Initial Submission to Telecom Domain Taskforce RFP

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

Ver. 1.1

sdo/02-09-02

Submitted by
Hitachi Ltd.

Content

1 Preface

This document (OMG document number sdo / 02-09-**) contains a submission proposed as Request for Proposal with the title *PIM and PSM for SDO*. This specification addresses in particular the specific needs and constraints of service systems using SDOs.

1.1 Submitter

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

1.2 Contact point

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: +81-44-959-0244

Fax: +81-44-959-0851

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

- **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 Acknowledgement

The submitters want to thank the active members of SDO Special Interest Group for their help in the preparation of this document:

1.4 Guide to the Submission

This submission represents PIM and PSM for SDOs. An SDO is a logical representation of hardware device or software component and this submission proposes a model and common interfaces for SDOs to utilize them to form application services in an ad hoc manner.

For the interoperable implementations using underlying heterogeneous technologies, SDO is primarily presented as a Platform Independent Model (PIM) using UML. The original CORBA-based implementation is also presented as one Platform Specific Model (PSM) derived from SDO PIM.

1.5 How to read this document

This document is organized as follows. The Preface (chapter 1) contains the information required to accompany a submission. The Preface identifies the submitter and submission and the information required to satisfy the business requirements incumbent upon all adopted technology. Chapter 2 contains a general introduction to the submission and the technology background. Chapter 3 contains the Platform Independent Model for SDO, and chapter 4 contains a Platform Specific Model in form of a CORBA IDL specification of SDO. Chapter 5 contains complete IDL for this proposal, and Chapter 6 summarizes this document. Appendix shows how the proposed specification is mapped to existing device level networking technology in an example of ECHONET mapping, which is a Japanese standard for home appliances network.

1.6 Proof of concept

The PIM described in this document was reverse engineered and generalized from several existing standards as well as actual application systems. An example of these standards is described in appendix of this document. New features are also added for the new usage as service system component. The proposed model and interface are based on these implementations and prototyped.

1.7 Relation to OMG Specification

The COS services like Relationship service and Notification service provide general functions. They may be used to implement the proposed resource data model and monitoring interface as CORBA specific model; however such usage is not required to be compatible to the PIM and PSM for SDO.

1.8 Relation to Pending OMG Specification

The PIM and PSM for SDO addresses the needs of basic data model and common interfaces for logical representation of hardware devices and software components. A model for software design may also address some features of SDO, which is currently under discussion as “Super Structure in UML 2.0” in Analysis and Design Taskforce. The proposed model can be used to implement such models on existing technologies.

1.9 Commercial availability

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

2 Overview

2.1 Scope

The increasing availability of high-performance and low-cost processor technology is enabling computing power to be embedded densely in devices like mobile phones, PDAs, and such facilities as Internet appliances and desktop computers. Furthermore, emerging networking technologies such as wireless LAN, IPv6, and Plug-and-play-enabled platforms enable such devices to connect to each other in an easy and ad-hoc manner and to construct a large scale network of such devices. They will collaborate, and will enable various services and applications. A goal of these infrastructures is to provide a distributed community of devices and software components that pool their services for solving problems, composing applications and sharing contents. The transitions of the particular section of the field-network standards into an OMG specification as well as defining appropriate model for them are the scope of this document.

In addition, the specification has been abstracted into Platform Independent Model expressed by UML, to make this service available to a wider audience. Intended is primarily embedded system usage. However, other areas might equally benefit from using this service.

2.2 Objectives

The PIM and PSM for SDO contained in this document are primarily intended to mediate among devices or software components to form application services. For this objective, basic model and interfaces are specified in this document.

The resource data model specifies capabilities and properties of SDO, which are used for discovery and usage. In addition to the properties about provided services, the resource data model should describe real-world properties such as SDO services (e.g. type of SDO and provided services), location, dependency on other SDOs, as well as organizational properties (e.g. ownership and usage permissions).

The SDO interfaces provide general functions, such as configuration, monitoring, reservation etc. that are identical for all SDOs. These are the

common SDO functions whereas SDOs also provide further specific functions (which are specific to SDO's respective functionality).

2.3 Resolution of RFP requirements and requests

The following model and interfaces are supported in this proposal:

- Mandatory Requirements

1) Resource data model

2) Configuration interface

3) Monitoring interface

- Optional Requirements

Reservation interface is supported in this proposal.

2.4 Responses to RFP issues to be discussed

Interfaces specified in OMG Trader Object Service may be applicable to the CORBA platform specific model of SDO discovery interface, however it is not included in this document.

2.5 Compliance

This specification consists of two parts, a Platform Independent Model (PIM) and a Platform Specific Model (PSM), specifying a realization of the PIM in the terms of CORBA IDL. Both parts each represent indivisible pieces of work. 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. Examples of mapping to other technologies are described in Appendix in this document.

No partial implementation of either the PIM or the PSM is deemed conformant.

3 Platform Independent Model

3.1 Overview and architecture

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

Because devices may contain other peripheral devices, operate multiple software components, and change its structure by extending or removing them, SDO need an extensible model. In this proposal, a minimum unit of SDO is an object which holds profiles describing itself, is accessed independently of other objects, and that is minimum element to be handled to form an application service. In case of software component, a software object described by component model which holds self-describing profile (interface) can be an SDO. In case of hardware device, a computer which has I/Os to interact physical world and has well know functionality can be an SDO, where I/Os may be held in peripheral devices. To assure the extensibility of SDO, SDOs can compose other SDOs (i.e. composite model) in this proposal.

And parameters describing SDO's properties should be extensible so that an application service can be formed as the context concerning SDOs. An example is location data for location-aware services. These parameters data have to be handled in relation to the target device even if they are acquired and maintained by other sensors. As other parameters describing an SDO are also extended, it is necessary for an SDO to be tolerant of these extensions.

For the features discussed above, SDO need an extensible framework in terms of its structure, properties, and relationships of them. Specification proposed in this document assures these extensibilities by Organization among SDOs and other regarding objects. Framework structure is as follows.

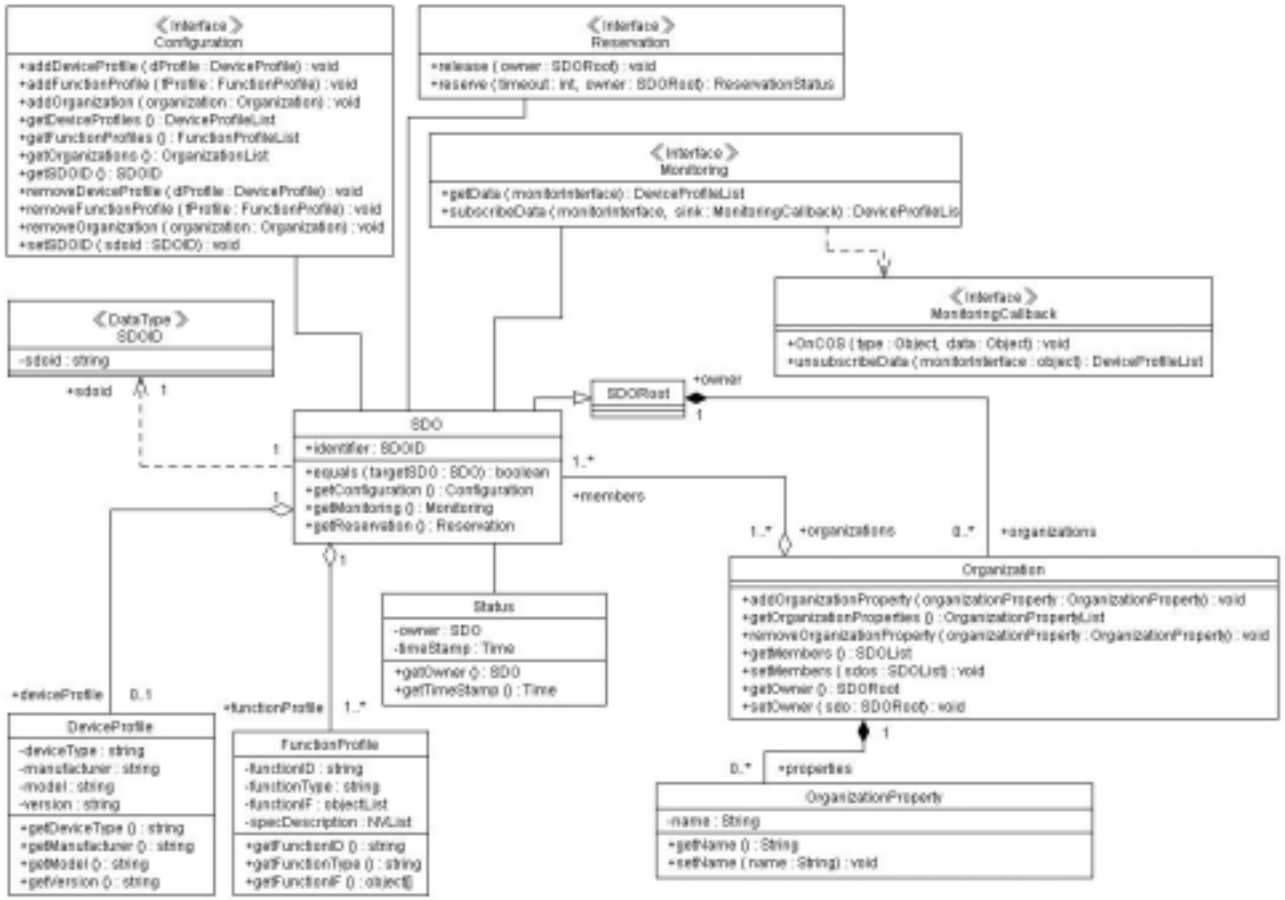


Fig. 1 Framework structure

3.2 Resource data model

3.2.1 Framework definition

3.2.1.1 SDORoot

SDORoot is a reference class to describe the semantic information of mutual relationships among SDOs. All the classes for this purpose are inherited from this class and hold concrete properties for it.

<Attributes>

Attribute	Type	Description
none		

<Operations>

Parameter	Type	Description
none		

3.2.1.2 SDO

An SDO represents a hardware device or a software component. An SDO mediates structure management and access to actual devices or software components. All the interfaces specified in Section 3.2 are navigated from SDO interface.

<Attributes>

Attribute	Type	Description
identifier	SDOID	Identifier in a specific domain. User of system designer can define any domain and identifier identifies SDO uniquely.

SDOID provides abstracted data type for identifier so that a SDO system is able to have a scalable naming system.

- Data type definition: SDOID

Attribute	Type	Description
sdoid	string	SDO identifier in a specified format.

<Operations>

(1) + equals (targetSDO : SDO) : boolean

Parameter	Type	Description
targetSDO	SDO	The reference of target SDO which is compared to the target SDO.
<return>	boolean	TRUE if the targetSDO points the same SDO with itself, and FALSE if not. Identifier is used to check it.

(2) + getConfiguration ( ) : Configuration

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

(3) + getMonitoring ( ) : Monitoring

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

(4) + getReservation ( ) : Reservation

Parameter	Type	Description
<return>	Reservation	Returns reference to Reservation interface with regard to the SDO.

3.2.1.3 Organization

Organization class represents relationships among SDOs. The type of organization is described by OrganizationProperty held in Organization class.

An organization class can contain multiple leaves to describe a set of SDOs, and a SDORoot for a semantic information of the organization (e.g., physical location, network domain, etc.)

<Attributes>

Attribute	Type	Description
none		

<Operations>

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

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

(2) + removeOrganizationProperty (organizationProperty: OrganizationProperty) : void

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

(3) + getOrganizationProperties () : OrganizationPropertyList

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

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

Parameter	Type	Description
sdos	SDOList	The SDOs to be added.

(5) + getMembers ( ) : SDOList

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

(6) + getOwner ( ) : SDORoot

Parameter	Type	Description
<return>	SDORoot	Reference of owner object.

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

Parameter	Type	Description
sdo	SDORoot	Reference of owner object.

3.2.1.4 OrganizationProperty

An OrganizationProperty describes a type of organization of a Organization class. There are several usage of OrganizationProperty depending on the implementation.

Examples of the type of organization are shown in section 3.1.2.

The type of organization is extensible by a new class inherited from this class.

<Attributes>

Attribute	Type	Description
name	string	Name of the OrganizationProperty.

<Operations>

(1) + getName ( ) : String

Parameter	Type	Description
<return>	string	Name.

(2) + setName (name : String) : void

Parameter	Type	Description
name	string	Name.

3.2.1.5 DeviceProfile

DeviceProfile describes profiles of a device. These profiles are static information and are used to check the hardware capabilities. For this purpose, device profile contains information to identify the kind of devices.

<Attributes>

Attribute	Type	Description
deviceType	string	General type name of device. This attribute describes general kind of devices to categorize them and specify fundamental capability of the device.
manufacturer	string	Manufacturer identifier of device.
model	string	Model name of device. This attribute specifies category name in the manufacture product.
version	string	Version number of device.

<Operations>

(1) + getDeviceType () : string

Parameter	Type	Description
<return>	string	Returns name of device type.

(2) + getManufacturer() : string

Parameter	Type	Description
<return>	string	Returns manufacturer name of device.

(3) + getModel () : string

Parameter	Type	Description
<return>	string	Returns model name of the device.

(4) + getVersion () : string

Parameter	Type	Description
<return>	string	Returns version of the device.

3.2.1.6 FunctionProfile

FunctionProfile describes profiles of a function. SDO has at least one FunctionProfile. A function is a minimum unit to be used to form an application service. A function has one or more interfaces and an application service is executed by calling the interfaces. For example, air conditioner has a function (a capability of service) for “fixing room temperature”, and has control interfaces for operations like “set temperature requirement”, “set operation mode (heating / cooling / dehumidifying)”, “turn on / off power”, and so on. A Function profile contains identifier in an SDO, function type used to map an application service and a list interfaces.

<Attributes>

Attribute	Type	Description
functionID	string	Identifier of a function in an SDO. An SDO has one or more functions, and functionID identifies each function in an SDO.
functionType	string	Type of function representing semantic category of a function. Each function has capability to be composed to an application service.
specDescription	NVList	Service specification represented by pairs of an attribute name and a value.
functionIFs	ObjectList	List of interfaces which this function

		has. A function has one or more interface to execute its service.

<Operations>

(1) + getFunctionID () : string

Parameter	Type	Description
<return>	string	FunctionID.

(2) + getFunctionType () : string

Parameter	Type	Description
<return>	string	FunctionTypes.

(3) + getFunctionIF () : stringList

Parameter	Type	Description
<return>	stringList	References to interfaces which this function has.

3.2.1.7 Status

Status represents dynamic condition used to check an SDO's availability to form an application service. The classes which manage dynamic status of the SDO are inherited from this class.

<Attributes>

Attribute	Type	Description
owner	SDO	SDO which owns this status.
timeStamp	Time	Time parameter which is updated when the status is changed.

<Operations>

(1) + getOwner () : SDO

Parameter	Type	Description
<return>	SDO	owner.

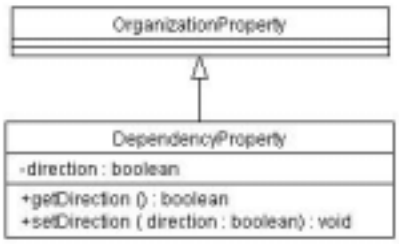
(2)+ getTimeStamp () : Time

Parameter	Type	Description
<return>	Time	timeStamp.

3.2.2 Structure description properties

3.2.2.1 DependencyProperty

DependencyProperty represents a SDO organization caused by SDO.



An example of use case is as follows.

- SDO(owner)-SDOs(members): composite device representation and software component in a device representation
- SDOs(member)-SDO(owner): shared SDO by multiple SDOs, for example an amplifier shared by AV components in a AV component stereo.

<Attributes>

Attribute	Type	Description
direction	boolean	TRUE if owner supervises members, and FALSE if members supervises owner.

<Operations>

(1) + getDirection ( ) : boolean

Parameter	Type	Description
<return>	boolean	Direction.

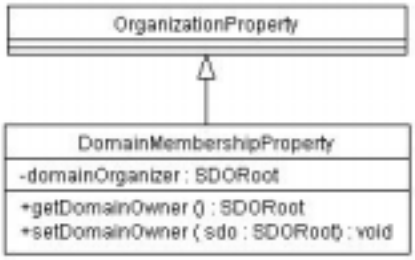
(2) + setDirection (direction : boolean) : void

Parameter	Type	Description
direction	boolean	Direction.

3.2.2.2 DomainMembershipProperty

DomainMembershipProperty represents organization caused by an object other than SDO.

Examples of use case are as follows.



- User(owner)-SDO(members): ownership representation
- Location(owner)-SDO(members): SDOs in a specific area.

Other classes inherited from SDORoot can be the owner of the organization, and various conditions of a set of SDOs can be represented.

<Attributes>

Attribute	Type	Description
none		

<Operations>

None.

3.2.3 Availability properties

3.2.3.1 OperationalStatus

OperationalStatus represents operational availability of the SDO.

<Attributes>

Attribute	Type	Description
statusList	OperationalStatusPropertyList	A List of groups of a name and a value that represent properties of current operation.

<Operations>

(1) + getStatus() : OperationalStatusPropertyList

Parameter	Type	Description
<return>	OperationalStatusPropertyList	Returns a group of OperationalStatusProperty which OperationalStatus class has.

(2) + getStatusValue (statusName : string) : String

Parameter	Type	Description

statusName	String	Name of the OperationalStatusProperty to identify status.
<return>	String	Returns statusValue.

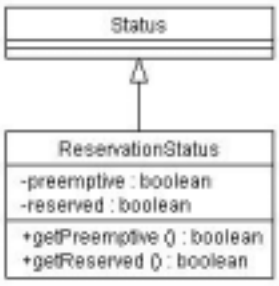
OperationalStatusProperty provides abstracted data type to describe each status of SDO operation.

- Data type definition: OperationalStatusProperty

Attribute	Type	Description
statusName	String	Name of status which concerns with operation of SDO. For example, activity status of SDO has a name "activityStatus", and power status of the device has a name "powerStatus".
statusValue	String	Variable to describe current status. For example, activityStatus takes the statusValue of "active" or "inactive".

3.2.3.2 ReservationStatus

ReservationStatus represents status concerning to the reservation of the SDO.



<Attributes>

Attribute	Type	Description
reserved	boolean	TRUE if this SDO is reserved, and FALSE if not.
preemptive	boolean	TRUE if the usage of the SDO is exclusive, and FALSE if not.

<Operations>

(1) + getReserved ( ) : boolean

Parameter	Type	Description
<return>	boolean	Reserved.

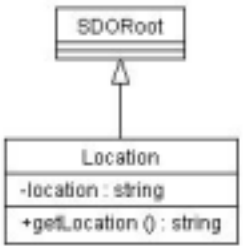
(2) + getPreemptive ( ) : boolean

Parameter	Type	Description

<return>	boolean	Preemptive.

3.2.3.3 Location

Location represents physical location of the SDO connected by Organization object holding DomainMembershipProperty.



<Attributes>

Attribute	Type	Description
location	string	Physical location of devices represented by the SDO.

<Operations>

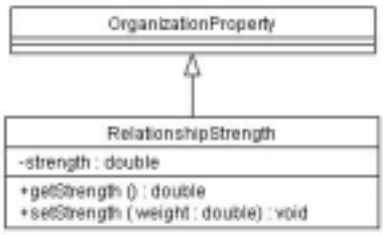
(1) + getLocation( ) : string

Parameter	Type	Description
<return>	string	Value of location.

3.2.4 Weight property

3.2.4.1 RelationshipStrength

RelationshipStrength is a class specializing OrganizationProperty. It is used to add the concept of weight into organizational relationships. Each instance of RelationshipStrength contains a weight value, which is associated with an organizational relationship.



<attributes>

Attribute: strength

Type: double

Description: indicates the strength of a weighted organizational relationship.

<Operations>

(1) getStrength(): double

Parameter: <return>

Type: double

Description: returns a value of organizational relationship strength.

(2) setStrength(strength: double): void

Parameter: strength

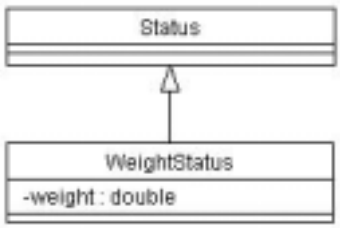
Type: double

Description: a strength value that will be assigned to an organizational relationship

The semantics and usage of organizational relationship strength varies across applications and application domains. It may indicate the usefulness of another SDO (or other SDOs), in order to select useful interaction partners and prioritize different organizational relationships for the interactions. It also depends on implementations how to calculate and update strength values. It may be calculated and updated based on the average response time between SDOs, the availability ratio of a SDO, the number of interactions between particular SDOs.

3.2.4.2 WeightStatus

WeightStatus is a class specializing Status. It indicates the current weight of an SDO. The concept of weight is used to adjust SDO's availability and control the SDO's lifecycle.



<Attributes>

Attribute: weight

Type: double

Description: indicates the weight of an SDO.

<Operations>

(1) getWeight(): double

Parameter: <return>

Type: double

Description: returns a weight value of an SDO.

Each SDO increases and decreases its weight (for living). It depends on applications (implementations) how to increase and decrease the weight value. For example, an SDO may gain weight in exchange for providing its own service, and lose weight for invoking another SDO's service and utilizing resources (e.g. CPU cycles, network bandwidth and memory space). The level of weight affects SDO's behaviors. For example, an abundance of weight may be considered as an indication of higher demand for the SDO; thus the SDO may be designed to favor replication in response to higher level of stored energy. A scarcity of weight (an indication of lack of demand) may eventually cause the SDO's removal. The concept of weight can prevent SDOs from overusing resources and other SDOs' services, since a certain amount of weight is required for them to take any activity and they will die when their weight levels become zero. The concept of weight also prevents the SDOs that have not been used by anyone continue to stay in network and use resources.

3.3 Interfaces definition

The following sections describe common interfaces for SDOs. These interfaces are ones in RFP and used to configure, monitor, and reserve SDOs. Resource data proposed in Section 3.1 are set or got by these interfaces.

3.3.1 Configuration interface

Configuration interface provides operations to add or remove data specified in resource data model. Operations to get references to a set of data or objects are also included to navigate specific resource data.

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

Parameter	Type	Description
dProfile	DeviceProfile	DeviceProfile to be added.

(2) + addFunctionProfile (fProfile : FunctionProfile) : void

Parameter	Type	Description
dProfile	FunctionProfile	FunctionProfile to be added.

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

Parameter	Type	Description
organization	Organization	Organization to be added.

(4) + getDeviceProfiles () : DeviceProfileList

Parameter	Type	Description
<return>	DeviceProfileList	DeviceProfiles that the SDO has.

(5) + getFunctionProfiles () : FunctionProfileList

Parameter	Type	Description
<return>	FunctionProfileList	FunctionProfiles that the SDO has.

(6) + get Organization () : OrganizationList

Parameter	Type	Description
<return>	OrganizationList	Organization that the SDO has.

(7) + getSDOID () : SDOID

Parameter	Type	Description
<return>	SDOID	SDOID of the SDO.

(8) + removeDeviceProfile (dProfile : DeviceProfile) : void

Parameter	Type	Description
dProfile	DeviceProfile	DeviceProfile to be removed.

(9) + removeFunctionProfile (fProfile : FunctionProfile) : void

Parameter	Type	Description
fProfile	FunctionProfile	FunctionProfile to be removed.

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

Parameter	Type	Description
organization	Organization	Organization to be removed.

(11) + setSDOID (sdoid : SDOID) : void

Parameter	Type	Description
sdoid	SDOID	SDOID to couple with this SDO.

3.3.2 Monitoring interface

Monitoring interface provides operations to monitor the resource data. Two kind of operations are proposed,

- Polling: data is acquired only when the method is called.
- Subscription: data is acquired every time when the specified event occurs.

An operation to stop the subscription is also proposed.

(1) + getData (monitorInterface : object) : object

Parameter	Type	Description
monitorInterface	object	Interface to get data to be monitored.
object	object	DeviceProfiles that the SDO has.

(2) + subscribeData (monitorInterface : object, sink : MonitoringCallback) : object

Parameter	Type	Description
monitorInterface	object	Interface to get data to be monitored.
sink	MonitoringCallback	Sink object to get callback operation.
<return>	object	DeviceProfiles that the SDO has.

Interface MonitoringCallback specifies interface for a sink object to get call back invocation from the subscribed SDO. (3) + unsubscribeData (sink : MonitoringCallback) : objectList

Parameter	Type	Description
monitorInterface	object	Interface to get data to be monitored.

<return>	DeviceProfileList	DeviceProfiles that the SDO has.

<operation> MonitoringCallback

(1) OnCOS (type : Object, data : Object) : void

Parameter	Type	Description
type	object	Interface to get data to be monitored.
data	object	Data value.

3.3.3 Reservation interface

Reservation interface provides operation to reserve the usage of an SDO. Reservation scheme is based on the criteria defined in the resource data model.

(1) + release (owner : SDORoot) : void

Parameter	Type	Description
owner	SDORoot	Owner object which reserves this SDO.

(2) + reserve (timeout : int, owner : SDORoot) : ReservationStatus

Parameter	Type	Description
timeout	int	Length of time to release reservation.
owner	SDORoot	Owner object which reserves this SDO.
<return>	ReservationStatus	Returns ReservationStatus which this operation generates.

4 Platform Specific Model: Mapping to CORBA IDL

4.1 Overview

This specification defines a SDO resource data model and common interfaces on CORBA systems. It represents the CORBA Platform Specific Model (PSM) derived from the SDO Platform Independent Model (PIM) described in section 3 of this document. In this particular case, some interfaces are mapped onto CORBA services. The PIM described in section 3 was derived from this PSM and other similar technologies through generalization.

Design rationale: How the CORBA services are used in CORBA PSM

4.2 Classes and data structures

4.2.1 Framework definition

4.2.1.1 SDORoot

```
interface SDORoot{
    attribute  OrganizationList  organizations;
};
```

SDORoot class defines the fundamental object class for the objects that are included in the SDO resource data model. All the classes that describe relationships among SDOs, like Location class, are inherited from this class.

Mapping from Platform Independent Model

This IDL interface definition is the result of one-to-one mapping from the UML class SDORoot, described in section 3.2.1.1.

4.2.1.2 SDO

```
interface SDO:SDORoot{
    attribute  string               sdoid;
    attribute DeviceProfile         dProfile;
    attribute FunctionProfile       fProfile;
    attribute Status                 status;
    attribute Organization          organization;
```

```
    boolean              equals(in SDO targetSDO);
    Configuration        get_configuration( );
    Monitoring           get_monitoring( );
    Reservation          get_reservation( );
    };
  typedef sequence<SDO>      SDOList;
```

SDO is a logical representation class of a hardware device or a software component. SDO has a data type structure SDOID, which provides the unique identifier that is assigned to a SDO.

Mapping from Platform Independent Model

This IDL interface definition is the result of one-to-one mapping from the UML class SDO, described in section 3.2.1.2.

4.2.1.3 DeviceProfile

```
  struct DeviceProfile{
    string                deviceType;
    string                manufacturer;
    string                model;
    string                version;
  };
```

DeviceProfile is the structure of the device-specific data which the SDO represents.

Mapping from Platform Independent Model

This IDL structure is the result of one-to-one mapping from the UML class DeviceProfile, described in section 3.2.1.5.

4.2.1.4 FunctionProfile

```
  struct FunctionProfile{
    typedef sequence<Object>  FunctionIFList;

    string                functionID;
    string                functionType;
    string                specDescription;
    FunctionIFList        functionIFs;
  };

  typedef sequence<FunctionProfile>   FunctionProfileList;
```

FunctionProfile is the structure of the function-specifying data which the SDO represents. An SDO owns at least one function, and a function has an identifier in the SDO, function type description for specifying available service and interfaces for usage the function. FunctionProfile is a collection of those information.

Mapping from Platform Independent Model

This IDL structure is the result of one-to-one mapping from the UML class FunctionProfile, described in section 3.2.1.6.

```
interface Organization{
    attribute SDORoot sdoroot;

    void      add_organization_property(in OrganizationProperty
                                        organizationProperty);
    SDOList   get_members( );
    void      set_members( in SDOList sdos);
    OrganizationPropertyList get_organization_properties( );
    void          remove_organization( in OrganizationProperty
                                       organizationProperty );
    SDORoot   get_owner();
    void          set_owner( in SDORoot sdo);
};
```

Organization class represents relationships among SDOs. An Organization class has at least one OrganizationProperty.

Mapping from Platform Independent Model

This IDL interface is the result of one-to-one mapping from the UML class Organization, described in section 3.2.1.3.

4.2.2 OrganizationProperty

```
interface OrganizationProperty{
    attribute Organization organization;

    string get_name();
    void   set_name(in string name);
};
```

OrganizationProperty class describes type of an Organization class.

Mapping from Platform Independent Model

This IDL interface is the result of one-to-one mapping from the UML class OrganizationProperty, described in section 3.2.1.4.

4.2.2.1 Status

```
interface Status{
   SDORoot  get_owner();
   Time     get_timestamp();
   };
```

Status represents current status of SDO, and is a structure of data that changes dynamically. The classes to describe concrete state of SDO like ReservationStatus and OperationalStatus are inherited from this class.

Mapping from Platform Independent Model

This IDL structure is the result of one-to-one mapping from the UML class Status, described in section 3.2.1.7.

4.2.3 Structure description properties

4.2.3.1 DependencyProperty

```
interface DepencencyProperty:OrganizationProperty{
   boolean  get_direction( );
   void     set_direction( in  boolean  direction);
};
```

DependencyProperty class describes a SDO organization caused by SDO. Usage of dependency property is relation between SDO and SDO, or between user and SDO.

Mapping from Platform Independent Model

This IDL interface is the result of one-to-one mapping from the UML class DependencyProperty, described in section 3.2.2.1.

4.2.3.2 DomainMembershipProperty

```
interface DomainMembershipProperty:OrganizationProperty{
   SDORoot   get_domain_owner( );
   void      set_domain_owner( in  SDORoot  sdo);
};
```

DomainMembershipProperty is the Organized relation caused by object other than SDO.

Mapping from Platform Independent Model

This IDL interface is the result of one-to-one mapping from the UML class DomainMembershipProperty, described in section 3.2.2.2.

4.2.3.3 Location

```
struct Location:SDORoot{
   string  location;
};
```

Location represents physical location of the SDO

Mapping from Platform Independent Model

This IDL interface is the result of one-to-one mapping from the UML class Location, described in section 3.2.3.3.

4.2.4 Availability properties

4.2.4.1 OperationalStatus

```
struct OperationalStatusProperty{
   String    statusName;
   String    statusValue;
};
typedef sequence<OperationalStatusProperty>
                              OperationalStatusPropertyList;

interface OperationalStatus:Status{
   OperationalStatusPropertyList get_status();
   String                        get_status_value(in  String
                                                  statusName);
};
```

OperationalStatus represents the status of current operation of the function which is belongs to the SDO.

Mapping from Platform Independent Model

This IDL structure is the result of one-to-one mapping from the UML class OperationalStatus, described in section 3.2.3.1.

4.2.4.2 ReservationStatus

```
interface ReservationStatus:Status{
   boolean get_reserved();
   boolean get_preemptive();
};
```

ReservationStatus represents the status of current reservation of the SDO.

Mapping from Platform Independent Model

This IDL structure is the result of one-to-one mapping from the UML class ReservationStatus, described in section 3.2.3.2.

4.2.5 Weight properties

```
     interface RelationshipStrength:
OrganizationProperty
{
     attribute double strength;
};
```

IDL definition for WeightStatus

```
interface WeightStatus: Status
{
     readonly attribute double weight;
};
```

4.3 Interfaces

4.3.1 Configuration interface

```
interface Configuration{
   void add_device_profile(in DeviceProfile  dProfile);
   void add_function_profile(in FunctionProfile  fProfile);
   void add_organization(inOrganization  organization);
   DeviceProfile      get_device_profiles( );
   FunctionProfileList get_function_profiles( );
   OrganizationList   get_organizations( );
   SDOID              get_sdoid( );
   void               remove_device_profile(in DeviceProfile
                                            dProfile);
   void               remove_function_profile(in FunctionProfile
                                            fProfile);
```

```
    void            remove_organization(in Organization
                                        organization);
    void            set_sdoid(in SDOID sdoid);
};
```

Configuration interface provides operations to add or remove data specified in resource data model. Operations to get references to a set of data or objects are also included to navigate specific resource data.

4.3.2 Monitoring interface

```
interface MonitoringCallback{
    void            on_cos(in Any type, in Any data);
};

interface Monitoring{
    DeviceProfileList get_data(in Object monitorInterface);
    DeviceProfileList subscribe_data(in Object monitorInterface,
                                     in MonitorinngCallback sink);
    DeviceProfileList unsubscribe_data(in Object
                                       monitorInterface);
};
```

Monitoring interface provides operations to monitor the resource data..

4.3.3 Reservation interface

```
interface Reservation{
    void            release(in SDORoot owner);
    ReservationStatus reserve(in long timeout, in
                              SDORoot owner);
};
```

Reservation interface provides operation to reserve the usage of an SDO.

5 The complete IDL

```
module SDOInterface
{
   typedef    Long  Time;
   struct SDOID;
   struct DeviceProfile;
   struct FunctionProfile
   struct OperationalStatusProperty;
   interface SDO;
   interface SDORoot;
   interface Status;
   interface Configuration;
   interface Reservation;
   interface Monitoring;
   interface MonitoringCallback;
   interface OperationalStatus;
   interface ReservationStatus;
   interface DepencencyProperty;
   interface DomainMembershipProperty;
   interface   Organization;
   interface   OrganizationProperty;
   typedef sequence<FunctionProfile>   FunctionProfileList;
   typedef sequence<OperationalStatusProperty>
                                 OperationalStatusPropertyList;
   typedef sequence<SDO>       SDOList;
   typedef sequence<Organization>             OrganizationList;
   typedef sequence<OrganizationProperty>   OrganizationPropertyList;

   struct SDOID {
      string   sdoid;
   };

   interface SDORoot{
      attribute  OrganizationList  organizations;
   };

   interface SDO:SDORoot{
      attribute  string                  sdoid;
      attribute DeviceProfile        dProfile;
      attribute FunctionProfileList  fProfiles;
      attribute Status                status;
```

```
    attribute Organization        organization;

    Boolean                  equals(in SDO targetSDO);
    Configuration           get_configuration( );
    Monitoring               get_monitoring( );
    Reservation              get_reservation( );
    };

struct DeviceProfile{
    string                        deviceType;
    string                       manufacturer;
    string                           model;
    string                           version;
};

struct FunctionProfile{
   typedef sequence<Object>  FunctionIFList;

    string                        functionID;
    string                       functionType;
    string                      specDescription;
    FunctionIFList                functionIFs;
};

interface Status{
    SDORoot  get_owner();
    Time        get_timestamp();
    };

struct OperationalStatusProperty{
    String    statusName;
    String    statusValue;
};

interface OperationalStatus:Status{
    OperationalStatusPropertyList get_status();
    String                              get_satus_value(in String statusName);
};

interface ReservationStatus:Status{
    boolean  get_reserved();
    boolean  get_preemptive();
};
```

```
interface Configuration{
   void  add_device_profile(in DeviceProfile  dProfile);
   void  add_function_profile(in FunctionProfile  fProfile);
   void  add_organization(inOrganization  organization);
   DeviceProfile       get_device_profiles( );
   FunctionProfileList  get_function_profiles( );
   OrganizationList     get_organizations( );
   SDOID                get_sdoid( );
   void                 remove_device_profile(in DeviceProfile dProfile);
   void                 remove_function_profile(in FunctionProfile  fProfile);
   void                 remove_organization(in Organization  organization);
   void                 set_sdoid(in SDOID  sdoid);
};

interface Reservation{
   void                 release(in  SDORoot  owner);
   ReservationStatus  reserve(in  long  timeout, in  SDORoot  owner);
};

interface MonitoringCallback{
   void                 on_cos(in Any type, in Any data);
};

interface Monitoring{
   DeviceProfileList  get_data(in Object monitorInterface);
   DeviceProfileList  subscribe_data(in Object  monitorInterface,
                                     in MonitorinngCallback  sink);
   DeviceProfileList  unsubscribe_data(in Object monitorInterface);
};

struct Location:SDORoot{
   string  location;
};

interface Organization{
   attribute  SDORoot  sdoroot;

   void       add_organization_property(in  OrganizationProperty
                                        organizationProperty);
   SDOList   get_members( );
   void       set_members( in  SDOList  sdos);
   OrganizationPropertyList  get_organization_properties( );
   void          remove_organization( in  OrganizationProperty
                                      organizationProperty );
```

```
        SDORoot   get_owner();
        void        set_owner( in  SDORoot  sdo);
    };

    interface OrganizationProperty{
        attribute  Organization  organization;

        string  get_name();
        void   set_name(in string  name);
    };

    interface DepencencyProperty:OrganizationProperty{
        boolean  get_direction( );
        void       set_direction( in  boolean  direction);
    };

    interface DomainMembershipProperty:OrganizationProperty{
        SDORoot  get_domain_owner( );
        void        set_domain_owner( in  SDORoot  sdo);
    };

};
```

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

Appendix: PSM for other platforms

1 Mapping to ECHONET

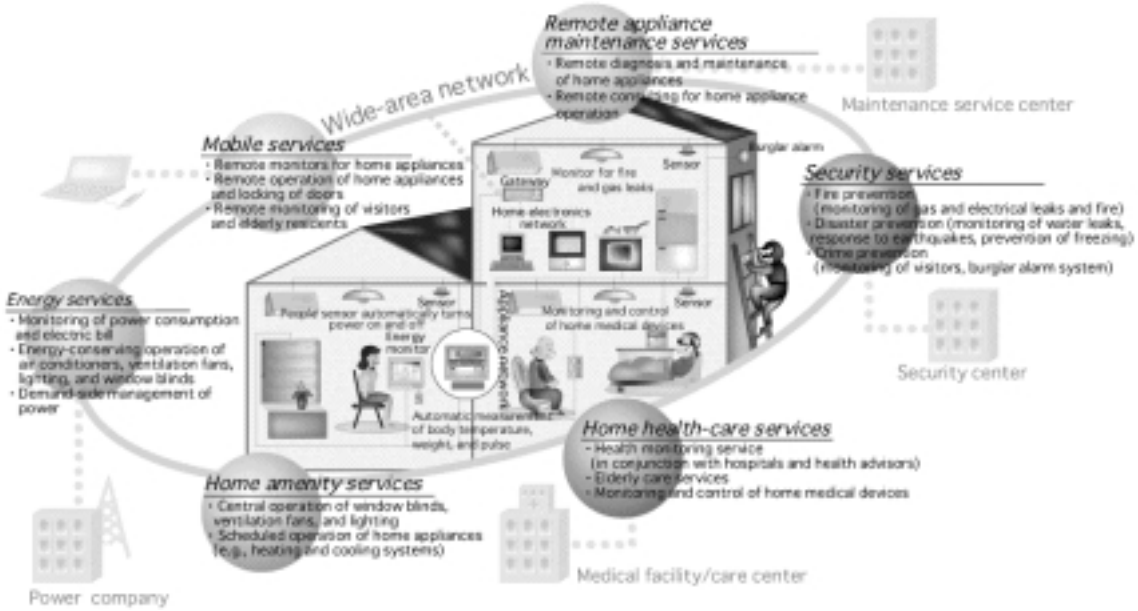
1.1 What is ECHONET

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

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

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

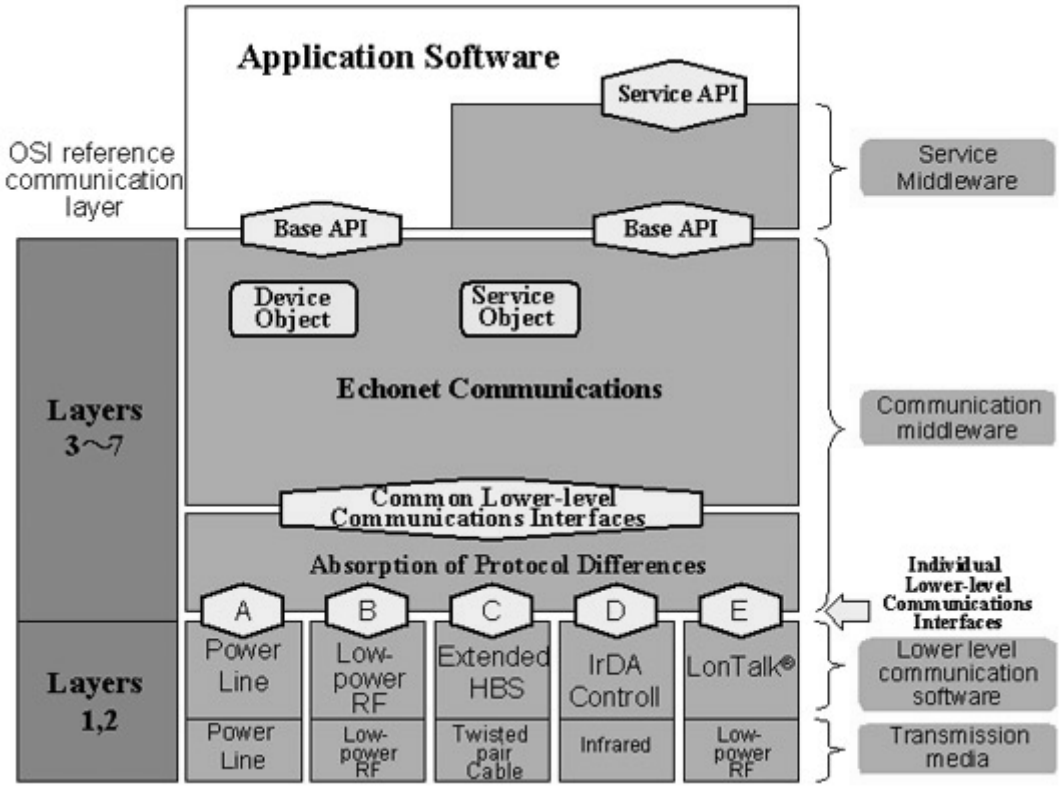
The consortium plans a validation test to evaluate the validity of the systems and software developed and to drive publicity in and outside Japan. It also expects to stage efforts to enhance security, strengthen interworking with the Internet, and develop new application middleware.



1.2 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.

1.3 Mapping SDO to ECHONET

1.3.1 Resource data model

1.3.1.1 Overview

In ECHONET, several standard objects have been specified to model home appliances, especially node profile object class and device object class. A node profile object describes an addressable device, and an 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 access from applications in other network domain to ECHONET devices. Composite SDO structure of SDOs can be mapped to this structure and enable unified management of hardware device and software components.

1.3.1.2 Example of property mapping from SDO to ECHONET

Fig. 2 Example of property mapping

- SDOID

SDOID is mapped to "Unique identifier data" if it represents ECHONET Device Object

- DeviceProfile

The properties specified in DeviceProfile class are mapped to some properties of Device Object and Profile Object in ECHONET, that are "Version data", "Manufacturer code", "Place of business code", "Product code", "Serial number" and "Date of manufacture". These properties are acquired from ECHONET middleware and provide unique description of a device.

- FunctionProfile

In ECHONET, functions of a device is described by property map holding an array of codes unique to each function. The properties specified in FunctionProfile classare 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 in ECHONET.

- OperationalStatus

ECHONET specifies properties representing status of an object, that are "Operating status", "Fault status" and "Fault content" of Device Object. These properties are represented by classes inherited from Status class, or in the Operational class as named values.

- Location

The "Instllation location" property is specified in each Device Object in ECHONET to describe the location of each component. (e.g., outdoor unit, indoor unit) Location class can handle this property to an or a set of SDOs that are mapped to ECHONET Device object.

1.3.2 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.