions), and `id` is used to distinguish different services (functions).
`interfaceType`	`String`	The type of the interface through which an SDO provides its service (function). The scheme to describe the interface type depends on underlying implementation technologies. In the proposed CORBA PSM, this attribute contains the repository ID of the IDL interface through

		which an SDO's service (function) is provided.
properties	NVList	List of properties specific to the service (function) that an SDO provides. The property values contained in this attribute are implementation dependent.
serviceRef	SDOService	Reference to the object that provides the service (function) represented by this profile.

3.2.9 Status



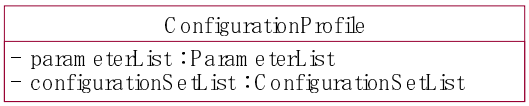
`Status` contains the current status of an SDO. It contains the name of the kind of status, and a set of status described by a pair of name and value for each status values. The current availability (`name`) of an SDO with concrete status data (e.g., list of power on/off, activated/deactivated) is an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

<Attributes>

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

3.2.10 ConfigurationProfile

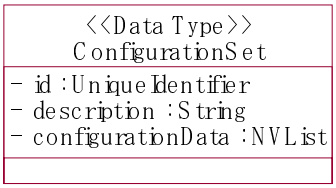
`ConfigurationProfile` contains a set of properties to configure an SDO.



<Attributes>

Attribute	Type	Description
`parameters`	`ParameterList`	A list of Parameter that represents the kinds of properties to configure an SDO.
`configurationSets`	`ConfigurationSetList`	A list of configurationSet that represents a set of properties with their values to configure an SDO.

- Data type definition: ConfigurationSet



Parameter defines the data type of a variable.

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

3.2.11 Examples of resource data model

This section shows several examples of SDO resource data defined in this specification in order to demonstrate how it can be used. Two different types of SDOs (Thermometer SDO and Airconditioner SDO) are described as example SDOs.
The other SDO, for example TemperatureController, gets the temperature of the room from the Thermometer SDO and controls the Airconditioner SDO.

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

DeviceProfile		
attribute	value	
deviceType	*Temperature Sensor*	
manufacturer	*TH310*	
model	*Thermo Inc.*	
version	*1*	
properties	name	value
	unit	*Celsius*
	rangeMin	*-50*
	rangeMax	*150*

SDO	
attribute	value
sdoID	*abc_12345*
sdoType	*EN_TemperatureSensor*

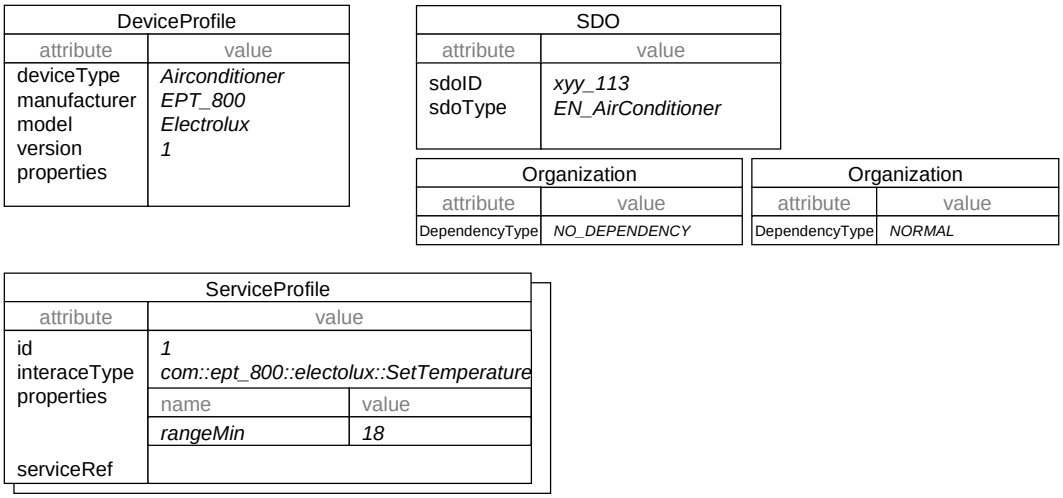
Status		
attribute	value	
name	*devicestatus*	
statusList	name	value
	thermoValue	*30*

Thermometer SDO

The `DeviseProfile` 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. *thermovalue* holds the current temperature (30 degrees) of the room wherever it is located. This value 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 `SDOServiceProfile`.

(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`, a `Organization` and two `ServiceProfiles`.



Airconditioner SDO

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

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

3.3 Interfaces

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

3.3.1 Overview of Interfaces

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

- `SDOInterface`, which defines a series of operations to obtain SDO's properties and the other interfaces that the SDO implements (`MonitoringInterface`, `ConfigurationInterface` and `SDOService`). All the SDOs must implement this interface.

- `MonitoringInterface`, which defines the operations to monitor changes in SDO's properties. Every SDO does not implement this interface. `NotificationCallbackInterface` is also defined as a call back interface of notification subscribed by using `MonitoringInterface`. Every SDO does not implement this interface.

- `ConfigurationInterface`, which defines the operations to configure SDO's profiles (e.g., device, service and configuration profiles) and Organizations associated with the SDO. Every SDO does not implement this interface.

- `SDOService`, which abstracts the service provided by an SDO. Every SDO does not implement this interface.

- `OrganizationInterface`, which defines the operations to establish and maintain `Organization`s. All the `Organization`s must implement this interface.

3.3.2 Data Structures used by Interfaces

3.3.2.1 SDOException

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

<Attributes>

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

List of Exception types

- `SDONotExists`

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

- `NotAvailable`

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

- `InvalidParameter`

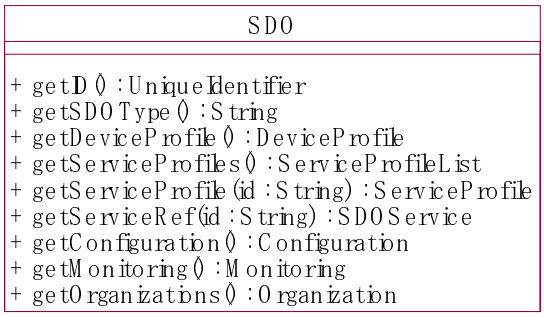
  This exception is raised when a parameter(s) specified in an operation call is invalid. For example, it may be raised when null is assigned in a parameter that should contain a value.

- `InterfaceNotImplemented`

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

3.3.3 SDO interface

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



(1) + getID ( ) : String

This operation returns `id` of the SDO .

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

Exceptions

This operation throws `SDOException` with the following type.

- `InvalidPrameter` type SDOException if the parameter `id` is null.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `NotAvailable` type SDOException if there is no response from the target SDO.

(2) - + getSDOType ( ) : String

This operation returns sdoType of the SDO.

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

Exceptions

This operation throws SDOException with the following type.

- `InvalidParameter` type SDOException if the parameter `sdoType` is null.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `NotAvailable` type SDOException if there is no response from the target SDO.

(3) + getDeviceProfile ( ) : DeviceProfile

This operation returns `deviceProfile` that the SDO has.

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

Exceptions

This operation throws `SDOException` with the following type.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `NotAvailable` type SDOException if there is no response from the target SDO.

(4) + getServiceProfiles ( ) : ServiceProfileList

This operation returns a list of the `serviceProfiles` which the SDO has.

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

Exceptions

This operation throws SDOException with the following type.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `NotAvailable` type SDOException if there is no response from the target SDO.

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

This operation returns a `serviceProfile` which is specified by the argument `id`.

Parameter	Type	Description
id	UniqueIdentifier	The identifier referring to one of the `serviceProfile`.

<return>	ServiceProfiles	The profile of the specified service.

Exceptions

This operation throws SDOException with the following type.

- `InvalidParameter` type SDOException if the `serviceProfile` which is specified by argument id does not exist.

- `SDONotExsits` type SDOException if the target SDO does not exists.

- `NotAvilable` type SDOException if there is no response from the target SDO.

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

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

Parameter	Type	Description
id	UniqueIdentifier	The identifier referring to a `serviceProfile` which represents the SDO service.
<return>	SDOService	The service reference of the specified service.

Exceptions

This operation throws SDOException with the following type.

- `InvalidParameter` type SDOException if argument "id" is null, or if the ServiceProfile which is specified by argument "id" does not exist.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `NotAvailable` type SDOException if there is no response from the target SDO.

(7) + getConfiguration () : Configuration

Configuration is one of SDO’s interface in order to configure its structure composed by DeviceProfile, ServiceProfile and Organization. This operation is used to get the Configuration interface.

Parameter	Type	Description
<return>	Configuration	Reference to Configuration interface with regard to the SDO.

Exceptions

This operation throws SDOException with the following type.

- `InterfaceNotImplemented` type SDOException if the target SDO has no Configuration interface.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `NotAvailable` type SDOException if there is no response from the target SDO.

(8) + getMonitoring ( ) : Monitoring

Monitoring is one of SDO's interface to monitor the various properties or parameters of an SDO. This operation is used to get the interface in order to monitor such properties or parameters of foreign SDO.

Parameter	Type	Description
<return>	Monitoring	Reference to Monitoring interface with regard to the SDO.

Exceptions

This operation throws SDOException with the following type.

- `InterfaceNotImplemented` type SDOException if the target SDO has no Monitoring interface.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `NotAvailable` type SDOException if there is no response from the target SDO.

(9) + getOrganizations ( ) : OrganizationList

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

Parameter	Type	Description

<return>	OrganizationList	Returns the list of Organizations that the SDO belong to.

Exceptions

This operation throws SDOException with the following type.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `NotAvailable` type SDOException if there is no response from the target SDO.

3.3.3.1 Usage: SDO interface

The section describes examples about how to use the operations to get SDO parameters (ID and DO Type), profiles (DeviceProfile and ServiceProfile) and invoke SDO interfaces (Configuration and Monitoring).

As an example, the operation to get SDO parameter `id` is shown in Figure 11.

In figure 11, sdo1 makes requests to sdo2 to acquire the parameter. The example of the operation to get other interfaces by using SDO interface will be described in section 3.3.6.

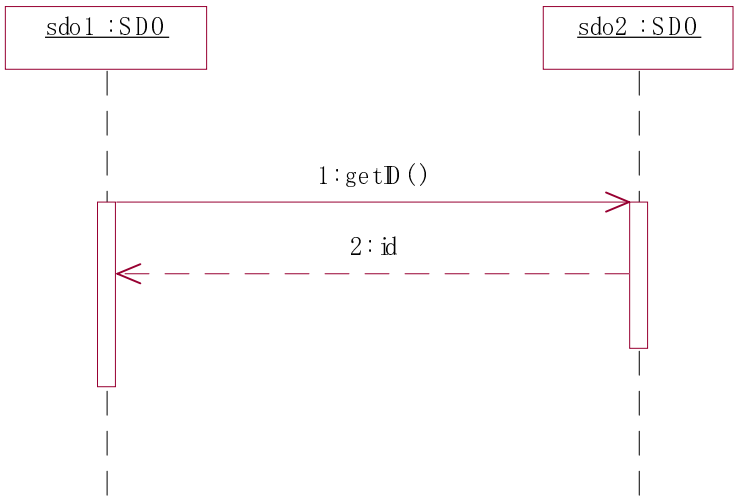


Figure 11. Sequence Chart: SDO operation concern with data resouce

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

Message 2: sdo2 returns description of identifier.

3.3.4 Configuration interface

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

<Operations>

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

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

Parameter	Type	Description
none		

Exceptions

This operation throws SDOException with the following type.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `NotAvailable` type SDOException if there is no response from the target SDO.

- `InvalidParameter` type SDOException if argument "dProfile" is null, or if the object which is specified by argument "dProfile" does not exist.

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

This operation adds ServiceProfile to the target SDO which 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 add the ServiceProfile if not exist, or overwrite if exist.

Parameter	Type	Description

sProfile	ServiceProfile	ServiceProfile to be added.

Exceptions

This operation throws SDOException with the following type.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `InvalidParameter` type SDOException if argument "sProfile" is null, or if the object which is specified by argument "sProfile" does not exist.

- `NotAvailable` type SDOException if there is no response from the target SDO.

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

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

Parameter	Type	Description
organization	Organization	Organization to be added.

Exceptions

This operation throws SDOException with the following type.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `InvalidParameter` type SDOException if argument "organization" is null, or if the object which is specified by argument "organization" does not exist.

- `NotAvailable` type SDOException if there is no response from the target SDO.

(4) + removeDeviceProfile ( ) : void

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

Parameter	Type	Description
none		

Exceptions

This operation throws SDOException with the following type.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `NotAvailable` type SDOException if there is no response from the target SDO.

- `InvalidParameter` type SDOException if argument "dProfile" is null, or if the object which is specified by argument "dProfile" does not exist.

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

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

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

Exceptions

This operation throws SDOException with the following type.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `InvalidParameter` type SDOException if argument "sProfile" is null, or if the object which is specified by argument "sProfile" does not exist.

- `NotAvailable` type SDOException if there is no response from the target SDO.

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

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

Parameter	Type	Description
organization	Organization	Organization to be removed.

Exceptions

This operation throws SDOException with the following type.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `InvalidParameter` type SDOException if argument "organization" is null, or the object which is specified by argument "organization" does not exist.

- `NotAvailable` type SDOException if there is no response from the target SDO.

(7) getConfigParameter () : ParameterList

This operation returns a list of Parameters.

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

Exceptions

This operation throws SDOException with the following type.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `InvalidReturnValue` type SDOException if the target SDO has no Parameter.

- `NotAvailable` type SDOException if there is no response from the target SDO.

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

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

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

Exceptions

This operation throws SDOException with the following type.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `InvalidParameter` type SDOException if the parameter which is specified by argument "name" has no value, or if argument "name" is null, or if the parameter which is specified by argument "name" does not exist.

- `NotAvailable` type SDOException if there is no response from the target SDO.

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

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

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

Exceptions

This operation throws SDOException with the following type.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `InvalidParameter` type SDOException if Arguments ("sdoID"and / or "name" and / or "value") is null, or if the parameter which is specified by arguments ("sdoID" and / or "name") does not exist.

- `NotAvailable` type SDOException if there is no response from the target SDO.

(10) + getConfigurationSets () : ConfigurationSetList

This operation returns a list of ConfigurationSets which the ConfigurationProfile has.

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

Exceptions

This operation throws SDOException with the following type.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `InvalidParameter` type SDOException if ConfigurationSet is null.

- `NotAvailable` type SDOException if there is no response from the target SDO.

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

This operation adds a ConfigurationSet to the ConfigurationProfile.

Parameter	Type	Description
configurationSet	ConfigurationSet	The ConfigurationSet which is added.

Exceptions

This operation throws SDOException with the following type.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `InvalidParameter` type SDOException if Arguments ("configurationSet") is null, or if the object which is specified by arguments ("configurationSet") does not exist.

- `NotAvailable` type SDOException if there is no response from the target SDO.

(12) + removeConfigurationSet (configurationSet : ConfigurationSet) : void

This operation removes a ConfigurationSet from the ConfigurationProfile.

Parameter	Type	Description
configurationSet	ConfigurationSet	The ConfigurationSet which is removed.

Exceptions

This operation throws SDOException with the following type.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `InvalidParameter` type SDOException if Arguments ("configurationSet") is null, or if the object which is specified by arguments ("configurationSet") does not exist.

- `NotAvailable` type SDOException if there is no response from the target SDO.

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

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

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

Exceptions

This operation throws SDOException with the following type.

- `SDONotExists` type SDOException if the target SDO does not exists.

- `InvalidParameter` type SDOException if Arguments ("`configID`") is null.

- `NotAvailable` type SDOException if there is no response from the target SDO.

3.3.4.1 Usage: Configuration

As an example, the sequence of Profile acquisition is shown in Figure 12. And, as an example of parameter acquisition, the sequence of "getConfigParameter()" is shown in Figure 13. First, the configuring SDO (sdo1) get the Configuration object by using of the operations, from the SDO which is configured (sdo2). And the sequences of the other operations which belong to the Configuration interface as shown in Figure 12 and 13.

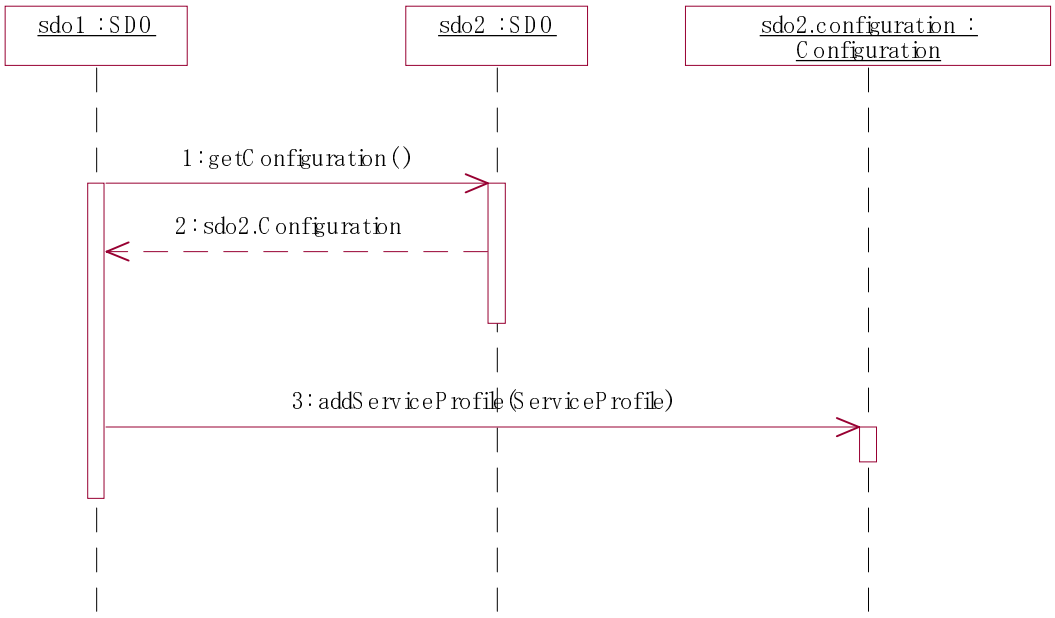


Figure 12. Sequence Chart: Configuration

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

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

Message 3: sdo1 adds the ServiceProfile object to sdo2.

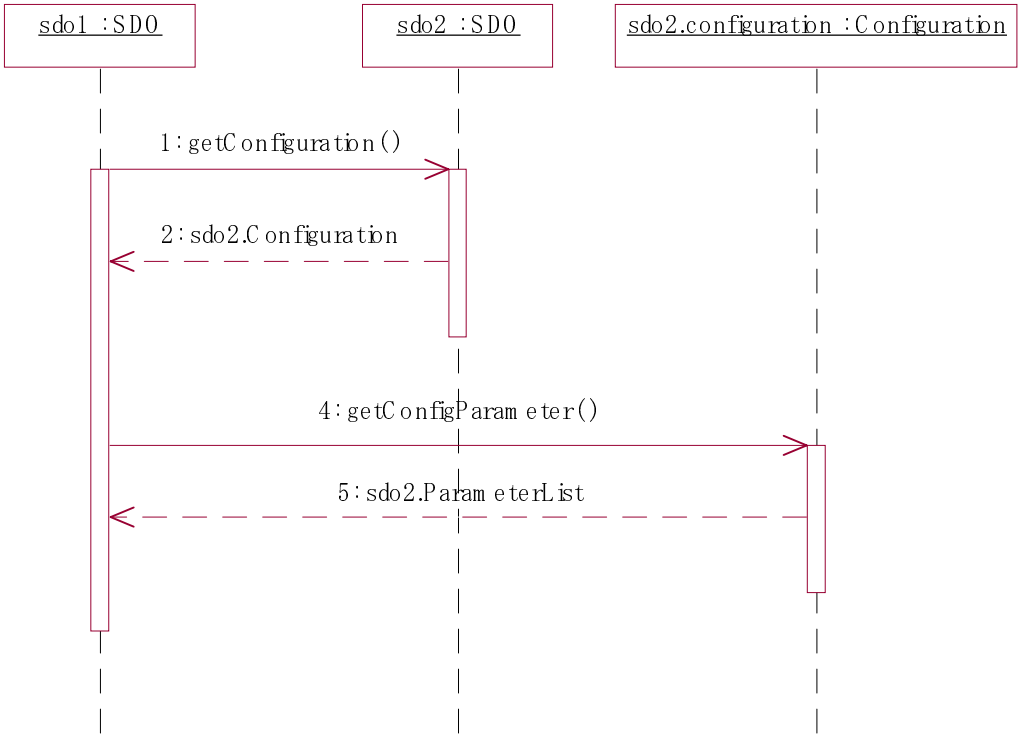


Figure 13. Sequence Chart: Configuration

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

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

Message 4: sdo1 requests parameter list of sdo2.

Message 5: sdo2 returns the parameters as list of names and values of properties represented as NVList object.

3.3.5 SDOService interface

SDOService interface provides operations for the services which are provided by functions of a SDO.

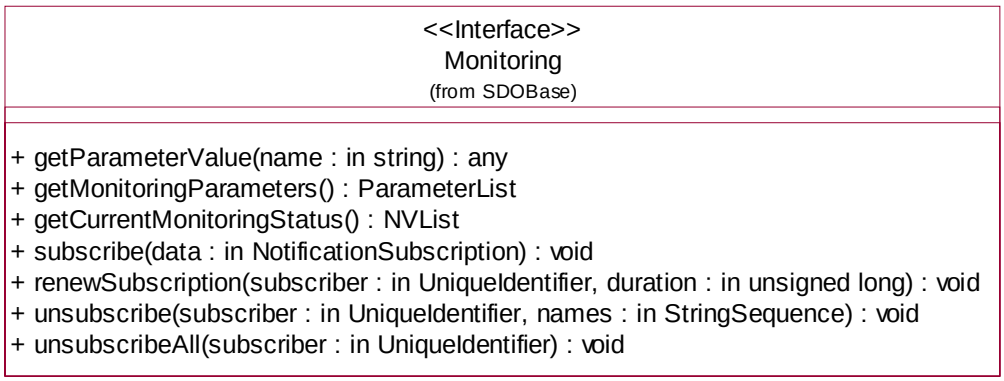
The interface of a service differs for every service. So, this interface does not have the fixed model.

3.3.6 Monitoring interface

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

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

The properties of an SDO can be monitored mainly in two different ways: by *polling* and by *subscription*.
Polling is the simpler way of monitoring. The observer requests the current values of the properties it is interested in. The SDO that wants to monitor one or more properties must send a request message to the particular SDO. The Monitoring interface provide functions (getCurrentStatus(), and getParameterValue()) that support the monitoring by polling. The monitoring by polling is described detailed in section 0.
Using *subscription* an observing SDO is notified about changes of monitored properties. The observing SDO has to subscribe to an SDO which is to be monitored. According to the subscription the monitored SDO notifies the subscriber using its Call-back interface (see section 3.3.6.6). The Monitoring interface supports the subscription through appropriate functions (subscribe(), renewSubscription(), unsubscribe()). The monitoring by subscription is described detailed in section 0.



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.

3.3.6.1 Data Structures Defined for Monitoring Interface

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

Name	**Short Description**
NotificationMode	Defines possible notification modes used in the subscription.
NotificationSubscription	This structure outlines the details of the subscription message describing what properties are being subscribed and the condition for the subscription.

Table 1. Data Structures required for Monitoring Interface

NotificationMode	Possible notification modes while subscribing to monitoring properties.

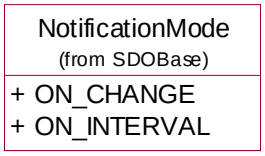
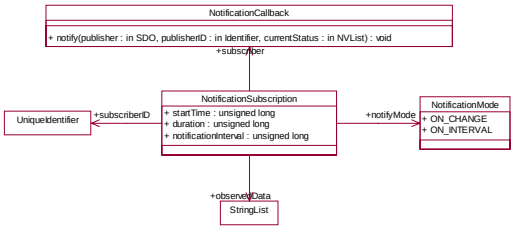


Figure 14. Notification Mode

ON_CHANGE - To be notified of the

	subscribed monitoring property every time the value of the parameter changes. ON_INTERVAL - To be notified of the subscribed monitoring property on the specified interval of time. That is, if a subscription to some property is made in this mode, the notification is sent to the subscribing SDO only at the specified time with the current value.
NotificationSubscription	 **Figure 15. NotificationSubscription Data Structure** 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. ▪ subscriber - The address or reference of the object that will receive notification messages; the object referenced here must implement the interface NotificationCallback. The value of this field depends on basis technology of SDO system. Please read section 3.3.6.6 for more details. ▪ subscriberID – The unique identifier of the SDO that subscribes to this property. ▪ notifyMode - The mode of notification (*On Change* or *On Interval*). The notifications are sent either when the value of at least one of subscribed monitoring properties changes (notification on change), or periodically. In this case the frequency of

	notifications is specified with the attribute notificationInterval. ▪ observedData - List of names of monitoring properties to be subscribed. ▪ startTime – Defines the time period (in milliseconds) at which monitoring of the properties should start. If it is not specified, then the subscription will be activated right after receiving the subscription message. ▪ duration - Indicates for how long (in milliseconds) the subscription should be last. ▪ notificationInterval – Time interval (in milliseconds) in which the notification is sent to the subscribing SDO, if the NotificationMode of the subscription is ON_INTERVAL.

Table 2. Detail Description of Data Structures required for Monitoring Interface

3.3.6.2 Operations provided by Monitoring Interface

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

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

unsubscribeAll (subscriber : UniqueIdentifier)	This operation unsubscribes all the subscribed properties of the specified SDO.

Table 2. Operations provided by Monitoring Interface

Detailed description of all operations

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

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

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

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

Exceptions

This operation throws `SDOException` with the following type.

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

(2) +getMonitoringParameters () : ParameterList

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

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

Exceptions

This operation throws `SDOException` with the following type.

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

(3) +getCurrentMonitoringStatus () : NVList

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

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

Exceptions

This operation throws `SDOException` with the following type.

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

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

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

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

Exceptions

This operation throws `SDOException` with the following type.

- `InvalidParameter` type SDOException if the condition specified for the subscription is not valid. E.g.: 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.subscribedData do not exist. This exception is thrown even if one of the defined properties does not exist. In this case, the name of non-existing property must be specified in the exception data.
- `NotAvailable` type SDOException if there is no response from the target SDO.

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

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

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

Exception

This operation throws `SDOException` with the following type.

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

(6) +unsubscribe (subscriber : UniqueIdentifier, names : StringSequence)

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

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

Exception

This operation throws `SDOException` with the following type.

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

(7) +unsubscribeAll (subscriber : UniqueIdentifier)

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

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

Exception
This operation throws `SDOException` with the following type.
- `InvalidParameter` type SDOException if there are no subscriptions from the observer SDO defined by the parameter *subscriber* to unsubscribe.
- `NotAvailable` type SDOException if there is no response from the target SDO.

3.3.6.3 Usage: Monitoring Interface

Monitoring by Polling

SDOs can get information on status of other SDO without subscription to event notifications as well. The 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 16. In the diagram shown in the picture, sdo1 makes requests to sdo2 to acquire its status.

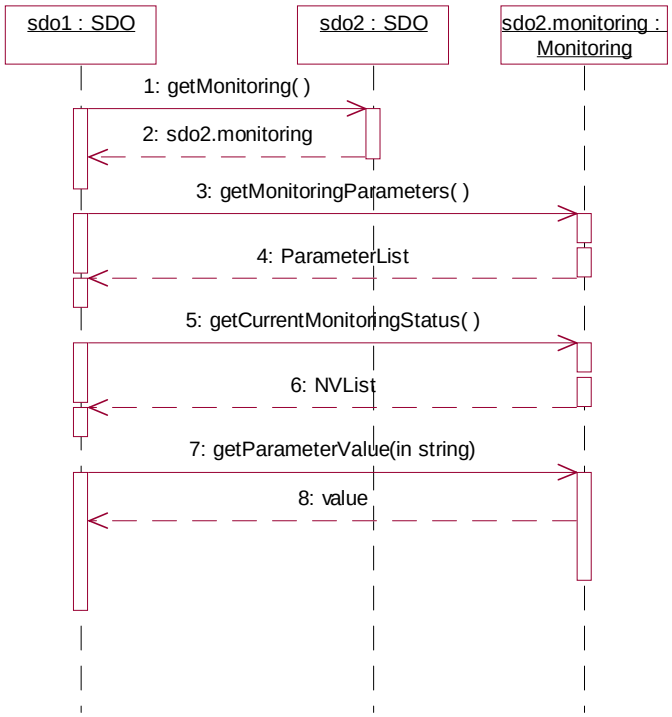


Figure 16. Sequence Chart: Monitoring

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 status of the sdo2.
Message 6: sdo2 returns the current status 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.

Monitoring by Subscription

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

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

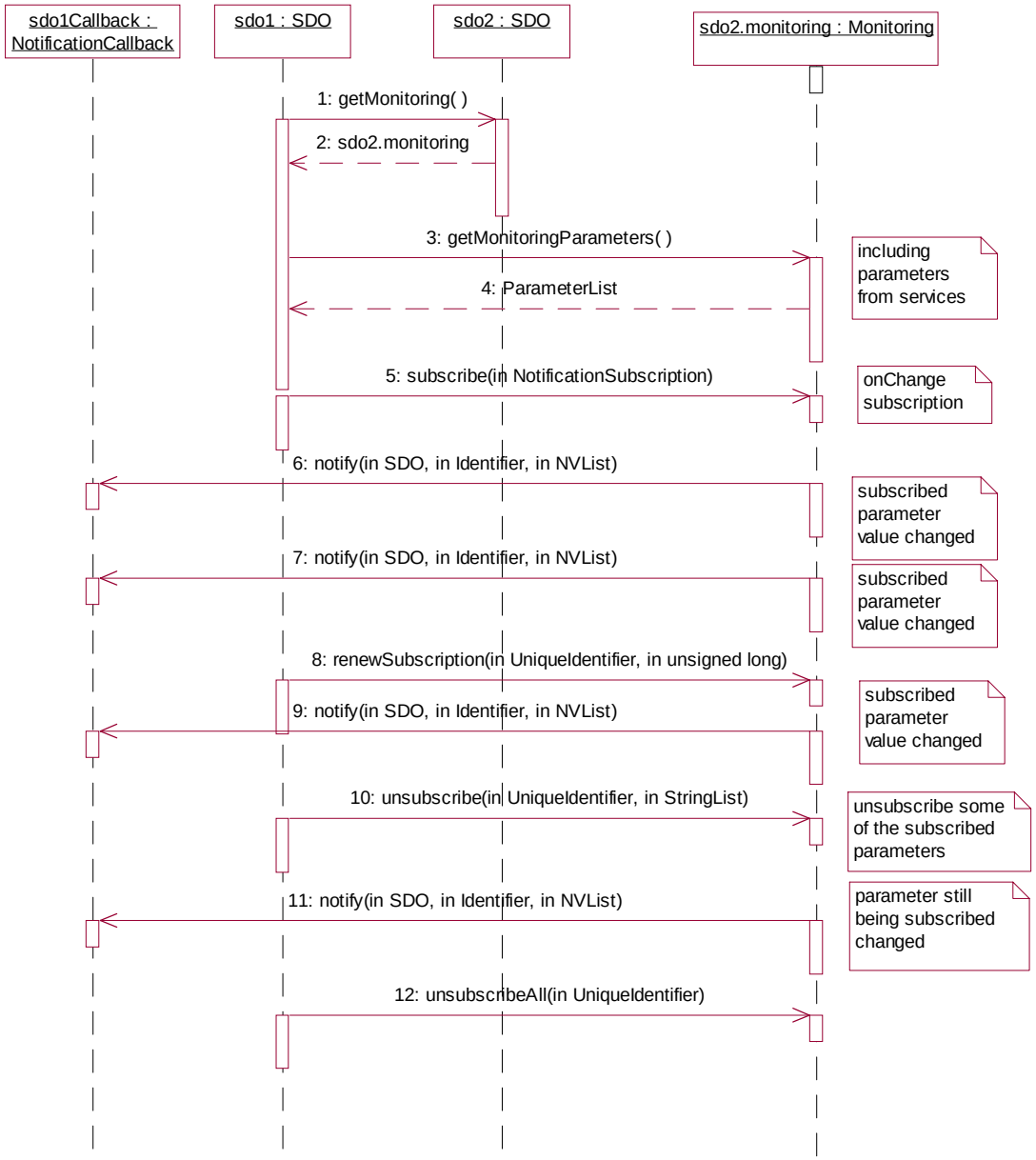
- when the value of the monitored properties change (*on_change* mode) , or
- on certain time interval (*on_interval* mode).

When subscribing, an SDO has to specify the mode and the properties it wants to monitor. Once subscribed the observing SDO (subscriber) is notified based upon the conditions specified. Since the subscriber is notified it must provide a callback interface, which is described in section 3.3.6.6.

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.

3.3.6.4 ON_CHANGE mode

This section described in detail the monitoring by subscription using ON_CHANGE mode. Monitoring ON_CHANGE basis is useful if the subscribing SDO just wants it to be notified as soon as one of the subscribed property value has changed. For example if the TemperatureController SDO 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.



j

Figure 17. Subscription and Notification On Change

The subscription of properties is shown in the sequence diagram (Figure 17). The interaction depicted in Figure 17 occurs between two SDOs, sdo1 and sdo2. In the 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
- The 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: short 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.

3.3.6.5 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 the property's value does not change).

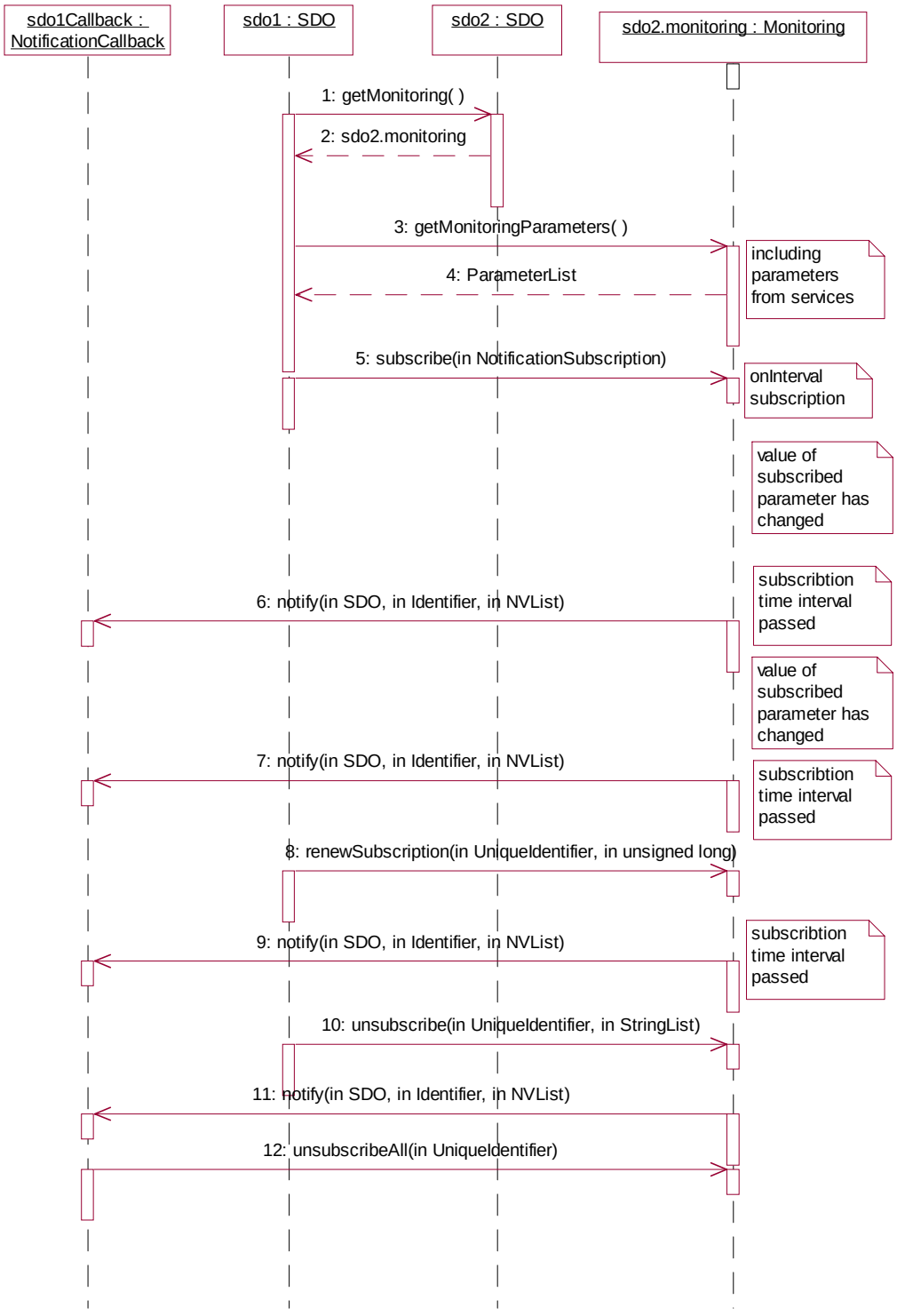
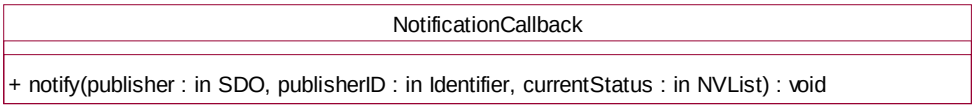


Figure 18. Subscription and Notification On Interval

Figure 18 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 17 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 18 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 18 between notification messages 10 and 12.

3.3.6.6 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.



3.3.6.7 Data Structures Defined for NotificationCallback Interface

Two general data structures (UniqueIdentifier and NVList) are used in the NotificationCallback interface. Please see section 3.3.6.1 for their detail description.

3.3.6.8 Operations provided by NotificationCallback Interface

(1) +notify(publisherID : Identifier, currentStatus : NVList)
This operation notifies the subscriber that either the value of the property has changed or the notification interval has elapsed.

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

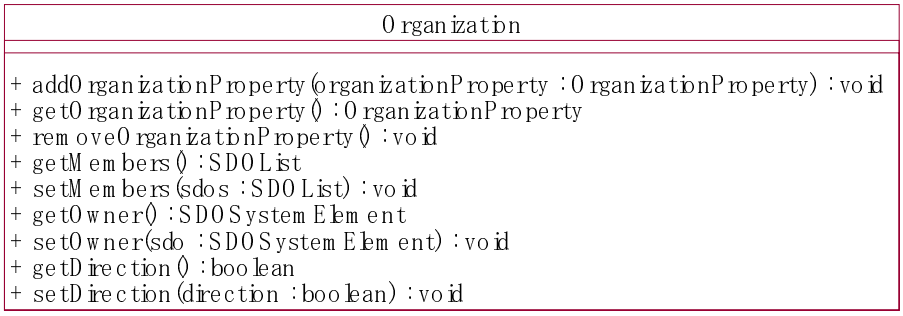
currentStatus	NVList	A list containing the properties and their current values. Please note that this list may not contain all the properties that an SDO has been subscribed to, probably because not all property have changed or because the notification interval of some properties has not elapsed yet.

3.3.6.9 Usage: NotificationCallback Interface

The examples of usage of operations of NotificationCallback interface are shown in Figure 17 and Figure 18. The operations are used to convey the changes in values of monitoring properties to notification subscriber.

3.3.7 Organization interface

The Organization interface is used to manage the Organization attribute.



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

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

Parameter	Type	Description
organizationProperty	OrganizationProperty	The type of organization to be

		added.

Exception

This operation throws SDOException with the following type.

- `InvalidParameter` type SDOException if argument "organizationProperty" is null, or if The object which is specified by argument "organizationProperty" does not exist.

- `NotAvailable` type SDOException if there is no response from the target SDO.

(2) + removeOrganizationProperty () : void

This operation removes the OrganizationProperty from a Organization.

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

Exception

This operation throws SDOException with the following type.

- `InvalidParameter` type SDOException if the Organization which the target SDO belong to has no organizationProperty.

- `NotAvailable` type SDOException if there is no response from the target SDO.

(3) + getOrganizationProperty () : OrganizationProperty

This operation returns the OrganizationProperties that a Organization has. A Organization has zero or more OrganizationProperties.

Parameter	Type	Description
<return>	OrganizationPropertyList	The types of organization contained in it.

Exception

This operation throws SDOException with the following type.

- `InvalidParameter` type SDOException if argument "organizationProperty" is null, or if The object which is specified by argument "organizationProperty" does not exist.

- `NotAvailable` type SDOException if there is no response from the target SDO.

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

This operation sets SDOs as members of the organization. The SDOs to be set is specified by argument “sdos”.

Parameter	Type	Description
sdos	SDOList	The SDOs to be added.

Exception

This operation throws SDOException with the following type.

- `InvalidParameter` type SDOException if the Organization which the target SDO belong to has no Member.

- `NotAvailable` type SDOException if there is no response from the target SDO.

(5) + getMembers ( ) : SDOList

This operation returns the list of SDO which is members of the Organization.

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

Exception

This operation throws SDOException with the following type.

- `InvalidParameter` type SDOException if argument "SDOList" is null, or I if the object which is specified by "sdo" in argument "sdos" does not exist.

- `NotAvailable` type SDOException if there is no response from the target SDO.

(6) + getOwner ( ) : SDOSystemElement

This operation returns the SDOSystemElement which is owner of the Organization.

Parameter	Type	Description

<return>	SDOEnity	Reference of owner object.

Exception

This operation throws SDOException with the following type.

- `InvalidParameter` type SDOException if the target SDO has no Owner.

- `NotAvailable` type SDOException if there is no response from the target SDO.

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

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

Parameter	Type	Description
sdo	SDOSystemElement	Reference of owner object.

Exception

This operation throws SDOException with the following type.

- `InvalidParameter` type SDOException if argument "sdo" is null, or if the object which is specified by "sdo" in argument "sdos" does not exist.

- `NotAvailable` type SDOException if there is no response from the target SDO.

(8) + getDirection () : boolean

This operation gets the relationship direction of the Organization.

Parameter	Type	Description
<return>	boolean	direction.

Exception

This operation throws SDOException with the following type.

- `InvalidParameter` type SDOException if the value of “direction” is null.

- `NotAvailable` type SDOException if there is no response from the target SDO.

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

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

Parameter	Type	Description
direction	boolean	direction.

Exception

This operation throws SDOException with the following type.

- `InvalidParameter` type SDOException if argument "direction" is null.

- `NotAvailable` type SDOException if there is no response from the target SDO.

3.3.7.1 Usage: Organization

As an example, the sequence of “getOrganizationProperties()” is shown in Figure 19. First, the configuring SDO (sdo1) get the Organization object by using of the operation “getOrganization()”, from the SDO which is configured (sdo2).

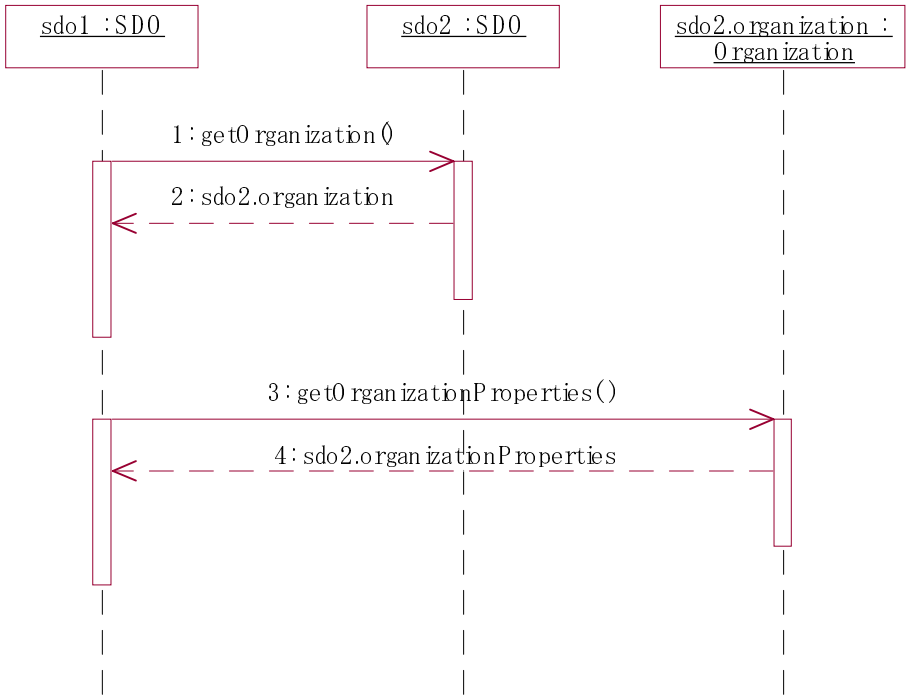


Figure 19. Complete UML diagram

Message 1: the configuring SDO (sdo1) gets the object implementing the Organization of configured SDO (sdo2).

Message 2: sdo2 returns the object sdo2.orgnanization that implements the Organization.

Message 3: sdo1 requests OrganizationProperty list of sdo2.organization.

Message 4: sdo2 sends response, containing the OrganizationProperty list of sdo2.organization.

4 Platform Specific Model: Mapping to CORBA IDL

This chapter introduces a CORBA specific model for the SDO PIM defined in chapter 3. The selected platform is CORBA version 3.0.
The SDO PIM defines the resource data model, interfaces, and necessary data structures for SDOs. In the PSM these interfaces and the data structures used in the individual methods are mapped to an according CORBA IDL specification. The complete IDL specification is presented in chapter 5.
An interface defined in the SDO PIM is mapped to a CORBA interface. An operation in a PIM interface is mapped to an CORBA operation. A private attribute in a PIM interface is mapped to an operation named get_<attribute name>. A public attribute in an interface is mapped to two operations; get_<attribute name> and set_<attribute name>. An PIM exception is mapped to a CORBA exception. The other data types in the SDO PIM (e.g. resource data) are mapped to the non-interface types in CORBA IDL. The proposed CORBA PSM is compliant with the IDL style guide (ab/98-06-03).

4.1 SDO Module

The interfaces and data structures defined in the CORBA PSM belong to module structure: org.omg.SDOPackage.

4.2 Data types used in CORBA PSM

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

```
typedef sequence<string>          StringList;
typedef sequence<SDO>             SDOList;
typedef sequence<Organization> OrganizationList;
typedef string                    UniqueIdentifier;
struct NameValue {
        string name;
        any value;
};
typedef sequence<NameValue>   NVList;
enum NumericType {
```

```
        SHORT_TYPE,
        LONG_TYPE,
        FLOAT_TYPE,
        DOUBLE_TYPE};
union Numeric switch (NumericType) {
        case SHORT_TYPE:  short short_value;
        case LONG_TYPE: long long_value;
        case FLOAT_TYPE: float float_value;
        case DOUBLE_TYPE: double double_value;
};
struct EnumerationType {
        StringList enumeration_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;
    };
    typedef sequence<OrganizationProperty> OrganizationPropertyList;
    struct DeviceProfile {
            string deviceType;
            string manufacturer;
            string model;
            string version;
            NVList properties;
    };
    struct ServiceProfile {
            string id;
            string interfaceType;
            NVList properties;
            SDOService serviceRef;
    };
typedef sequence<ServiceProfile> ServiceProfileList;

struct ConfigurationSet {
            string id;
            string description;
            NVList configurationData;
};
typedef sequence<ConfigurationSet> ConfigurationSetList;
```

4.3 Exceptions

The methods of SDO interfaces can raise SDOException. The kind of exception is defined in the attribute *type* (see section 3.3.2.1). 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, InvalidReturnValue, and NotFound.

```
#define exception_body { string description; }
...
exception NotAvailable                exception_body;
exception InterfaceNotImplemented     exception_body;
exception InvalidParameter            exception_body;
exception InvalidReturnValue          exception_body;
exception NotFound                    exception_body;
```

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

4.4 Interfaces

The SDO PIM defines several interfaces that can be implemented by an SDO. The SDO interface is a mandatory interface, whereas Configuration and Monitoring including NotificationCallback are optional interfaces. This means each SDO implementation has at least to implement the SDO interface and may additionally implement the other interfaces.
In the proposed CORBA model all interfaces as defined in the SDO PIM are directly mapped to CORBA interfaces. The IDL specification includes corresponding interface declarations. Additionally, all data structures used in the methods of these interfaces are also defined in the IDL specification.

The SDO IDL specification includes following interfaces declarations:

- interface SDOSystemElement
- interface SDO
- interface SDO Service
- interface Configuration
- interface Monitoring
- interface Organization

4.4.1 SDOSystemElement Interface

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

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

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

4.4.2 SDO Interface

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

```
interface SDO : SDOSystemElement {
  UniqueIdentifier get_id()
        raises (InvalidReturnValue, NotAvailable);
  string get_SDO_type()
        raises (InvalidReturnValue, NotAvailable);
  DeviceProfile get_device_profile ()
        raises (NotAvailable);
  ServiceProfileList get_service_profiles ()
        raises (NotAvailable);
  ServiceProfile get_service_profile (
        in string id
        ) raises (InvalidParameter, InvalidReturnValue, NotAvailable);
  SDOService get_service (
        in string id
        ) raises (InvalidParameter, InvalidReturnValue, NotAvailable);
```

```
    Configuration get_configuration ()
            raises (InterfaceNotImplemented, NotAvailable);
    Monitoring get_monitoring ()
            raises (InterfaceNotImplemented, NotAvailable);
};
```

4.4.3 Configuration Interface

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

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

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

4.4.4 SDOService

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

4.4.5 Monitoring Interface

The interface Monitoring in the PIM is mapped to an 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 3.3.6 are implemented using the OMG Notification Service [3].

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

To use the mechanisms defined in Notification Service, the Monitoring interface inherits the interfaces StructuredPushSupplier and StructuredPushConsumer, defined in CosNotifyComm module.
The interface **StructuredPushSupplier** enables SDOs to publish event notifications to the Notification Service event channel (“referred henceforth as the *notification channel*). This interface supports the behavior of objects that send Structured Events into the notification channel using push-style communication. (Models of event propagation are described in [4].) 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** covers the group of subscription operations defined for Monitoring interface in chapter 3.
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 chapter 3.

4.4.6 Organization Interface

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

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

```
    OrganizationProperty get_organization_property ()
            raises (NotAvailable);
    SDOList get_members ()
            raises (NotAvailable);
    void set_members (
            in SDOList sdos
            ) raises (InvalidParameter, NotAvailable);
    SDOSystemElement get_owner ()
            raises (NotFound, NotAvailable);
    void set_owner (
            in SDOSystemElement sdo
            ) raises (InvalidParameter, NotAvailable);
    boolean getDirection()
            raises (NotAvailable);
    void setDirection (
            in boolean direction
            ) raises (NotAvailable);
};
```

5 The complete IDL

```
// 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 enumeration_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;
        };
        typedef sequence<OrganizationProperty> OrganizationPropertyList;
        struct DeviceProfile {
            string deviceType;
            string manufacturer;
            string model;
            string version;
            NVList properties;
        };
        struct ServiceProfile {
            string id;
            string interfaceType;
            NVList properties;
            SDOService serviceRef;
    };
    typedef sequence<ServiceProfile> ServiceProfileList;
    struct ConfigurationSet {
            string id;
            string description;
            NVList configurationData;
    };
    typedef sequence<ConfigurationSet> ConfigurationSetList;
```

```
   /** ------- Exceptions -------*/
exception NotAvailable                  exception_body;
exception InterfaceNotImplemented  exception_body;
exception InvalidParameter              exception_body;
exception InvalidReturnValue          exception_body;
exception NotFound                        exception_body;

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

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

```
            raises (InterfaceNotImplemented, NotAvailable);
};

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

```
};

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

interface SDOService {};

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

```
            ) raises (InvalidParameter, NotAvailable);
  boolean getDirection()
            raises (NotAvailable);
  void setDirection (
            in boolean direction
            ) raises (NotAvailable);
};
};
#endif //_SDO_PACKAGE_IDL_
```

6 Summary and requests versus requirements

Resource data model for SDO and common interfaces are proposed in this document. See section 2.3, 2.4, and 2.5 about RFP requirements and compliance.

[References]

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

[2] UML Profile for CORBA

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

[4] Event Service Specification, version 1.1

Appendix A: Complete UML diagram

The complete UML diagram of SDO is as follows.

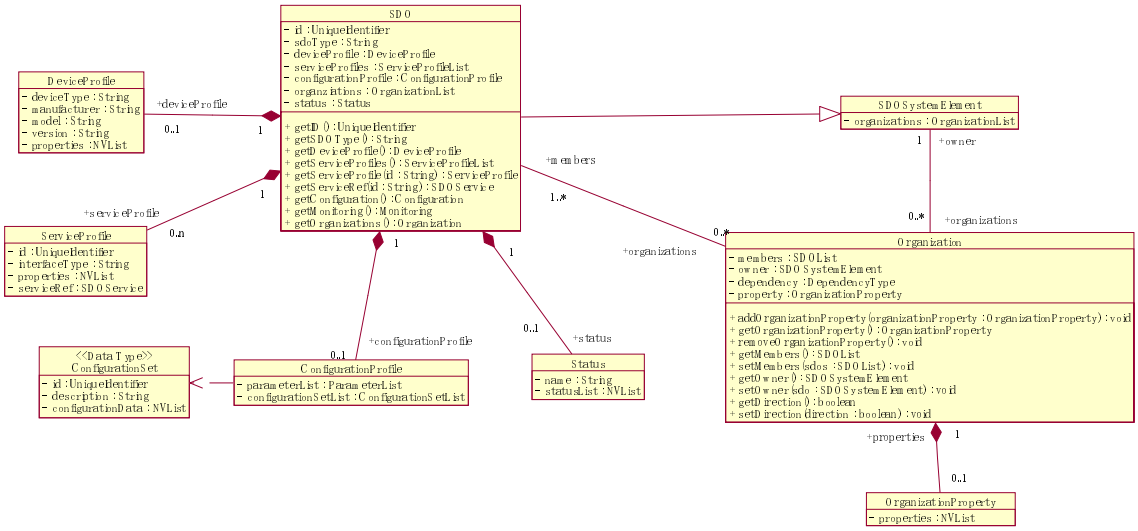


Figure 20. Complete UML diagram

Appendix B: Mapping to ECHONET

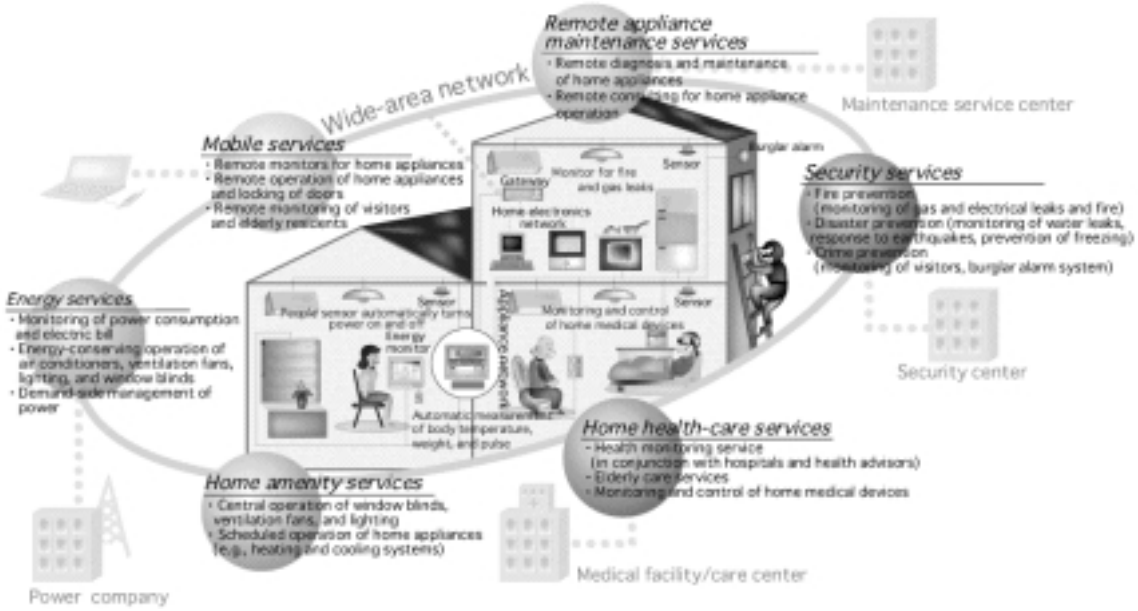
B.1 What is ECHONET

(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.

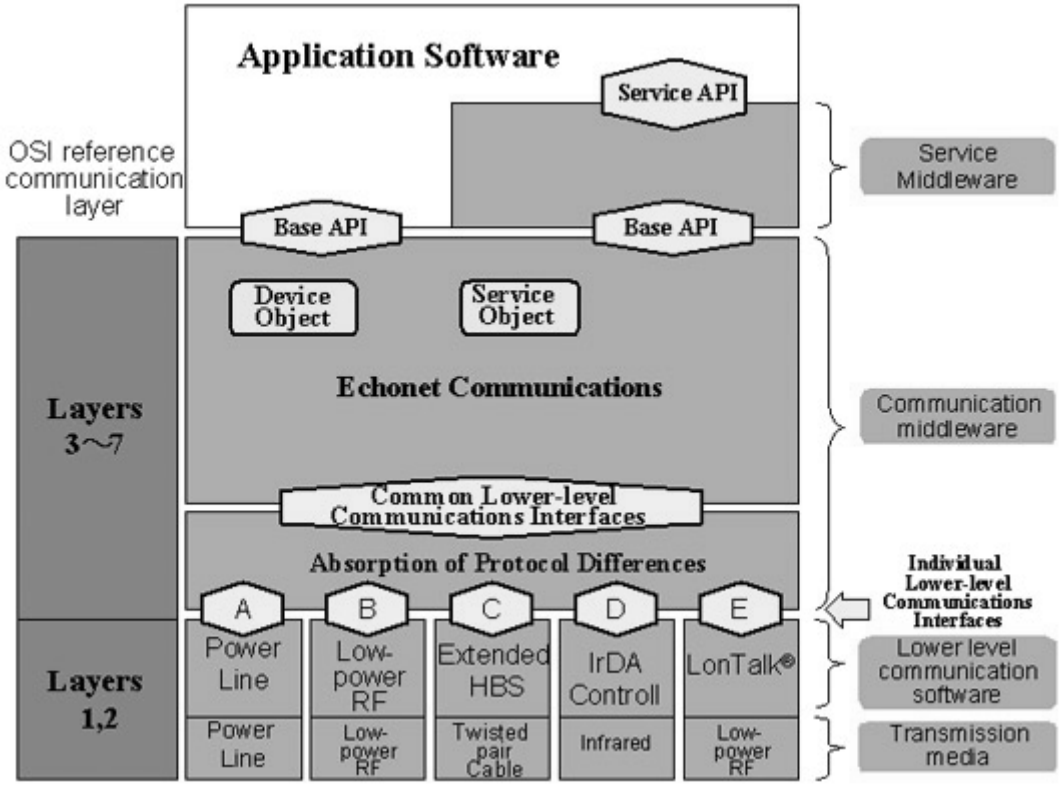
The consortium plans a validation test to evaluate the validity of the systems and software developed and to drive publicity in and outside Japan. It also expects to stage efforts to enhance security, strengthen interworking with the Internet, and develop new application middleware.



B.2 ECHONET architecture

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

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



*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 othercountries.

B.3 Mapping SDO to ECHONET

B.3.1 Resource data structure

In ECHONET, several standard objects have been specified to model home appliances. Typical ones are node profile object and device object. A node profile object describes an addressable device, and a device object describes common attributes of home appliances as well as appliance specific attributes. As a device may have multiple functions and separated hardware (e.g., an air-conditioner may have indoor units, and an outdoor unit), a node object can contain multiple device objects.

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

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

B.3.2 Property mapping from ECHONET to SDO

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

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

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

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

- SDO.id

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

- DeviceProfile

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

- ServiceProfile

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

- Status

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

- Location

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

B.3.3 Common interfaces

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