Universidad Politécnica de Madrid

Autonomous Systems Laboratory

Specific Communication Service Mapping Mapping to CORBA

Version 2.0

Ricardo Sanz

15th January 2002

Status: Final

Specific Communication Service Mapping Mapping to CORBA

Abstract:

The mapping of ACSI to CORBA will define how the concepts, objects and services of ACSI will be implemented using CORBA distributed objects technology, allowing interoperability among substation functions and devices of different manufacturers.

This document has been issued in accordance with the document *IST-1999-10258 Annex 1 – "Description of Work"*. The identification of this deliverable in the Periodic Progress Reports is DL 3.1.

Copyright:

Table of Contents

Part 1
Introductory material

1. Introduction

1.1 Purpose of this document

This document is the DOTS deliverable 3.1 *Specific Communication Service Mapping to CORBA.*

The main content of this deliverable is a mapping from the draft standard IEC 61850 abstract specification of communication service to a concrete communication infrastructure based on CORBA specifications.

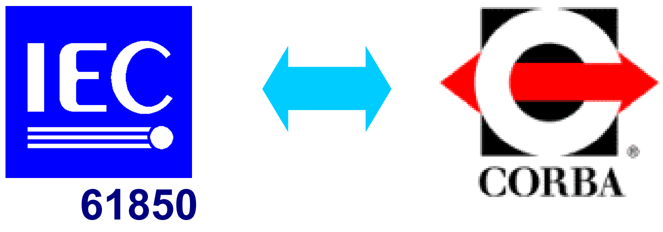


Figure 1-1: Mappping between IEC 61850 and

This mapping will be submitted to the IEC Technical Committee in charge of the specification of other mappings (TC57) accompanied by the General Model Definition document, elaborated in the workpackage number 1.

This document will serve as a basic guideline to provide a *real implementation* of IEC 61850 models over real platforms, using CORBA technology as support for

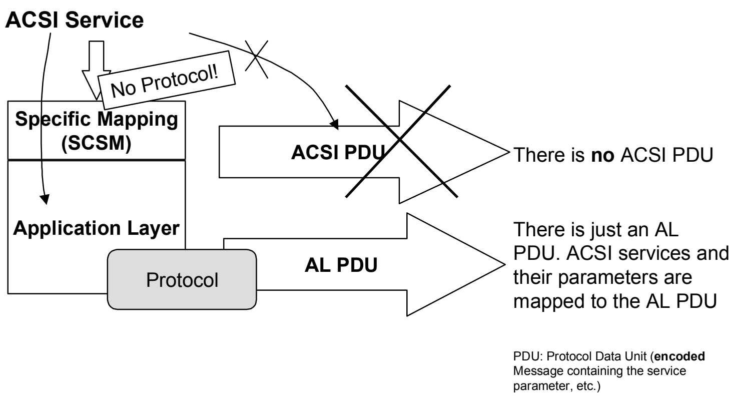

ACSI Service
No Protocol!
Specific Mapping (SCSM)
Application Layer
Protocol
ACSI PDU
There is no ACSI PDU
AL PDU
There is just an AL PDU. ACSI services and their parameters are mapped to the AL PDU
PDU: Protocol Data Unit (encoded Message containing the service parameter, etc.)

Figure 1-2: ACSI mapping to an specific application (communication) layer [From IEC 61850].

distribution. The Abstract Communication Service Interface (ACSI) specified by IEC 61850 needs to be mapped to a real (not abstract) Specific Communication Service to be usable by application developers. This document provides an ACSI mapping to CORBA [OMG 1999].

This means mainly that the communication services used to make a distributed SAS application willl be those provided by CORBA.

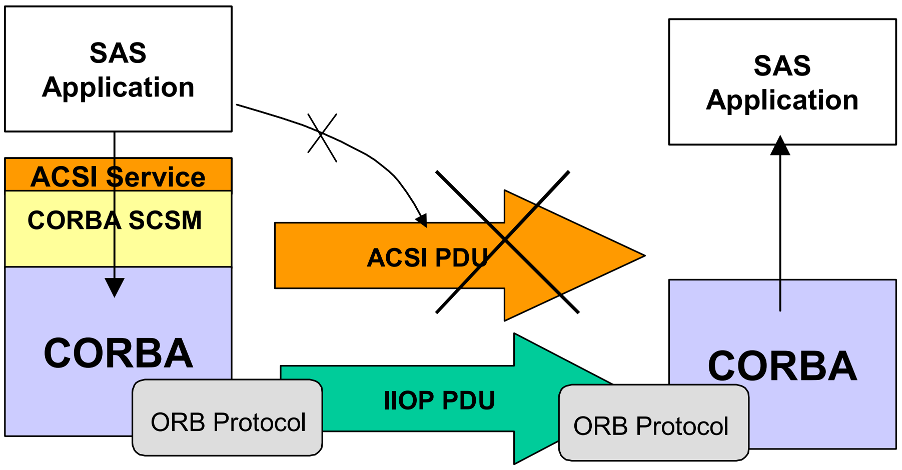


Figure 1-3: Basic view of the ACSI mapping to CORBA (compare with Figure 1-1).

It should be noted however that CORBA is not a communication service but a middleware service, providing other types of functionality and methodology that go beyond those of pure communication services (for example automatic generation of skeletons and proxies, interface repositories, server object management, etc).

1.2 Scope of the ACSI

The IEC 61850 ACSI applies —but is not restricted— to the communication between devices in electrical substations. The ACSI provides services for device interaction that are structured in a collection of interfaces. The following three top-level basic abstract interfaces are considered[1]:

1. **Services** to be used by a remote client for:
   - Real-time data access
   - Device control
   - Event reporting and logging
   - File transfer
   - Self-description of devices
   - Data typing and discovery of data types

[1] All interfaces defined in the ACSI are abstract, so, no further consideration/mention of abstractness will be done (this means that we will use the term "interface" where the IEC 61850 Part 7-2 uses "abstract interface".

2. **System-wide event distribution** between an application in one device and many remote applications in different devices.
3. **Transmission of sampled measured values**.

IEC 61850 Part 7-2 defines the abstract communication service interface (ACSI) in terms of:

- An abstract model of hierarchical data object classes of all data and control information that can be accessed by the ACSI.
- An abstract model of the services that operate on the objects[2] (interaction between a client application and the server application)
- The service procedures of the abstract services that has to be implemented in the server
- The parameter associated with each service

This part of IEC 61850 does not specify individual implementations or products, nor does it constrain the implementation of entities and concrete interfaces within a substation device. This part does not specify the mapping of the abstract functionality to standard application layers. These mappings are specified in the specific communication service mappings (SCSM) part of this standard (Part IEC 61850-8-x or IEC 61850-9-x).

1.3 Mapping Rationale

This mapping will enable the development of applications that follow IEC 61850 specification over a CORBA compliant middleware.

1.3.1 What is IEC 61850? What is it for?

IEC 61850 is an IEC draft standard titled: ***Communication Networks and Systems in Substations***.

It focus on the recognized necessity of making easy the interoperation of systems, networks and devices in industrial control installations and specifically in the electric utilities sector. This necessity is targeted by two approaches (in draft status) in the context of power substations: the north-american UCA 2.0 architecture, and the IEC 61850 standard.

The objective of IEC 61850 is to develop a communication standard that meets performance and cost requirements, and which will support future technological developments. The standard tries to use open protocols and provide support for self descriptive objects to be possible to add new functionality.

[2] Even when the modeling tries to be object-oriented it still has some procedural corruption, for example when it says "*services that operate on the objects*" it is thinking in "*procedures that operate on the objects*". A true OO version of this should be "*services that are requested from objects*".

1.3.2 What is CORBA? What does it do? What is OMG?

CORBA is the acronym for **C**ommon **O**bject **R**equest **B**roker **A**rchitecture, OMG's open, vendor-independent architecture and infrastructure that computer applications use to work together over networks. Using standard protocols like OMG's IIOP, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network.

Some people think that CORBA is the only specification that OMG produces, or that the term "CORBA" covers all of the OMG specifications. Neither is true. The OMG specifies many other object-oriented topics, like UML, MOF or XMI.

1.3.3 What is ACSI?

The **A**bstract **C**ommunication **S**ervice **I**nterface is a *virtual interface* to an IED providing abstract communication services, e.g. connection, variable access, unsolicited data transfer, device control and file transfer services, independent of the actual communication stack and profiles used.

Being abstract, it is not usable to implement any application. To make possible the construction of applications, it is necessary to map it to a specific communication service. This is what this document is about: it contains a Specific Comunication Service Mapping (SCSM) for the ACSI.

1.3.4 What is the reason for trying to map IEC 61850 ACSI to CORBA ?

CORBA is the de-facto standard for distributed object systems. While on Microsoft Windows desktop machines the natural middleware is COM/DCOM/COM+ and Java in Internet-oriented applications, CORBA goes beyond any limitation providing a true open infrastructure for application freedom.

Any application that requires some form of heterogeneity, robustness, transparency, scalability and/or evolvability will find in CORBA a suitable platform. This is the reason why most industrial, object-oriented, critical application are being deployed using CORBA technology [Sanz 2000].

1.3.5 Can CORBA be considered an Specific Communication Service ?

It is not very clear what is an SCS from the point of view of IEC 61850, but CORBA (and more specifically the IIOP) can be considered a SCS. In fact, the ACSI looks like it has been an evolution from the MMS specification.

1.3.6 What is the CORBA communication stack proper ?

CORBA systems can be deployed over any communication system. The most common communication stack is TCP/IP because this is the protocol suite used by the basic interoperability protocol IIOP. IIOP stands for Internet Inter-ORB Protocol, *i.e.* the IOP over the Internet protocols, *i.e.* TCP/IP.

But using ORB based interfaces makes possible the use of other specialized protocols for any specific purpose without the need of modifying the objects that implement the application functionality. You can find CORBA over ATM or hardware bus protocols.

1.3.7 Is any standardization organization using OMG specifications ?

The following OMG specifications are ISO standards:

IDL	ISO/IEC 14750
Trader	ISO/IEC 13235
MOF	2nd CD ISO/IEC 14769
Interoperability	ISO/IEC 19500-2

Other SDOs (Standards Development Organizations) and Consortia that are using OMG specifications are:

Parley API v1.1	OMG IDL, used for API spec
IEEE 1226	OMG IDL, used for testing I/Fs
Davic 1.4.1	CORBA 2.2 is referenced
W3C	OMG IDL, used in DOM
ANSI X12	UML, used for modeling EDI business processes
Un/CEFACT	UML, used for modeling EDI business processes
ISO JTC1/SC24	OMG IDL, used for VRML work

1.4 Objectives of the mapping

Within this report, the objects and services of the ACSI will be mapped to CORBA. Data
exchange information will consist of real-time monitoring and control data, including measured
values, events and files

The mapping of ACSI to CORBA will define how the concepts, objects and services of ACSI should
be implemented using CORBA distributed objects technology, allowing interoperability among
substation functions and devices of different manufacturers.

1.5 Document Structure

This document is structured in three parts:

1. Introductory material
2. The mapping proper
3. Reference material

The introductory material goes from Chapter 1 (Introduction) to Chapter 5 (Mapping Details). This material includes the introduction (this chapter), glossary, mandatory references and material necessary to interpret this mapping (Chapters 4, Assessment of 61850 & CORBA, and 5, Mapping Details). Please read these chapters before going to part 2.

The second part contains the mapping specification part. We have tried to adopt the same structure than the mapping documents published by the IEC, in order to simplify the coordinated use of both documents and ease the process of submission to the Technical Committee in charge of the specification of other mappings (TC57).

Each chapter in this part, from chapter 6 to chapter 19, addresses the mapping of the specific chapter of the ACSI specification.

For example, chapter 6 of this document addresses the ***Mapping of the Server Model***, that corresponds to chapter 6 of IEC 61850 Part7-2 ***Server Model***. Chapter 7 of this document addresses the ***Mapping of the Application Association Model***, that corresponds to chapter 7 of IEC 61850 Part7-2 ***Application Association Model***.

The last part contains reference material like additional bibliographic references, IDL files and UML models.

2. Definitions, acronyms and abbreviations

Read the separate document *DOTS Project Glossary* (DOTS Document No. IST10258/010).

3. Mandatory References

This section contains all mandatory references used in this document. References are grouped by document origin: DOTS, IEC, OMG and Others. For this document "*mandatory*" means that what these documents say is applicable to this mapping *except explicit negation of applicability*.

3.1 DOTS References

Distribution of DOTS deliverables is specified in the DOTS Project Programme. Other DOTS documents distribution policies are specified in the DOTS Project Quality Plan. Contact DOTS Overall Project Manager to get further information about any DOTS document.

We mention here references that are not publicly available, even if them are, in principle, mandatory. We can do this because this is a DOTS project document and not a plublicly available specification independent of DOTS. In the case of submittal, we must modify this to refer only to available publications (available at least for the intended audience of this document).

Title	DOTS Project Programme
Date	18/10/1999
Release	1.0
Publisher	DOTS Consortium

Title	General Requirements Specification
Number	IST10258/001
Date	17/2/2000
Release	1.1
Publisher	DOTS Consortium

Title	Project Quality Plan
Number	IST 10258/002
Date	2/6/2000
Release	1.1
Publisher	DOTS Consortium

Title	DOTS General Model Definition

Number	IST10258/004
Date	27/4/2000
Release	1.0
Publisher	DOTS Consortium

Title	DOTS Report on IEC TC57 meeting at Orlando
Number	IST10258/006
Date	20/3/2000
Release	1.0
Publisher	DOTS Consortium

Title	DOTS Project Glossary
Number	IST10258/010
Date	13/4/2000
Release	0.2
Publisher	DOTS Consortium

3.2 IEC References

IEC references can be obtained from IEC and in most cases also from ISO and national standardization bodies. In the case of IEC 61850, it is in DRAFT status. This means that the only way of obtaining it is getting in contact with the IEC technical committee in charge of it (IEC TC 57).

Title	Communications requirements for functions and device models.
Number	Draft IEC-61850 Part 5, 57/WG10/33/WD
Date	24/02/2000
Publisher	IEC

Title	Configuration language for electrical substation IEDs (Intelligent Electronic Devices).
Number	Draft IEC-61850 Part 6, 57/JTF2/2/WD
Date	11/07/2000
Publisher	IEC

Title	Basic communication structure for substations and feeder equipment
Number	Draft IEC-61850 Part 7
Date	8/11/1999
Publisher	IEC
Notes	The versions of the different parts of this document used for this mapping are:

- Part 7-1 Principles and models. Version 2. CD 08th November 1999
- Part 7-2 Abstract communication service interface (ACSI). Ed 1.0, 57/481/CD, 30th June 2000.
- Part 7-3 Common data classes. Ed 1.0, 57/482/CD, 30th June 2000 .
- Part 7-4 Compatible logical node classes and data classes. 57/WG10/38/WD Version 3.WD 7th January 2000

3.3 OMG References

OMG specifications are public specifications. This means that you can get them and use them *for free* from the OMG web site (`www.omg.org`).

Title	The Common Object Request Broker: Architecture and Specification
Number	CORBA V2.3
Date	October 1999
Release	2.3.1
Publisher	Object Management Group

Title	Real-time CORBA 1.0 Adopted Specification
Number	ptc/99-06-02
Date	June 1999
Release	1.0
Publisher	Object Management Group

Title	OMG Unified Modelling Language Specification
Number	ad/99-01-02
Date	January 1999
Release	1.3
Publisher	Object Management Group

Title	C++ Language Mapping
Number	ptc/2000-01-02
Date	January 2000
Release	-
Publisher	Object Management Group

Title	Java ™ Language to IDL Mapping
Number	ptc/2000-01-06
Date	January 2000
Release	-
Publisher	Object Management Group

Title	OMG IDL to Java ™ Language Mapping
Number	ptc/2000-01-08
Date	January 2000
Release	-
Publisher	Object Management Group

Title	Fault Tolerant CORBA Joint Revised Submission
Number	orbos/99-10-05
Date	October 25, 1999
Release	-
Publisher	Object Management Group

Title	CORBA Messaging Joint Revised Submission
Number	orbos/98-05-05
Date	May 18, 1998
Release	-
Publisher	Object Management Group

Title	Persistent State Service 2.0 Joint Revised Submission
Number	orbos/99-07-07
Date	August 2, 1999
Release	2.0
Publisher	Object Management Group

Title	Notification Service Joint Revised Submission
Number	telecom/98-11-01
Date	November 3, 1998
Release	2.0
Publisher	Object Management Group

Title	Naming Service Specification
Number	orbos/01-02-65
Date	February 2001
Release	
Publisher	Object Management Group

Title	Response to the OMG RFP for Schedulability, Performance and Time
Number	Ad/2000-08-04

Date	August 14, 2000
Release	1.0
Publisher	Object Management Group

4. Assessment of 61850 & CORBA

4.1 Introduction

This document contains a mapping between two somewhat different perspectives of distributed systems. This chapter addresses some specific issues of the mapping that were not covered in the DOTS General Model Definition Document.

This mapping is based mainly on Part 2 of the document Basic communication structure for substations and feeder equipment (Draft IEC-61850 Part 7[3]).

From a basic and fundamental perspective, the mapping to CORBA should be immediate because the IEC61850 claims to be object oriented. This is not as straightforward, however, because the Draft IEC-61850 Part 7 is not as object-oriented as it may look at the very beginning. Some quirks will be necessary to provide a clean CORBA view of the SAS domain addressed by IEC 61850. These quirks, however, will clarify some of the possible interpretations of the IEC 61850 Part 7 series, getting a clearer OO picture of it.

IEC 61850 Part 7 series is structured in four documents:

1. Part 7-1 Principles and models.

   This section gives an overview on the architecture of the co-operation and communication between substation devices like substation hosts, protection devices, breakers, transformers etc. The architecture and part of the specifications are applicable to many other applications outside the substations, but the main focus is to meet the requirements for communication in substations as defined in part IEC 61850-5.

2. Part 7-2 Abstract communication service interface (ACSI).

   This part of the standard IEC 61850 defines the common service models (abstract communication service interface, ACSI) for use in the utility substation and any other real-time co-operation of any kind of field device. It provides the detailed specification of what kind of objects are available and how to communicate with these objects. The ACSI has been defined to be independent of the underlying protocol. Specific

[3] The versions of the different parts of this document used for this mapping are also different because the IEC 61850 Part 7 document is progressing in four parts. Specific numbering of the versions used for this version of the mapping to CORBA can be found in section 3.2 of this document (IEC References).

communication service mappings (SCSM) are specified in part 8 (station bus) and part 9 (process bus) of this standard.

3. Part 7-3 Common data classes.

This part of IEC 61850 defines attributes and attribute class definitions for classes of data objects relating to substation applications. The definitions are based on existing or emerging standards and applications. In particular the class definitions are based upon:

- *the specific data types defined in IEC 60870-5-101 and IEC 60870-5-103.*
- *the common class definitions from the Utility Communication Architecture 2.0: Generic Object Models for Substation & Feeder Equipment (GOMSFE).*

4. Part 7-4 Compatible logical node classes and data classes.

This part of IEC 61850 defines logical node classes, data object classes and their relationship in the context of substations and feeder equipment.

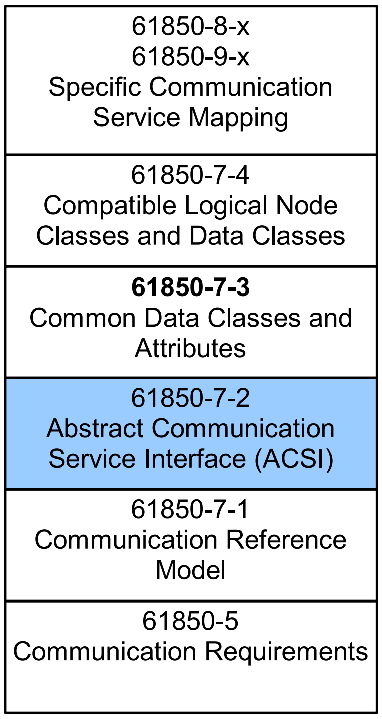

61850-8-x
61850-9-x
Specific Communication Service Mapping
61850-7-4
Compatible Logical Node Classes and Data Classes
61850-7-3
Common Data Classes and Attributes
61850-7-2
Abstract Communication Service Interface (ACSI)
61850-7-1
Communication Reference Model
61850-5
Communication Requirements

Figure 4-1: IEC 61850 Parts.

This document IST10258/009 addresses specifically the mapping of the ACSI (Part 7-2) to CORBA. This mapping is obviously based in other parts of the IEC61850 Draft standard (See Figure 4.1).

4.2 The IEC 61850-7 Object Model

4.2.1 The basic model

The IEC 61850 Part 7-2 specifies a collection of abstract interfaces used to interoperate with objects that constitute an application. These objects are instantiated from classes that are specified in sections 7-3 and 7-4 of the

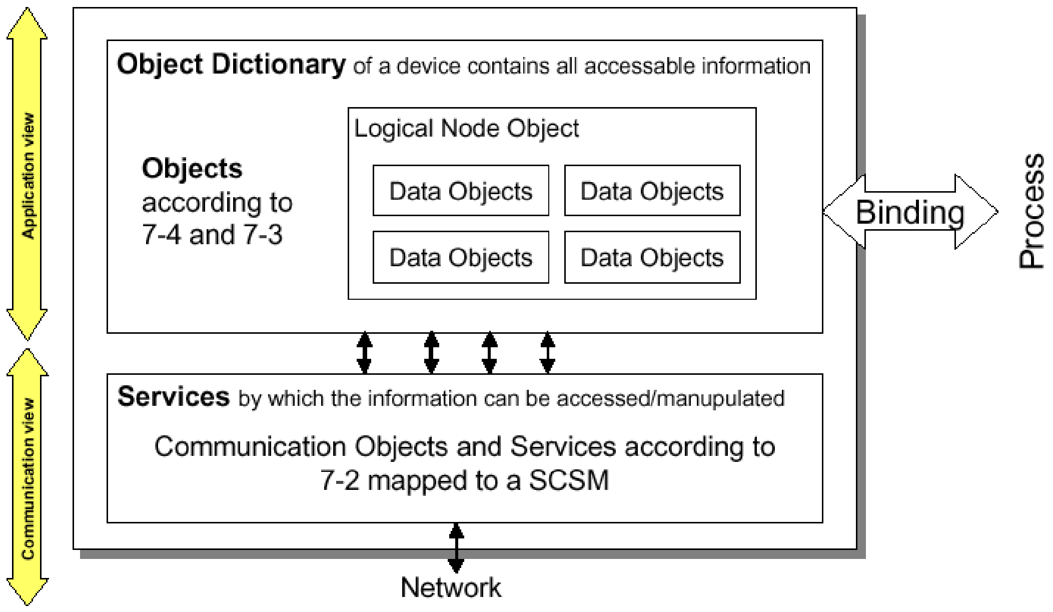

Application view
Communication view
Object Dictionary of a device contains all accessable information
Objects according to 7-4 and 7-3
Logical Node Object
Data Objects
Data Objects
Data Objects
Data Objects
Binding
Process
Services by which the information can be accessed/manupulated
Communication Objects and Services according to 7-2 mapped to a SCSM
Network

Figure 4-2: Object Dictionary and Services of a device [From IEC 61850 Part

specification.

The software view of this model is the following:

We have an application running in a computer that is composed by a hierarchically-organized collection of objects and we have an external interface that should provide the capability of access/manipulating those objects.

The objects can be any object following 7-3 and 7-4 and the access services are specified in 7-2.

4.3 Problems with the IEC 61850 Model

4.3.1 Introduction

Apart from minor –reparable by careful editing– errors and errata, there are some more fundamental problems with the IEC 61850 draft standard that makes difficult the construction of the mapping.

4.3.2 The model is not finished

As was said before (Section **¡Error! No se encuentra el origen de la referencia.** *¡Error! No se encuentra el origen de la referencia.*) the model is not yet finished (at least the documents are not finished) and there are some pending issues to be

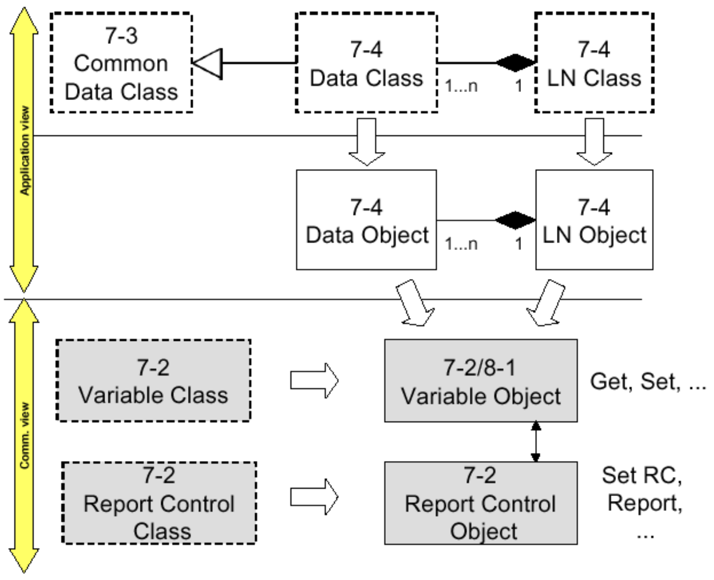


Figure 4-3: Application and communication classes and objects [IEC 61850-7-1 page 15].

addressed, mainly cleaning, consistence and completion. There are parts still missing and others that need some rework.

4.3.3 Inconsistencies

Du to the draft status, the document set present some inconsistencies that make difficult the integration of all ideas in a common model. As an example, the next figure (from Part 7-1) shows the global relations between what the different parts specify. It uses the term ***variable object*** to refer to the collection of entities used in the communication view, and derived from what is said in part 7-2. But this part never uses this term to refer to anything, instead it proposes a model more related with the CORBA view.

We expect that from the collaboration of the DOTS consortium in the IEC task force most of these inconsistencies will dissapear through the real implementation of the standard (this is the real test of any specification).

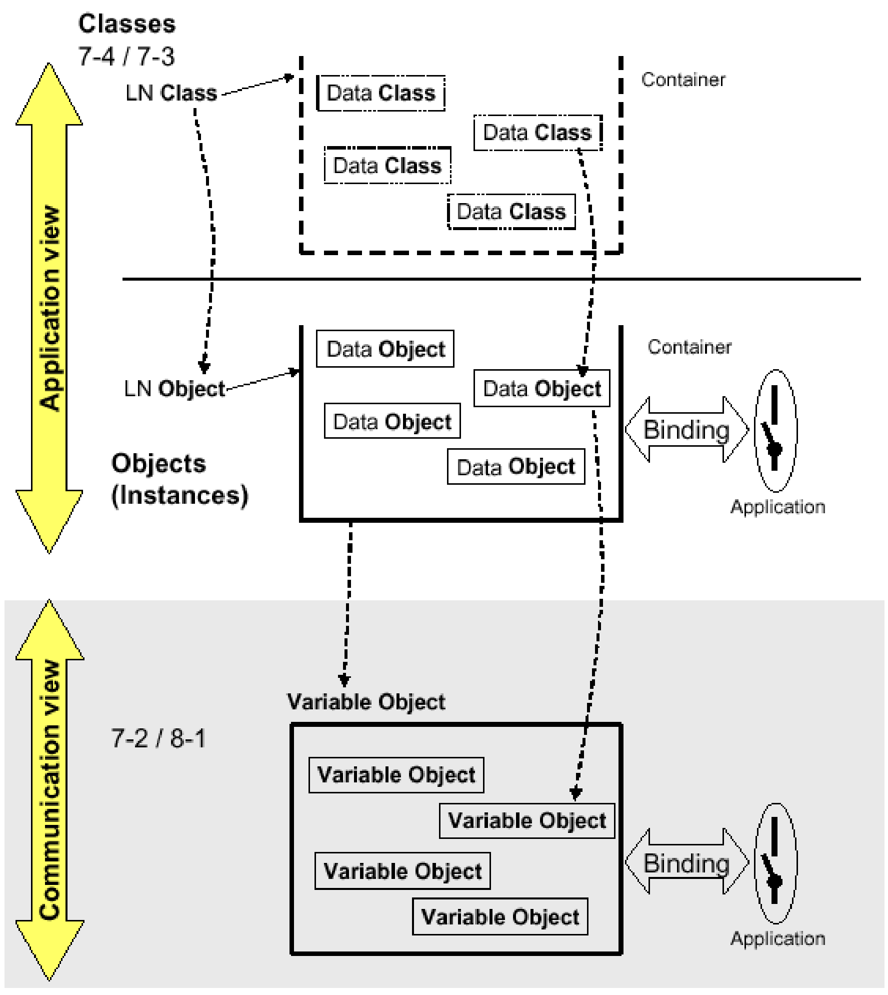


Figure 4-4: Application to communication mapping [IEC 61850-7-1 page 16].

4.3.4 The partial-OO nature of the ACSI

The model is not pure OO (and hence the need of a mapping to CORBA). One question emerges all the time:

What is the reason for separating object definition (7-3, 7-4) from object interfacing (7-2) ?

This separates in two different specifications what should be in a single one. This is due perhaps to the procedural oriented perspective that has guided IEC 61850 formulation:

- We have *data structures* (7-3 and 7-4) that can be referred-to using the word *class* or *object* but that does not make them objects.
- We have *access methods* (7-2) for these data

This is the classical separation done in procedural programming technology. Sections 7-3 and 7-4 describe a database and section 7-2 an abstract API to access it.

4.3.5 Violations of encapsulation

One of the ten commandments of object orientation and specially important in CORBA is
encapsulation. This is violated sometimes when some objects can provide services that refer to inner state of other objects.

Draft 61850-7-1 page 12

As depicted in Figure 4 all application-specific information of a device (information according to 61850-7-4 and -7-3) that can be accessed using the services defined in 61850-7-2 build an ***object dictionar****y.*

4.3.6 Evolution in the specification perspective

The draft standard has had a changing perspective that has been going progressively towards object orientation and separation of different componential specifications. This has been good for us but the process hasn't concluded and some oldies still remain in the text.

This evolution has had bad consequences for our mapping in some cases. For example:

Draft 61850-7-1 page 14

[Editor's Note 1 – The main purpose of 7-3 and 7-4 are the class definitins while the 7-2 services - when applied in an SCSM - deal with application objects. On the other side 7-2 uses also an object oriented approach describing its own services. 7-2 has also classes and objects. But the classes of 7-2 are mainly independent of the classes of 7-3 and 7-4. One mistake in the past was, to combine these two worlds of classes. Another was the mix of classes and objects that caused many disconnects in the past. Hope that this draft reflects the many discussion we had so far.
I guess we have now a clear understanding that 7-4 and 7-3 define content while 7-2 provides in conjunction with a SCSM the tool. Both are important - but we must clearly distinquish both.]

We cannot understand why combining data and access method is a *mistake* from an object-oriented perspective. The main work of this mapping is to combine-back these, now separated, parts of the standard.

In chapter 5 we will explain how the data expressed in 7-3 and 7-4 can be accessed using 7-2 by means of the construction of a collection of CORBA compliant objects.

4.3.7 Loose Typing and Unnecessary polymorphism

The ACSI is build without a strong typing policy, specifying polymorphic interfaces that are difficult to map and unnecessarily complex if done with strict adherence to the text of the specification.

Some operations (mainly directory operations like `LogicalNodeDirectory` in page 35) are polymorphic in their specification in the ACSI. This approach is not very sensible in most cases , because when a client requests one of these operations, it expects a non polimorphic return result and it should be better to specify a non-polimorphic operation for each return type.

4.4 CORBA Basics

4.4.1 Introduction

The best way to learn about CORBA technology and how to use it is to get a good book. The OMG web site (`www.omg.org`) suggests a good collection of literature.

In this section we will assess the CORBA technology from the perspective of the mapping, introducing some CORBA elements that will be used in the mapping. Complete information about these elements can be found in the CORBA specification.

The mapping will be based in the definition of CORBA objects and the specification of architectures and deployment policies that will let developers build IEC 61850 application over a CORBA middleware.

4.4.2 Object Request Forwarding

The central idea of CORBA is the transparent access to remote services provided by objects.

This transparency is achieved by means of the use of:

- An Interface definition language that enables the clean coupling of clients and servers.
- Client and server stubs that are generated automatically by the IDL compiler.
- An interoperability protocol tan provides the necessary support for communcation.

This makes possible the deployment of objects over heterogeneous platforms (HW-OS) and using heterogeneous languages.

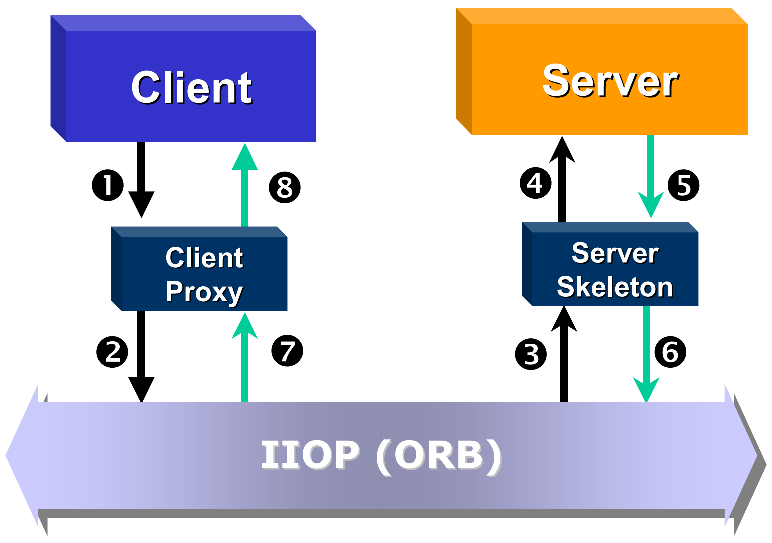


Figure 4-5: Object request forwarding process.

4.4.3 Objects, Interfaces, Servers, Servants, POAs and more

CORBA objects are incarnated by pieces of run-time code called servants. Servants can be built in any programming language supported by the IDL compiler.

Object servants are controlled by POAs and POAs are managed by POA Managers. They are the entities that control the flow of requests to servants.

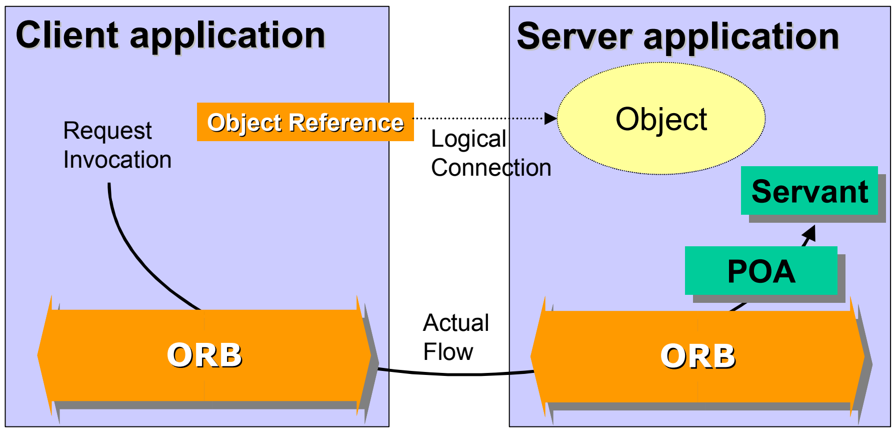


Figure 4-6: CORBA view of client server interaction.

Any POA manager can be in any of four possible processing states; *active*, *inactive*, *holding*, and *discarding*. The processing state determines the capabilities of the associated POAs and the disposition of requests received by those POAs. Figure 4-7 illustrates the processing states and the transitions between them. A POA manager is created in the *holding* state and only upon activation it can proceed with request handling.

4.4.4 OMA Services

CORBA applications can be built from scratch or exploiting prebuilt entities. Some of the most important ones are specified by the OMG and constitute what is called the Common Object Services or CORBAservices.

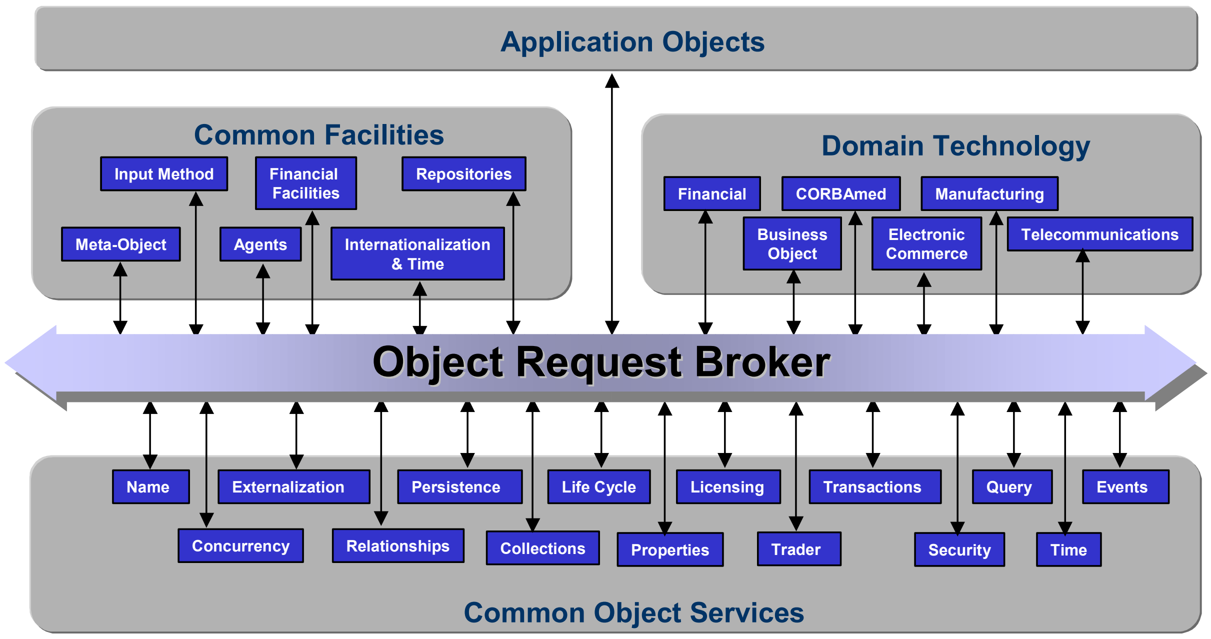


Figure 4-8: The Object Management Architecture provides a structured approach to component/interface organization.

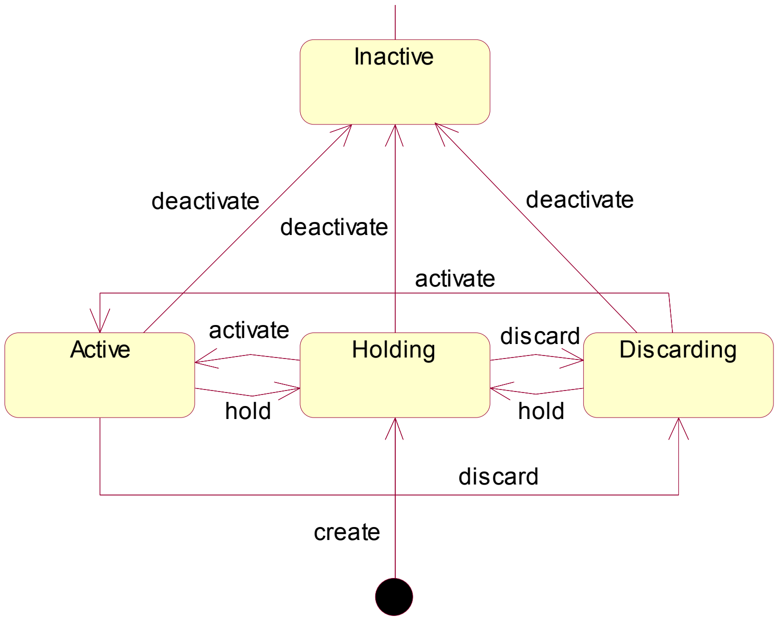


Figure 4-7: A POA Manager can be in four states: holding, active, discarding and inactive.

Examples of these services of special importance for the mapping are:

- Naming service
- Time service
- Event service
- Security service

4.4.5 Object IDs, Object References and Object Names

CORBA uses several types of entities to refer to objects in CORBA applications. We will consider three of these elements:

- **Object References** (also called Interoperable Object References or IORs): are the equivalent to a pointer but with distributed scope. It is the resource used to invoke a specific service of an object [See CORBA 2.3.1 Section 13.6]. Any object reference includes a field (the object ID) that is used by the POA to locate the servant that incarnates an object. IORs are the cornerstone of location transparency.
- **Object Names:** Are used to query a Name Service [See CORBAservices Section 3] in order to get an IOR for a specific object[4]. Names are used to give objects meaningful names that can be simpler to handle by humans (for example, by programmers).
- **Object IDs** (OIDs): Are used by the POA to identify the object (among those managed by that POA) that is addressed in a specific request. It is extracted from the IOR.
- **Object Keys** (OKeys): Are the field of the IOR that contains the OID.

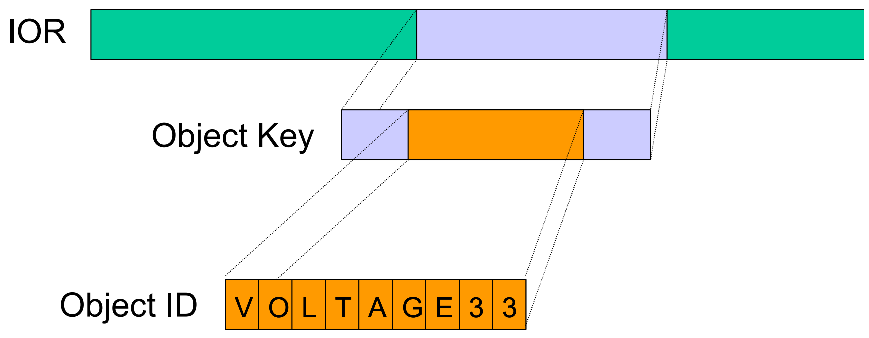


Figure 4-9: Containment relation between the interoperable Object Reference (IOR) the Object Key (OKey) and the Object ID (OID)

The Object Reference (IOR) is the cornerstone of CORBA interoperability because it contains all the information needed to locate an object and forward a request to it. The Name is only used to query the Name Service to get the Object Reference.

[4] The Name Service stores name-to-reference associations

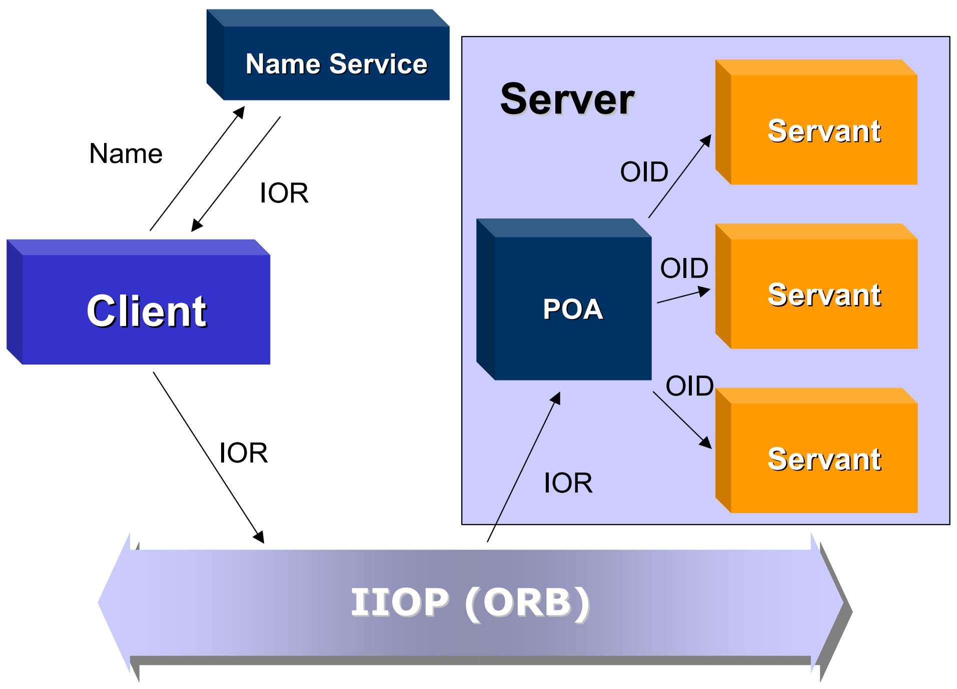


Figure 4-10: Relations between different types of object references, identifiers and names.

4.4.6 The CORBAservices Name Service

This service will be used by the mapping to handle DOTS object naming and localization.

5. Mapping Details

5.1 Introduction

This chapter addresses some issues relevant for the understanding of the mapping. As is mentioned before the true nature of IEC 61850 is not pure OO, and some deviation will appear (in fact this is the reason why the mapping is necessary). The next section analyzes the different alternatives we have considered for the mapping.

This mapping addresses the definition of the following ACSI services (defined in Part 7-2) using CORBA technology.

Server model:
ServerDirectory

Logical device model:
LogicalDeviceDirectory

Logical node model:
LogicalNodeDirectory

Data object model:
GetDataObjectValues
SetDataObjectValues
GetDataObjectDefinition
DataObjectDirectory

Data attribute model:
GetDataAttributeValues
SetDataAttributeValues

Data set model:
GetDataSetValue
SetDataSetValue
CreateDataSet
DeleteDataSet
DataSetDirectory

Publish and subscribe data transfer model:
Report
GetReportControlValue
SetReportControlValue
CreateReportControl
DeleteReportControl
GetLogControlValue
SetLogControlValue
GetLogStatusValue
QueryLog
EmtyLog

GOOSE model:
Activate
Deactivate
GetGOOSEControlValue
SetGOOSEControlValue

ControL model:
Select
SelectWithValue
Cancel
Operate
CommandTermination
Synchrocheck
TimeActivatedOperate

Substitution model:
Substitute
UnSubstitute

Transmission of sampled values model:
GetSampledMeasuredValuesControlValue
SetSampledMeasuredValuesControlValue

Time synchronisation model:
Prepare
Measure
Synchronise

FILE transfer model:
GetFile
SetFile
DeleteFile
FileDirectory

Mapping the ACSI to CORBA means mapping this Abstract Communication Interface to the interface specification provided by CORBA, *i.e.* CORBA IDL. This mapping contains this specification and, beyond that, some rationales of the decisions taken and some information relevant for the interpretation and application of the mapping.

The basic idea is to provide an adapter to a CORBA infrastructure that can present an ACSI-like interface for an application accessing remote IEDs (See figure 5-1).

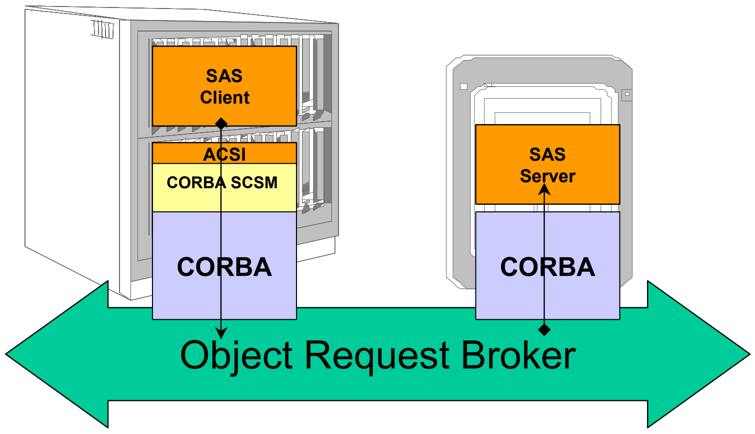
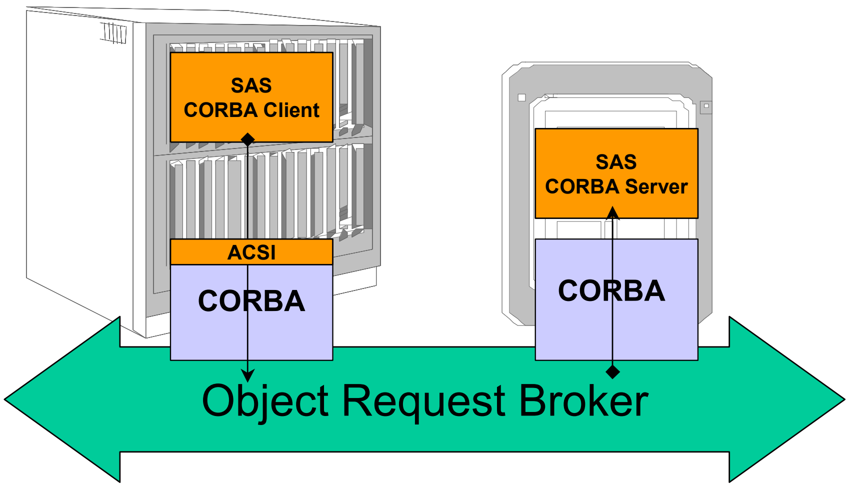


Figure 5-2: Providing ACSI over a CORBA infrastructure.

As we shall demonstrate in this mapping, CORBA fits so well with the ACSI specification that it can be provided directly as a CORBA interface. ACSI objects can be implemented as CORBA objects providing remote access interfaces that are



compliant with the ACSI (See Figure 5-2).

This means that a service invocation through the ACSI will be done as a CORBA request forwarded through a local proxy of the the remote server. This is done using a CORBA reference to the remote server:

```
Result = Remote_server->service_requested(input paramenters);
```

There is an intrinsic asymmetry between CORBA servers and clients, but this asymmetry is only related with a specific request. In most real-time control applications objects tend to be both client and servers simultaneously (obviously for different interactions). The CORBA stacks for clients and servers differ but they are automatically generated by the IDL compiler and developers should be only partially aware of these issues.

5.2 How to map

As was mentioned before, the ACSI (an the rest of the IEC 61850) is not fully OO. This raises several questions about hot to map "non-pure-OO" to "OO".

There are, from our point of view, three approaches to the mapping to CORBA:

- Consider that there is only one CORBA object per ACSI server (coarse grained mapping)
- Consider that other lower-level entities (for example, logical nodes) should be also built as CORBA objects (medium grained mapping)
- Consider that all encapsulable entities should be CORBA objects (fine grained mapping)

All three approaches are analyzed in the following sections.

5.2.1 Coarse grained mapping

In this approach there are only one type of CORBA objects, namely what the ACSI calls a ***server***. Figure 5-2 shows the IEC 61850 view of the interaction between client and server.

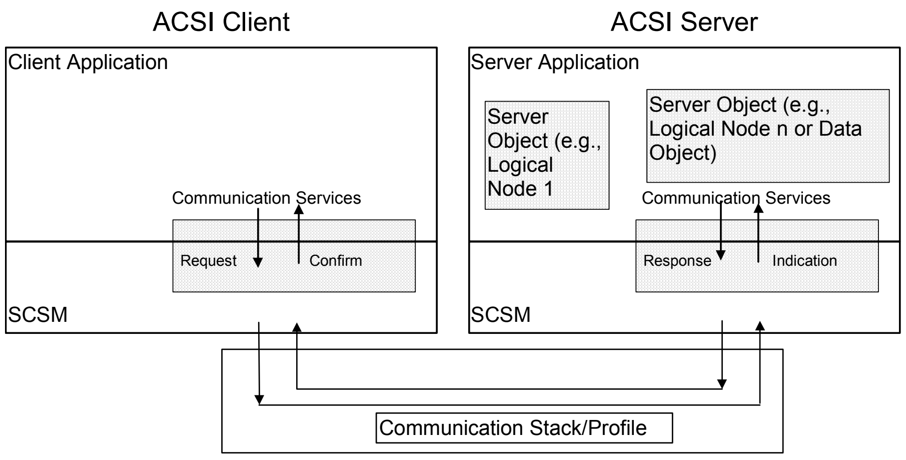


Figure 5-3: ACSI view of client-server interaction.

The approach of the ACSI is that communication associations are done between a client and a server, so any ACSI Based request is sent to the server even when it is specified as interface to a lower level component of the server.

In CORBA, associations operate the same way (even the name "***server***" is the same) but logical associations are done object to object and server activity is transparent[5].

5.2.2 Medium grained mapping

In the medium grained mapping, CORBA objectification reach the level of entities that appear in the ACSI specification (server, logical device, logical node, etc.). Each one of these elements will be a CORBA object and interaction will be possible between any pair of them.

5.2.3 Fine grained mapping

In fine grained mapping every entity based on IEC 61850 is defined as a CORBA object. This means modeling more than six hundred classes as IDL interfaces (being most of them pure data structures).

So the level selected for the mapping is the medium grained because:

- It fits best the ACSI specification
- It is manageable

5.3 Basic mapping

The procedure of the basic mapping is the automatic translation of ACSI specifications to CORBA IDL specifications. There will be, however, special situations where this automatic translation will not be possible. In these cases there will be two possibilities:

- A special mapping will be done.
- No mapping will be done (*i.e.* the feature will not be supported by the mapping).

The ACSI uses ASN.1 syntax in the specification of class members:

Brief ASN.1 rules for interpreting the object formalism:

*Each field has a field identifier possibly followed by a name. The field identifier begins with an '&' and is followed by a **reference** name, beginning with either a lower case or an upper case letter.*

- *If the field identifier begins with '&' **Upper case** letter:*
  - *If there is no following name, the field identifies a type.*

[5] Obviously to the extent the programmer decides, because it is possible for a programmer to modify policies that affect server behavior. This is done mainly through the `PortableServer` interface (POA).

    - *If the following name is mixed case and begins with an upper case letter, or if the following name is upper case and the name of a Universal type, the field identifies a value set.*
    - *If the following name is upper case and the name of an Object Class, the field identifies an Object Set.*

- *If the field identifier begins with '&'* ***lower case*** *letter:*
    - *If the following name is upper case and the name of an Object Class, the field identifies an Object (instance).*
    - *If the following name is mixed case and begins with an upper case letter, or if the following name is upper case and is the name of a Universal type, the field identifies a value.*

The basic mapping maps these entities to CORBA entities:

ACSI ASN.1	CORBA IDL
Type	Type
Value set	Sequence
Object Set	Sequence
Object instance	Interface
Value	Attribute

5.4 Fundamental data types mapping

The mapping of fundamental types is done as the following table indicates:

ACSI Data Type	CORBA Data Type
`OCTET STRING`	`sequence <octet>`
`VisibleString`	`string`
`BIT STRING`	`sequence <boolean>`[6]
`INTEGER`	`long`
`REAL`	`float`
`Sequence`	`struct`
`Array`	Array
`ENUMERATED`	`enum`
`NULL`	`Nil`

5.5 Mapping of common types

5.5.1 ObjectName and ObjectReference

An ACSI ObjectName is an string. The easy way is to map it to string:

```
typedef string ObjectName;
```

[6] Mapping a bit string to a sequence of boolean is not very effective, but it is necessary because ACSI bit string specifications are open-ended and this makes impossible the determination of a shrink-wrapped container for them.

But ObjectReference is confusing because is has a complete different meaning in ACSI and in CORBA. In ACSI it is a complete hierarchical name (See 5.6 Object Naming). In CORBA it is an Interoperable Object Reference.

In this mapping we will use both things: object names and object references. Both ACSI entities are names that are mapped to a `CosNaming::name`. If a user wants the equivalent of the ACSI ObjectName (i.e. the name without the path) he can get it from the last name component of the `CosNaming::name` name of the object. We will use the type `ACSI::ObjectName` for it.

```
module ACSI{
    typedef CosNaming::name ObjectName;
};
```

5.5.2 TimeStamp

CORBA does not support 6 octet integers so the mapping can be done to an array of octects or to an integral type of size enough. We have selected here an `octet` array but it could also be a `long long` (64 bit)[7]:

```
enum TimeClass {milliseconds,
                hundred_microseconds,
                ten_microseconds};
typedef sequence <Boolean> TimeQuality;

typedef long Nanosec;

struct TimeStamp{
    octect time[6];
    TimeClass timeCl;
    TimeQuality timQ;
};
```

5.5.3 RelativeTime

```
typedef long long RelativeTime;
```

[7] Type `long long` appeared in CORBA 2.3.

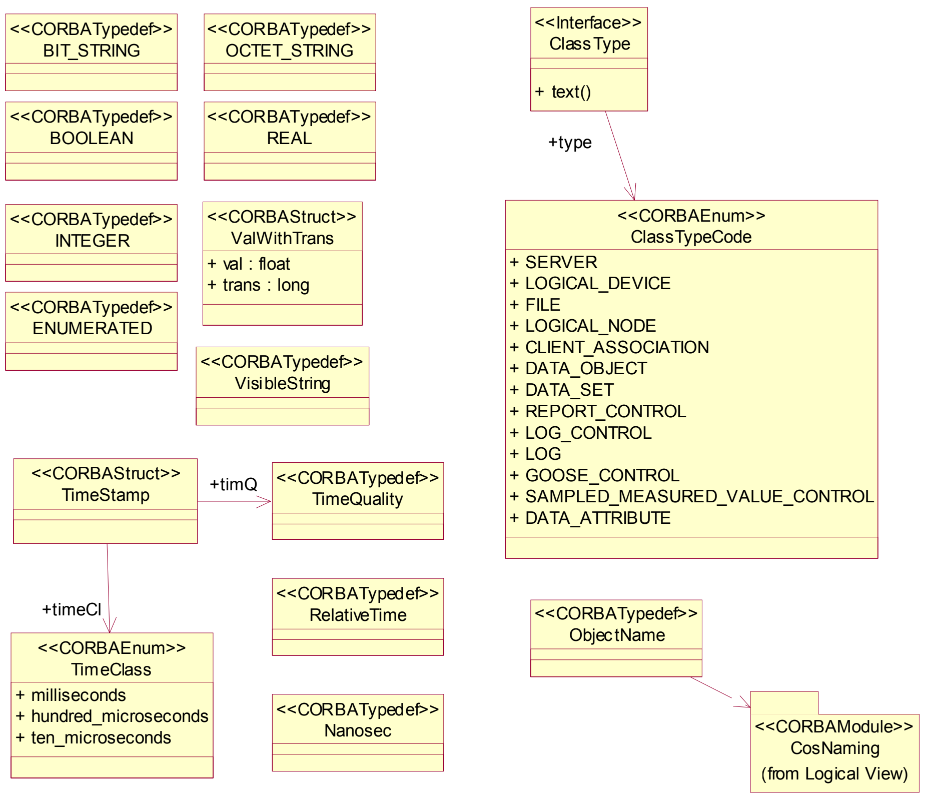


Figure 5-4: Some basic classes of the mapping.

5.6 Object Naming

In this mapping, the central mechanism for object interaction and hierarchy construction is the CORBA Object Reference. To handle object names described by IEC 61850 the CORBA Name Service will be used.

In this mapping we will use DOTS-specific object identifiers (`ObjectId`) by means of the `PortableServer::IdAssignmentPolicy` (with `USER_ID` value). This will enable us to generate OIDs strings using the object naming policy expressed in IEC 61850 Part 7-2, section 11.4.2.1 (object naming). Each class in the class model of IEC 61850-7 has its own class name that has server scope. These class names are the basic building blocks when defining names of objects (instances). The name structure for the whole path-name of an IEC 61850 object is as follows [8]:

<LD instance name> / <LN instance name>.<Data object instance name>.<Data attribute instance name>

[8] LN = logical node, LD = logical device.

These ACSI object path-name has server scope because the ACSI focuses in a one-to-one association between server and client; so, the server name should be added to DOTS naming to provide wider scope.

Naming schemas specified by the ACSI are:

Logical device specific scope is defined using the "/" by up to 32 characters.
Example: / myXCBR1 . Pos1 . stVal

Server specific scope is defined as up to 32 characters, then "/" followed by up to 32 characters.
Example: my-feeder-XY-logical-device / myXCBR1 . Pos1 . stVal

Association specific scope is defined using "@" followed by up to 32 characters.
Example: @ myXCBR1 . Pos1 . stVal

This mapping forbids the use of shell metacharacters (In DOTS we are going to use UNIX shells and MS Windows shells) in any of these names to avoid problems in the use of command line tools for object name management or with heterogeneous Name Service implementations.

These names will be used to register DOTS objects in a federated name service. The name bindings will use IEC 61850 names as `NameComponent.id` fields and IEC 61850 object class names as `NameComponent.kind` fields (See ACSI IDL).

```
struct NameComponent
{
  Istring id;     // The name
  Istring kind;   // More info
};
```

For example the name `my-feeder-XY/myXCBR1.Pos1.stVal` running on server `process-AX` will be registered using the `CosNaming::name` shown in the following table:

CosNaming::name	
id field	**kind field**
Process-AX	SERVER
my-feeder-XY	LOGICAL_DEVICE
myXCBR1	LOGICAL_NODE
Pos1	DATA_OBJECT
StVal	DATA_ATTRIBUTE

Naming contexts can be run in the different servers. The root naming context should be run on a high availability system. A reference to the root naming context can be obtained using the `resolve_initial_references()` operation:

```
CORBA::Object_var obj;
Obj = orb-> resolve_initial_references("NameService");

CosNaming::NamingContext_var root_context;
root_context = CosNaming::NamingContext::_narrow(obj)
```

```
assert(!CORBA::is_nil(root_context));
```

From the plaint interpretation of the ACSI this structure should implicate that only two levels of naming contexts are possible:

- Root naming context
- Server level naming context

This is not true because using CORBA distributed model logical-nodes are no longer physically but logically attached to servers (the same for the rest of the CORBA objects). This means that server `process-AX` can be running in a computer at the substation level and the logical device `my-feeder-XY` can be running at a different RTU. ACSI servers can be mapped to CORBA servers but this is not a one to one relation (we can have several ACSI servers running in a

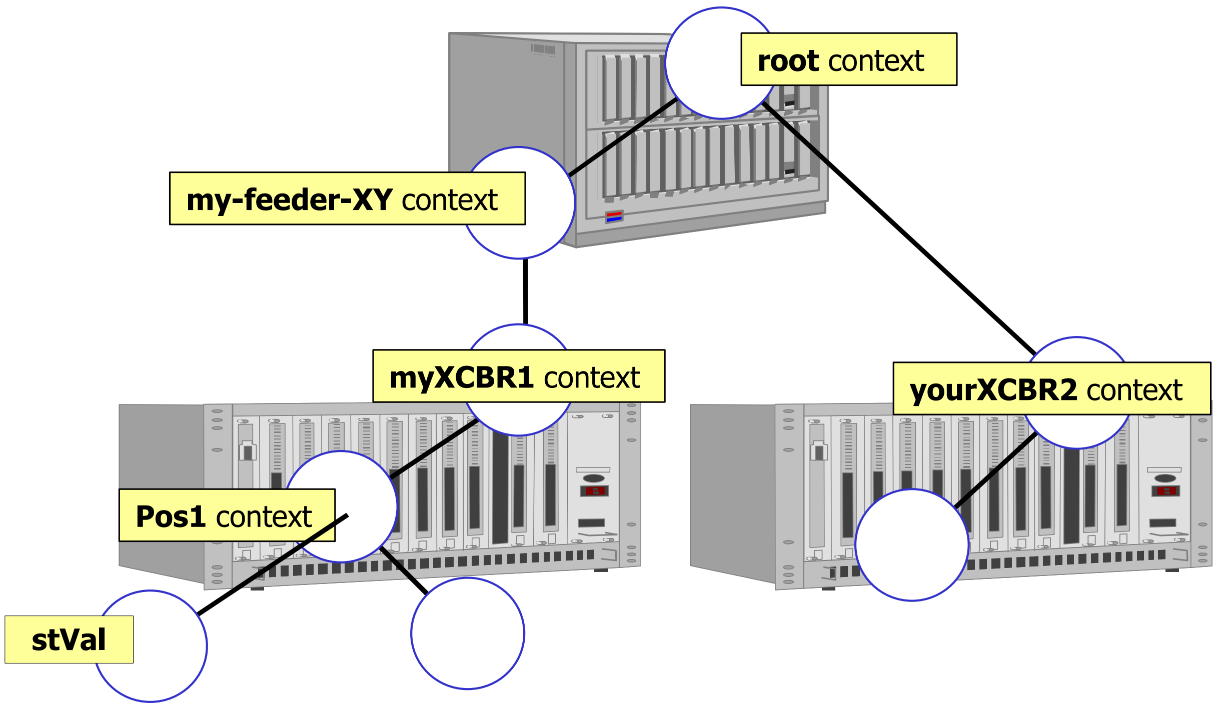


Figure 5-5: Naming contexts trees.

CORBA server and CORBA servers that are not running any ACSI server).

5.7 Polimorphic operations

Consider for example the operation:

```
??? LogicalNode::LogicalNodeDirectory(in ObjectClass class)
```

That can provide seven types of results depending on the argument (See ACSI page 37).

The mapping can be done to a non-polimorphic set of operations (clear), to a polimorphic operation with increased complexity in types definition (strictly compliant) and to a single operation based on `CORBA::Any` (neat).

Polimorphic	Non Polimorphic	
`ListOfObjectNames LogicalNodeDirectory(ObjectClass class);`	`DataObjectList`	`DataObjectDirectory ();`
	`DataSetList`	`DataSetDirectory ();`
	`ReportControlList`	`ReportControlDirectory ();`
	`LogControlList`	`LogControlDirectory ();`
	`LogList`	`LogDirectory ();`
	`GOOSEControlList`	`GOOSEControlDirectory ();`
Using `CORBA::Any`		
`CORBA::Any LogicalNodeDirectory(ObjectClass class);`		

In this last case, the client obtains a `CORBA::Any` as result and extracts from it the desired list because it knows the type of directory requested:

```
// Variables to receive the CORBA::Any
CORBA::Any any;

// Variable to extract the List from the CORBA::Any
ACSI::DataSetList dsl;

// Request invocation
any = LN->LogicalNodeDirectory(DATA_SET);

// Extraction and use
if (any >>= dsl) {
   // Use the list returned
} */
```

It is also possible to use the `DynAny` interface to determine in run time the type of the contents.

5.8 Pull Interfaces for long lists

There are many operations of the ACSI that return lists of entities (open ended sequences in IDL). If these lists are small, the classic interface definition will suffice. For example in the definition of the logical device class, the following `LogicalDeviceDirectory()` operation is used to request a list of logical nodes in that logical device:

```
typedef sequence <LogicalNode> LogicalNodeList;
LogicalNodeList LogicalDeviceDirectory ();
```

If the list of logical nodes to be returned is very long this operation can exhaust systems resources. If it is the case for any operation, this effect can be avoided/controlled by means of the use of iterator-based interfaces. For example, using a pull iterator:

```
typedef sequence <LogicalNode> LogicalNodeList;
```

```
interface LogicalNodeIterator{
  LogicalNodeList next();
  void destroy();
};

interface LogicalDevice{

LogicalNodeList LogicalDeviceDirectory(out LogicalNodeIterator it);
  // . . .
};
```

The operation LogicalDeviceDirectory() returns the LogicalNodeList, but, if it is going to be too big, the operation only returns part of the total list and an iterator in the it variable to continue requesting the rest of the list in successive requests.

5.9 CORBA identifiers

CORBA identifiers cannot contain hyphens. When hyphens appear they have been mapped to underscores.

5.10 Mapping of enumerated types

Enumerated types in the ACSI do not follow a common notation . There are collection of identifiers in UPPERCASE, in lowercase and in mixed case. With or without hyphens.

5.11 Mapping of optional fields

Strongly typed languages are not very good at managing optional fields. In the ACSI specification there are lots of these optional fields. The common IDL mapping of optional fields is done by means of unions. For example:

```
union QoS_Control switch(boolean){
case TRUE:
   unsigned long reservedBandwidth;
}
```

The union `QoS_Control` contains a field `reservedBandwidth` only if the discriminator is TRUE.

This is mostly not done in this mapping to avoid cluttering the IDL and making it incomprehensible.

5.12 ACSI/CORBA communication service model

CORBA based application will be deployed over computational equipment in the same way as was specified by the ACSI. This can be done easily because CORBA object functionality exactly matches the server object model proposed by the ACSI[9].

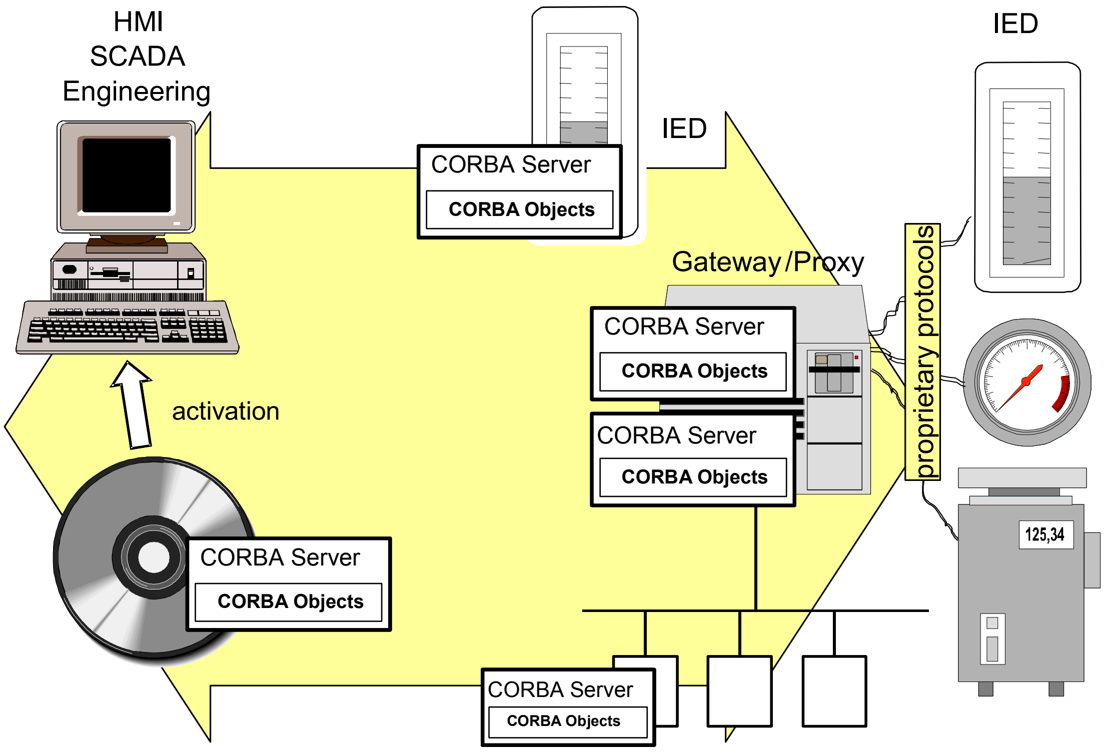


Figure 5-6: ACSI/CORBA Servers will reside where ACSI servers where planned. The only difference is the change of point-to-point interactions/associations to a multipartite relation provided by the broker.

There are four communication mechanisms used in the ACSI that can be mapped easily to CORBA mechanisms:

ACSI	CORBA
Request/response	Request
Request/no response	Oneway request
GOOSE message	Canonical push event channel
Sampled value transmission	Push/pull real-time event channel

Normal (two-way) requests and `oneway` requests are implemented by the brokering mechanisms. Event channels are provided by event services and there are three main alternatives:

- The CORBA Event Service provides event channels but not filtering and real time performance

[9] In fact, it is worth noting that both specifications use the name "server" to refer to very similar entities.

- The CORBA Notification Service provides event channels with filtering (and much more functionality) but not real-time performance

- Custom real-time event services have been demonstrated to provide good enough performance for fast applications.

In relation with the `oneway` invocation semantics for CORBA requests it is worth noting that what the CORBA specification says is that it does not mean non-blocking in the client side but a best effort semantics in the transmission of the message. This means that *compliant* brokers can block the client or even drop the request without notification.

Depending on the tightness of the timing requirements it can be necessary to use proprietary transports different from the standardized interoperable protocol (IIOP).

5.13 Recommended reading

This mapping is based on a large collection of document and its understanding presupposes deep knowledge about the two specifications involved: IEC 61850 and CORBA. This knowledge can be gained reading all the specifications used, but gathering this information is not an easy task because of two different reasons:

- IEC 61850 is *evolving* because it is a draft (in some aspects it is a very early draft). It is not easy to get a complete, static view of the specification because each part is progressing at a very different pace and is being done by a different collection of people.

- While CORBA is consolidated, it is still evolving and there are lots of specifications that are very relevant for DOTS but difficult to track because they are still moving targets. The reason is that OMG work is divided in several groups related with core technology (CORBA itself), specific platforms (like real-time platforms) and specific domains (like utilities or telecoms) that are making specifications in a continuous procedure.

Even when we have tried to produce a self-contained document, to understand this mapping it is necessary to have a rather good knowledge about DOTS, IEC 61850 and CORBA. Mandatory readings are:

- DOTS General Model Definition
- OMG CORBA 2.3
- IEC 61850 Part 7: Basic communication structure for substations and feeder equipment

The full collection of recommended readings follows in three different collections:

- DOTS Documents
- IEC/ISO Documents
- OMG Documents

5.13.1 DOTS Recommended literature

- Project Programme[10]
- Project Glossary
- General Requirements Specification
- General Model Definition
- Definition HW and SW

5.13.2 IEC/ISO Recommended literature

- IEC 61850 Part 1: Basic Principles
- IEC 61850 Part 5: Communications requirements for functions and device models.
- IEC 61850 Part 7: Basic communication structure for substations and feeder equipment
- ISO/IEC 8824 Part 1: Abstract Syntax Notation One (ASN.1) - Specification of Basic Notation
- ISO/IEC 8824 Part 2: Abstract Syntax Notation One (ASN.1) - Information object specification

5.13.3 OMG Recommended Literature

OMG organizes the work in different task forces

- ORBOS (ORB & OS)
    - CORBA 2.3
    - CORBA Messaging Joint Revised Submission
    - CORBAservices: Common Object Services Specification
        - Naming service
        - Trading Service
        - Event Service
        - Time Service
        - Security Service
    - Persistent State Service 2.0
- PTC (Platform Technology Committee)
    - Real-time CORBA 1.0 Adopted Specification
    - C++ Mapping
- Telecom (Telecom Domain Task Force)
    - Notification Service Joint Revised Submission
    - Telecom Log Service

[10] Some of the recommended literature can be available only to speficic groups of readers. The SCSM to CORBA to be plublicly released should contain no references to non-publicly available documents.

- AD (Analysis and Design Task Force)
    - OMG Unified Modeling Language Specification

Part 2
The mapping

6. Mapping of Server Model

6.1 Server class definition

6.1.1 ServiceAccessPoint

This attribute is intended to be used as system wide identification of the server. In CORBA applications this functionality is achieved by means of the IOR of the server. This IOR is generated by the POA upon object creation (unless persistence policy is used) and can be obtained by means of the C++ IDL compiler-generated `_this()` operation.

We will use an `ACSI::ObjectName` type for this identification field:

```
Module ACSI{
   typedef CosNaming::name ObjectName;
. . .
}
```

6.1.2 LogicalDevices

The collection of logical devices in the server.

6.1.3 Files

File objects in the server.

6.1.4 ClientAssociations

See next chapter.

6.2 Server services

The server `ServerDirectory` service returns a list of objects in the server. `ClassTypeCode` LOGICAL_DEVICES, FILES and CLIENT_ASSOCIATIONS can be requested to this service.

The service Response+ is a type Any containing the list of requested entities (see sections 5.7 and 5.8 in this document).

The service Response- is mapped to two exceptions.

6.3 Access control

Access control can be implemented straightforwardly by means of the CORBA Security Service [CORBAservices Page 15-1].

The mapping of the attribute to CORBA is:

```
struct AccessControl {
   boolean execute;
   boolean getValue;
   boolean setValue;
   boolean getDefinitionResult;
   boolean getDirectoryResult;
   boolean createObject;
   boolean deleteObject;
};
```

6.4 IDL

```
interface Server {
  attribute ObjectName ServiceAccessPoint;
  attribute LogicalDeviceList LogicalDevices;
  attribute FileList Files;
  attribute ClientAssociationList ClientAssociations;
  any ServerDirectory (in ClassTypeCode classtype)
    raises (FailedDirectory,NoSuchType);
};
```

6.5 UML

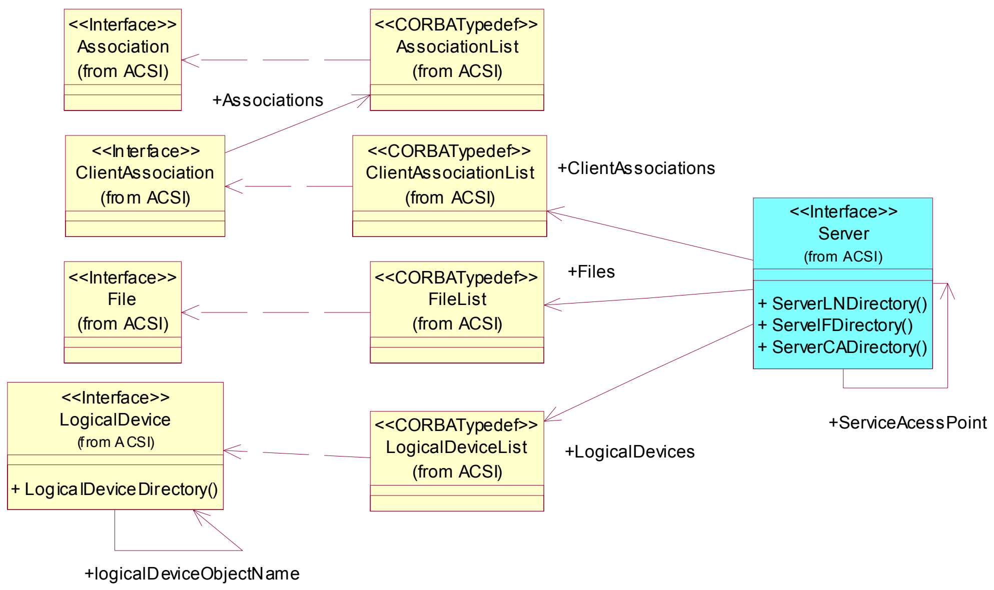


Figure 6-1: UML model for the ACSI::Server interface.

7. Mapping of Association Model

7.1 Concept of abstract associations

The associations in the ACSI are not the same in CORBA applications because the basic CORBA association model is done on the basis of individual invocations. This means that associations are created dynamically by the underliying mechanisms provided by the broker and the POA and are not specficially handled by the applitation iself.

This is obviously a sacrifice of application tight control, for the benefit of easy of use, but in some cases it is necessary to:

- Reduce the burden of on-invocation association establishment
- Control resources for real time /embedded performance

These resources are provided now by means of new APIs introduced in the RT-CORBA specification and the Messaging specification.

Using the RT-POA interfaces it is possible to determine association establishment policies and also resource management.

Using the `PortableServer::Current` interface, the status of the present invocation context can be determined, and servers can manage associations using ACSI mapped control structures. In most cases the management of associations is related with the POA policies used. This can make necessary for a server the use of several POAs to employ different policies with different clients.

QoS settings are interfaces derived from `CORBA::Policy`. `PolicyManager` is a locality constrained interface that can be accessed through the ORB interface. Aditional policies can be established at the object level and at the thread level (through `PolicyCurrent`);

7.2 IDL

```
/* Association ______________________________________ */

enum TriggeringPolicy {
     user_triggered,
     network_scheduled};
```

```
enum AssociationEstablishment {
     pre_established,
     pre_defined,
     dynamic};

enum AssociationRole {
     client,
     server,
     peer};

struct AssociationChar {
  TriggeringPolicy triggeringPolicy; // OPTIONAL
  boolean          queued;           // OPTIONAL
  boolean connection_oriented;         // OPTIONAL
  AssociationEstablishment established;         // OPTIONAL
  AssociationRole  role;             //OPTIONAL
};

enum AssociationTypes {
qUU,     // queued user triggered unidirectional
  qUB_CO,  // queued user triggered bi_directional connection oriented
  qUB_CL,  // queued user triggered bi_directional connectionless
  qUB_FC,  // queued user triggered bi_directional flow control
  qUB_Seg, // queued user triggered bi_directional segmenting
  bNU      // buffered network scheduled unidirectional
};

enum AssociationEndPointState {
     closed,
     open,
     requesting,
     replied,
     responding};

interface Association {
  attribute AssociationChar associationChar;
  attribute AssociationTypes associationType;
  attribute AssociationEndPointState aEPState; // OPTIONAL
  attribute long maxReqCalling;           // OPTIONAL
  attribute long maxReqCalled;            // OPTIONAL
  attribute long maxUCSC;                 // OPTIONAL
  attribute long maxUCSS;                 // OPTIONAL
  attribute long maxOSCC;                 // OPTIONAL
  attribute long maxOSCS;                 // OPTIONAL
};
```

7.3 UML

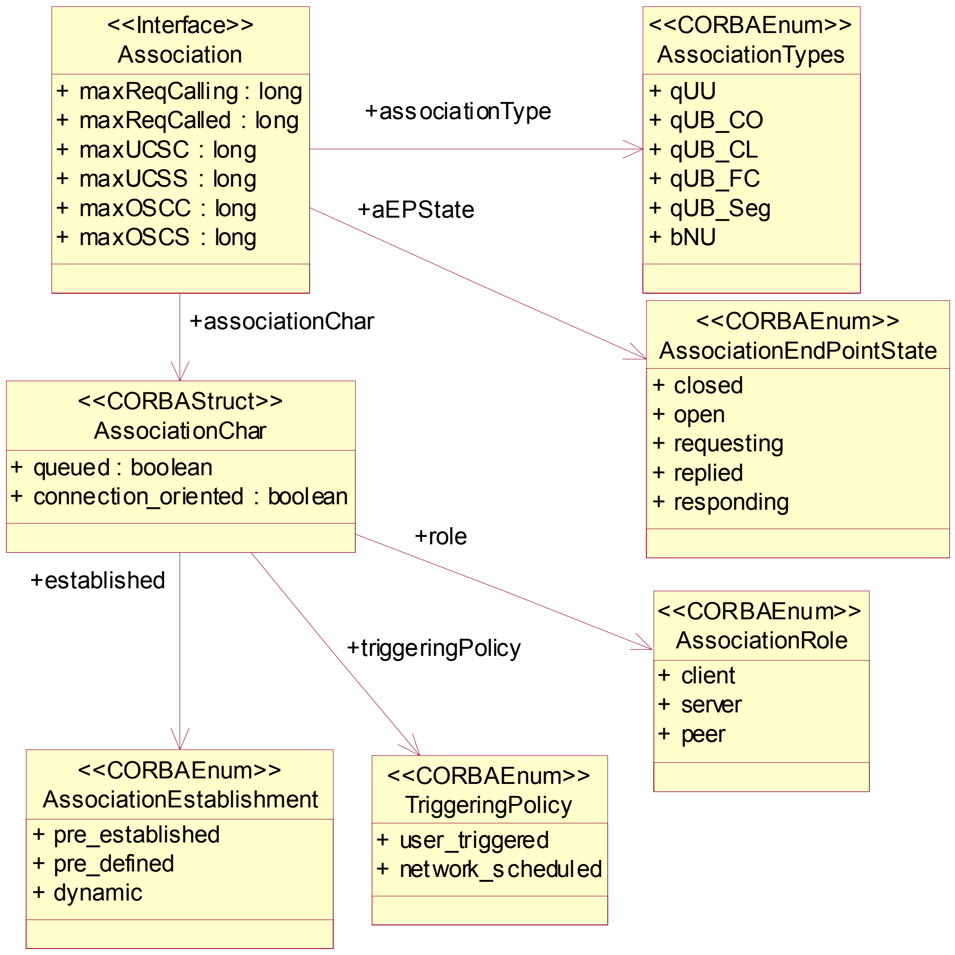


Figure 7-1:Associations.

8. Mapping of Logical Device Model

8.1 Logical device class definition

8.1.1 logicalDeviceObjectName

This attribute is intended to be used as system wide identification of the logical device. The ACSI only addresses server scope, but this mapping requires application scope.

In CORBA applications this functionality is achieved by means of the IOR of the server. This IOR is generated by the POA upon object creation (unless persistence policy is used) and can be obtained by means of the C++ IDL compiler-generated `_this()` operation.

We will use an ACSI::ObjectName type for this identification field:

```
Module ACSI{
   typedef CosNaming::name ObjectName;
. . .
}
```

8.1.2 LogicalNodes

The collection of logical nodes in the logical device.

8.1.3 AccessControl

The access control attribute.

8.2 Logical device access services

8.2.1 LogicalDeviceDirectory

The LogicalDevice`Directory` service returns a list of logical nodes in the server.

The service Response+ is a type LogicalNodeList containing the list of requested entities (see section 5.8 in this document).

The service Response- is mapped to two exceptions.

8.3 IDL

```
interface LogicalDevice {
   attribute ACSI::ObjectName logicalDeviceObjectName;
   attribute LogicalNodeList logicalNodes;
   attribute AccessControl accessControl;
   LogicalNodeList LogicalDeviceDirectory ();

};

typedef sequence <LogicalDevice> LogicalDeviceList;
```

8.4 UML Model

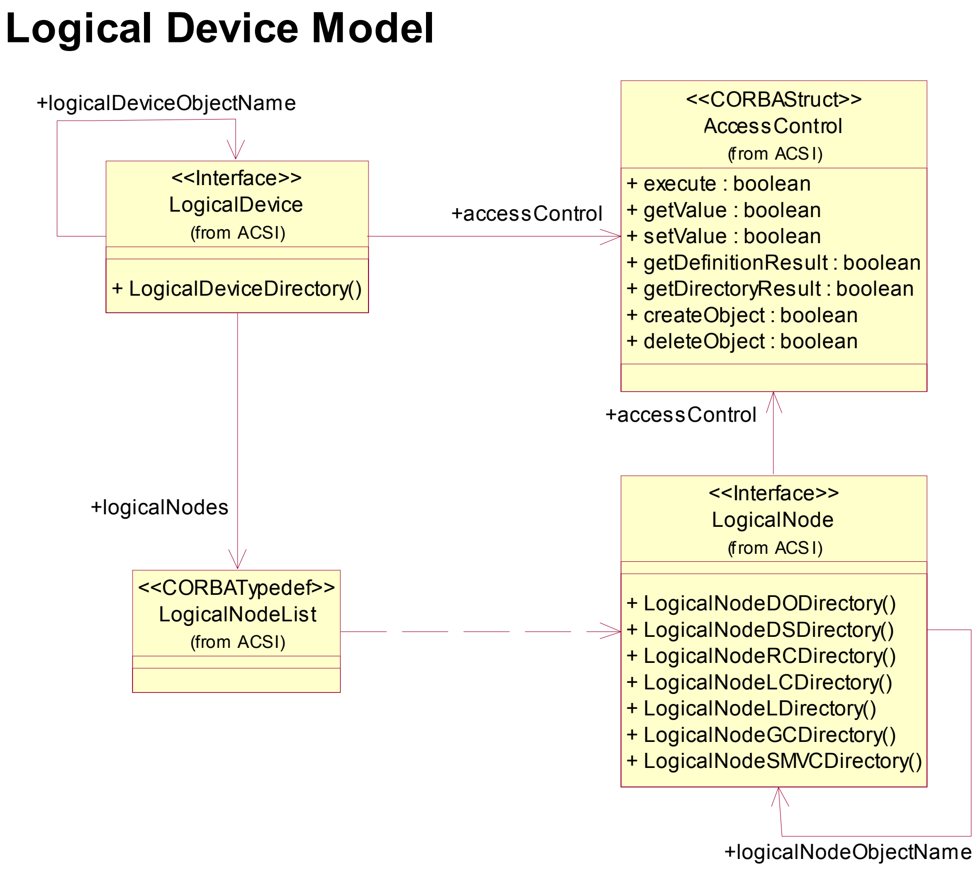


Figure 8-1: UML model of the Logical Device Interface.

9. Mapping of Logical Node Model

9.1 Logical node class definition

9.1.1 LnState

The collection of states that are possible for a LogicalNode. An enumerated type:

```
enum LogicalNodeState {On, Blocked, Test, TestBlocked, Off};
```

9.1.2 LogicalNodeObjectName

This attribute is intended to be used as system wide identification of the logical node. The ACSI only addresses server scope, but this mapping requires application scope.

In CORBA applications this functionality is achieved by means of the IOR of the server. This IOR is generated by the POA upon object creation (unless persistence policy is used) and can be obtained by means of the C++ IDL compiler-generated `_this()` operation.

We will use an `ACSI::ObjectName` type for this identification field:

```
Module ACSI{
   typedef CosNaming::name ObjectName;
. . .
}
```

9.1.3 DataObjects

A sequence of data objects contained in the Logical Node.

9.1.4 DataSets

A sequence of data sets contained in the Logical Node.

9.1.5 ReportControls

A sequence of report control objects contained in the Logical Node.

9.1.6 LogControls

A sequence of report control objects contained in the Logical Node.

9.1.7 Logs

A sequence of log objects contained in the Logical Node.

9.1.8 gOOSEControl

An optional field that will exist only for LN0.

9.1.9 SampledValueControls

An optional field that will exist only for LN0.

9.1.10 AccessControl

The access control attribute.

9.2 Logical node access services

9.2.1 LogicalNodeDirectory

This service provides a list

9.3 IDL

```
enum DataAttributeCharacteristic {op, st, co, mx, sp,
   ct, dc, cf, rc, lc, gc, sc, ax};


struct LNState {
   string dataAttributeName;
   sequence <LogicalNodeState> dataAttributeContent;
   sequence <DataAttributeCharacteristic> dataAttributeCharacteristic;
};

interface LogicalNode {
   enum LogicalNodeState {On, Blocked, Test, TestBlocked, Off};

  // Attributes
    attribute LNState lnState;
   attribute ACSI::ObjectName logicalNodeObjectName;
   attribute DataObjectList dataObjects;
   attribute DataSetList    dataSets;
   attribute ReportControlList reportControls;
   attribute LogControlList logControls;
   attribute LogList logs;
   union LN0_g switch (boolean){
      case TRUE:
      GOOSEControl gOOSEControl;
   };
   union LN0_s switch (boolean){
      case TRUE:
      SampledValueControlList sampledValueControl;
   };
```

```
    attribute AccessControl accessControl;

   // Operations
    Any LogicalNodeDirectory (in ClassTypeCode classtype)
           raises (FailedDirectory,NoSuchType);
};

typedef sequence <LogicalNode> LogicalNodeList;
```

9.4 UML

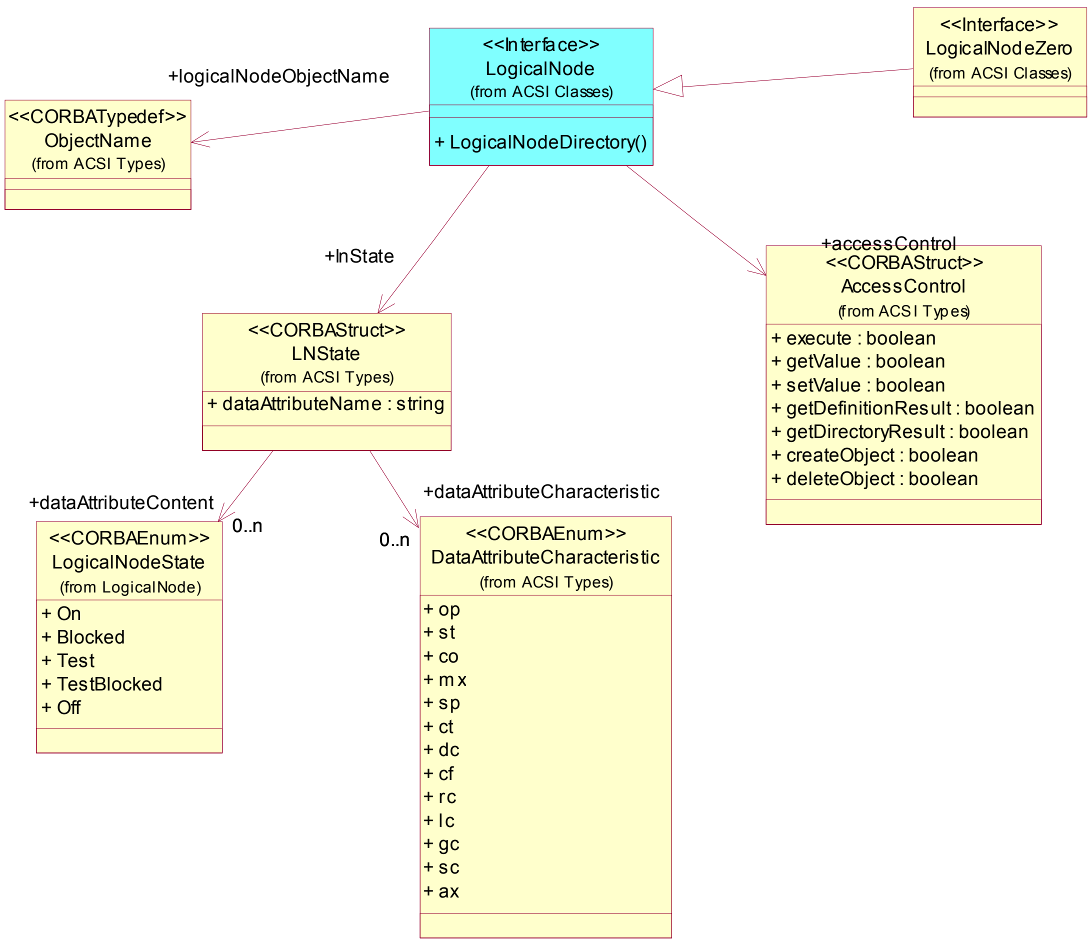


Figure 9-1: UML model of a logical device.

10. Mapping of Data Object Model

10.1 Data object class definition

10.1.1 DataObjectName

This attribute is intended to be used as identification of the data object within the logical node.

We can use an `ACSI::ObjectName` type for this identification field:

```
Module ACSI{
   typedef CosNaming::name ObjectName;
. . .
}
```

This means that the name includes also names of higher level entities.

10.1.2 DataAttributes

A list of data attributes:

```
sequence <DataAttribute> dataAttributes;
```

10.1.3 AccessControl

An access control attribute.

10.2 Data object class services

10.2.1 GetDataObjectValues

This service is intended for retrieving a collection of object values from a specific server. This is an example of the problems mentioned in section 4.3.5. This is a service addressed TO A SERVER to get values of some DATA OBJECTS. Were they true objects, the request should be addressed directly to them and not to a higher instance (unless some specific functionality is sought, which is not the case). So, WE MAP IT AS A `Server` operation and not as a `DataObject` operation.

As was said before, Data Objects are mapped to CORBA objects and this means that the server will act as an intermediary for this task of gathering data.

In relation with the mapping of services that move objects from here to there (GETs and SETs) we can quote the CORBA 2.3.1 section on Objects-by-Value:

> *Objects, more specifically, interface types that objects support, are defined by an IDL interface, allowing arbitrary implementations. There is great value, which is described in great detail elsewhere, in having a distributed object system that places almost no constraints on implementations.*
>
> *However there are many occasions in which it is desirable to be able to pass an object by value, rather than by reference. This may be particularly useful when an object's primary "purpose" is to encapsulate data, or an application explicitly wishes to make a "copy" of an object.*

ValueTypes should be adequate for this mapping but they are not yet implemented in most commenrcial ORBS.

This service can also return also the list of data object names, so the `inout` modifier:

```
ListOfDataObjectValues Server::GetDataObjectValues(
    inout ListOfDataObjectNames namelist,
    in boolean specificationWithResult)
    raises (GetFailed);
```

It looks pretty stupid but it can return one input parameter (`ListOfDataObjectNames`) withouh modification.

The `ListOfDataObjectValues` is a sequence of data object values[11] or reasons for failure in getting the value of a specific object.

10.2.2 SetDataObjectValues

Read the discussion before:

```
ListOfDataObjectConfirmations Server::SetDataObjectValues(
    in ListOfDataObjectNames namelist,
    in ListOfDataObjectValues namelist,)
    raises (SetFailed);
```

[11] As CORBA structs. Remember that object passing in CORBA is done by reference unless you employ the Objects-by-Value semantics in the IDL:

```
valuetype exampleValue { // note this is not a CORBA::Object
        string name;
        float value;
        boolean test;

};
```

10.2.3 GetDataObjectDefinition

It is not easy to map this service because it is difficult to determine what is a "DataObjectDefinition". The ACSI says that it is in the same page:

- The "list of data attribute object types" : this can be done using CORBA typecodes (but this means getting into dynamic CORBA[12]) or as text strings (but this will be almost unusable).
- The list of "data attribute definitions": What is this ?

Another possibility is returning the list of `ACSI::ObjectName` that, following ACSI norms is constructed using DataAttribute types as part of the names.

10.2.4 DataObjectDirectory

This service gets the mentioned list of `ACSI::ObjectName` of the data attributes that compose the DataObject:

```
sequence <ObjectName> ObjectNameList;

ObjectNameList Server::DataObjectDirectory(
    in DataObjectName name)
    raises (DirectoryFailed);
```

Going back to the discussion in 10.2.1 about who should be addressed with these requests, being this a `DataObject::` operation intead of a `Server::` operation, it should be:

```
sequence <ObjectName> ObjectNameList;

ObjectNameList DataObject::Directory()
    raises (DirectoryFailed);
```

It is –obviously– not necessary to pass as parameter the name of the DataObject addressed.

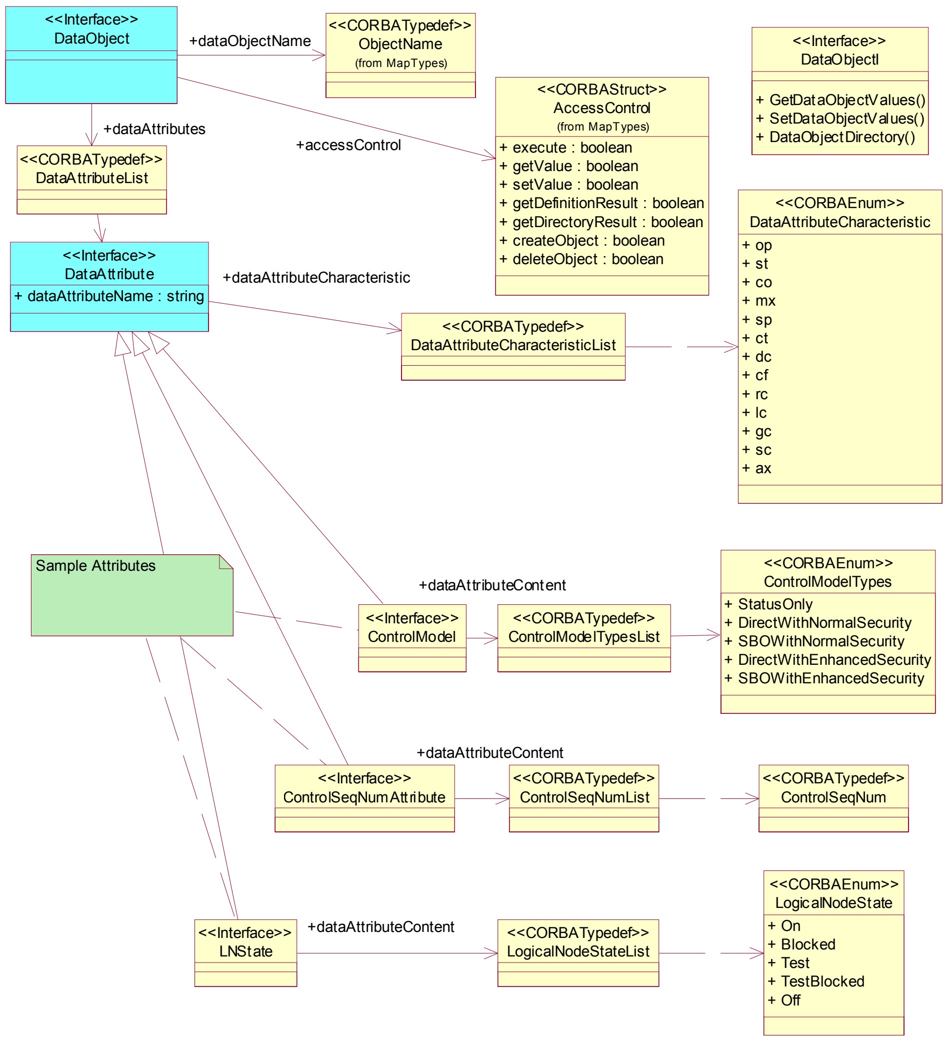
10.3 IDL

```
sequence <ObjectName> ObjectNameList;
typedef ObjectNameList ListOfDataObjectNames;

sequence <DataAttribute> DataAttributeList;

interface DataObject {
   ObjectName dataObjectName;
    DataAttributeList dataAttributes;
    AccessControl accessControl;
};

sequence <DataObject> DataObjectList;
```

[12] *I.e.* going beyond MinimumCORBA implementations.

```
ListOfDataObjectValues Server::GetDataObjectValues(
    inout ListOfDataObjectNames namelist,
    in boolean specificationWithResult)
    raises (GetFailed);

ListOfDataObjectConfirmations Server::SetDataObjectValues(
    in ListOfDataObjectNames namelist,
    in ListOfDataObjectValues namelist,)
    raises (SetFailed);

ObjectNameList Server::DataObjectDirectory(
    in DataObjectName name)
    raises (DirectoryFailed);
```

10.4 UML



<<Interface>> DataObject
+dataObjectName
<<CORBATypedef>> ObjectName (from MapTypes)
<<Interface>> DataObjectI
+ GetDataObjectValues()
+ SetDataObjectValues()
+ DataObjectDirectory()
+dataAttributes
+accessControl
<<CORBAStruct>> AccessControl (from MapTypes)
+ execute : boolean
+ getValue : boolean
+ setValue : boolean
+ getDefinitionResult : boolean
+ getDirectoryResult : boolean
+ createObject : boolean
+ deleteObject : boolean
<<CORBATypedef>> DataAttributeList
<<Interface>> DataAttribute
+ dataAttributeName : string
+dataAttributeCharacteristic
<<CORBATypedef>> DataAttributeCharacteristicList
<<CORBAEnum>> DataAttributeCharacteristic
+ op
+ st
+ co
+ mx
+ sp
+ ct
+ dc
+ cf
+ rc
+ lc
+ gc
+ sc
+ ax
Sample Attributes
+dataAttributeContent
<<Interface>> ControlModel
<<CORBATypedef>> ControlModelTypesList
<<CORBAEnum>> ControlModelTypes
+ StatusOnly
+ DirectWithNormalSecurity
+ SBOWithNormalSecurity
+ DirectWithEnhancedSecurity
+ SBOWithEnhancedSecurity
+dataAttributeContent
<<Interface>> ControlSeqNumAttribute
<<CORBATypedef>> ControlSeqNumList
<<CORBATypedef>> ControlSeqNum
<<CORBAEnum>> LogicalNodeState
+ On
+ Blocked
+ Test
+ TestBlocked
+ Off
<<Interface>> LNState
+dataAttributeContent
<<CORBATypedef>> LogicalNodeStateList

Figure 10-1: UML model for DataObjects.

11. Mapping of Data Attribute Model

11.1 Data attribute class definition

11.1.1 DataAttributeName

The name of the data attribute is mapped to a full hierarchical name:

```
ACSI:ObjectName dataAttributeName;
```

11.1.2 DataAttributeContent

It is not very clear what is the data attribute content in the ACSI. In page 44 it says:

The DataAttributeContent defines the type (range of possible values) of the data attribute while being communicated.

Does it mean that this field is a variable of that type on only a typecode ?

Our view is that this class is a pure virtual class. The content will vary from heir to heir and only the DataAttributeName and the DataAttributeCharacteristic will be common to all heirs.

11.1.3 DataAttributeCharacteristic

The ACSI specifies that the data attribute characteristic is a value set an not a single value. Its elementary values are of an enumerated type:

```
enum DataAttributeCharacteristic {
     op,
     st,
     co,
     mx,
     sp,
     ct,
     dc,
     cf,
     rc,
     lc,
     gc,
     sc,
     ax
  };

typedef sequence <DataAttributeCharacteristic>
```

```
        DataAttributeCharacteristicList;
```

`DataAttributeCharacteristicList` is the type used to declare characteristics of data attributes.

11.2 DataAttribute Services

11.2.1 SetDataAttributeValues

This is an equivalent service as SetDataObjectValues. The discussion about the addressed object is also applicable.

11.2.2 SetDataAttributeValues

This is an equivalent service as GetDataObjectValues.

11.3 Attribute and parameter types

See section 5.4, Fundamental data types mapping.

11.4 Common types

See section 5.5, Mapping of common types.

11.5 IDL

```
interface DataAttribute{
   attribute string dataAttributeName;
   attribute DataAttributeCharacteristicList
                    dataAttributeCharacteristic;
};
```

11.6 UML

See UML model in Chapter 10.

12. Mapping of Data Set Model

12.1 Data set class definition

A data set is a ordered collection of DataObjects (with an order known by the client and by the server). In relation with the services specified we are in the same situation as with the rest of objects in the ACSI. We will map this interface to an interface for a Server object and not to an interface for a DataSet Object.

```
struct DataSet {
   ObjectName dataSetObjectName;
   DataObjectList dataObjectNames;
   AccessControl accessControl;
};
```

12.1.1 DataSetObjectName

The name of the data set.

```
attribute ObjectName dataSetObjectName;
```

12.1.2 DataObjectNames

The catalog of objects contained in the set.

```
attribute DataObjectList dataObjectNames;
```

12.1.3 AccessControl

An access control attribute.

```
attribute AccessControl accessControl;
```

12.2 Data set services

12.2.1 GetDataSetValue

Get the values of the data set. This service specifies an input parameter SpecificationWithResult to be equivalent to a similar service for DataObjects. But in this case the second argument is only a single data set so it has no sense to get back this argument (in case of success it is known, in case of failure it is known where is the problem).

```
DataObjectList GetDatSetValue(
     in boolean specificationWithResult,
      inout ObjectName name)
       raises (GetFailed);
```

12.2.2 SetDataSetValue

```
ListOfConfirmations SetDatSetValue(
   inout ObjectName name,
   in DataObjectList list)
      raises (SetFailed);
```

12.2.3 CreateDataSet

```
void CreateDataSet(
   in ObjectName name,
   in DataObjectList list)
      raises (CreateFailed);
```

12.2.4 DeleteDataSet

```
void DeleteDataSet(
   inout ObjectName name)
      raises (DeleteFailed);
```

12.2.5 DataSetDirectory

```
DataObjectList DataSetDirectory(
   inout ObjectName name)
      raises (FailedDirectory);
```

12.3 IDL

12.3.1 IDL as ACSI Specifies

```
struct DataSet {
   ObjectName dataSetObjectName;
   DataObjectList dataObjectNames;
   AccessControl accessControl;
};

interface DataSetI{
   DataObjectList GetDatSetValue(
      in boolean specificationWithResult,
       inout ObjectName name)
        raises (GetFailed);
   ListOfConfirmations SetDatSetValue(
      inout ObjectName name,
      in DataObjectList list)
         raises (SetFailed);
   void CreateDataSet(
      in ObjectName name,
      in DataObjectList list)
         raises (CreateFailed);
   void DeleteDataSet(
      inout ObjectName name)
         raises (DeleteFailed);
   DataObjectList DataSetDirectory(
      inout ObjectName name)
         raises (FailedDirectory);
};

typedef sequence <DataSet> DataSetList;
```

12.3.2 Better IDL

A better IDL is achieved splitting the interface into DataSet interface and Server interface parts. The Server is requested to create and destroy and the DataSet is requested for data manipulation.:

```
interface DataSet {


   attribute ObjectName dataSetObjectName;
   attribute DataObjectList dataObjectNames;
   attribute AccessControl accessControl;

   DataObjectList GetDatSetValue()
         raises (GetFailed);
   ListOfConfirmations SetDatSetValue(
      in DataObjectList list)
         raises (SetFailed);
   DataObjectList DataSetDirectory()
         raises (FailedDirectory);
};

interface Server {
   void CreateDataSet(
      in ObjectName name,
      in DataObjectList list)
         raises (CreateFailed);
   void DeleteDataSet(
      inout ObjectName name)
         raises (DeleteFailed);
};
```

12.4 UML

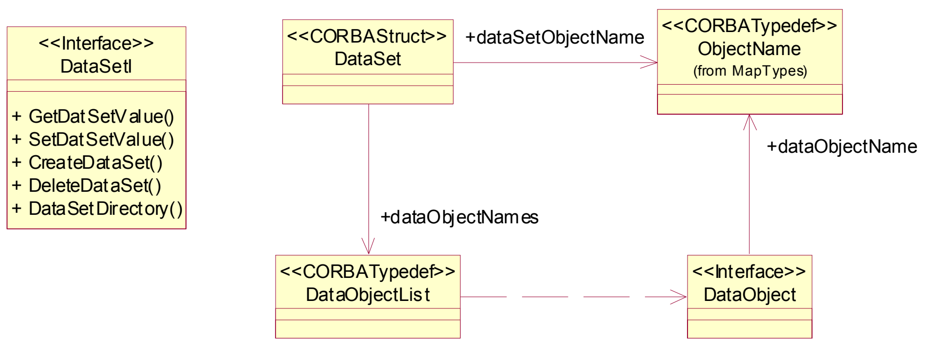


Figure 12-1: DataSet Model.

13. Mapping of Publish and Subscribe Data Transfer

13.1 Overview

This section includes the model of reporting of the ACSI. The specification indicates that the server contains a report controller that handles report generation and forwarding to a client.

We have split the specification into two interfaces, one for the report server and one for the report client. The specification, however, lacks some basic services as creating/deleting report controls and client subscribing to a report.

While this specification is intended for a one-to-one connection it can easily be extended to a one-to-many (reporting to many clients) easily (in fact this version supports this feature). This can be done without a large bandwidth cost if the ORB provides a multicast transport.

13.2 Report control class definition

The ReportControl class stores configuration parameters for report generation. This class is mapped as follows:

```
interface ReportControl {
  attribute ObjectName rcNam;
  attribute PrtEnaAttribute rptEna;
  attribute RptIdAttribute rptID;
  attribute OptFldsAttribute optFlds;
  attribute DatSetNamAttribute datSetNam;
  attribute GIAttribute gI;
  attribute BufTimAttribute bufTim;
  attribute TrgsAttribute trgs;
  attribute TrgOpsAttribute trgOps;
  attribute RBEPdAttribute rBEPd;
  attribute IntgPdAttribute intgPd;
  attribute SeqNumAttribute seqNum;
  AccessControl accessControl;
};
```

Attributes used for the report control class are:

```
  attribute ObjectName rcNam;
  attribute RptEnaAttribute rptEna;
  attribute RptIdAttribute rptID;
```

```
      attribute OptFldsAttribute optFlds;
      attribute DatSetNamAttribute datSetNam;
      attribute GIAttribute gI;
      attribute BufTimAttribute bufTim;
      attribute TrgsAttribute trgs;
      attribute TrgOpsAttribute trgOps;
      attribute RBEPdAttribute rBEPd;
      attribute IntgPdAttribute intgPd;
      attribute SeqNumAttribute seqNum;
      attribute AccessControl accessControl;
```

The optional fields attribute indicate special content in the report:

```
   typedef sequence <boolean> OptionalFields;
```

This is a sequence of boolean fields that define bahavior particularities for the report generator:

0. Reserved
1. Include sequence number
2. Report time stamp
3. Include data set name
4. Include reason for inclusion
5. Include DataObjectNames

13.3 Reporting services

We have two interfaces here: one for the control report object to be configured and operated (`ReportControlI`) and a callback interface for the report client (`ReportClientI`):

13.3.1 Report

This is the only operation supportd in the callback interface:

```
oneway void Report(in ReportFormat data); // No raises
```

Some more operation would be useful to inform the client about non-requested changes in service (for example if the server fails and is going to shutdown).

```
      ListOfReportControlAttributes GetReportControlValue(
         in ObjectName rcNam,
         in ListOfReportControlAttributeNames names)
            raises (GetFailed);

      ListOfConfirmations SetReportControlValue(
         in ObjectName rcNam,
         in ListOfReportControlAttributes names)
            raises (SetFailed);
```

13.3.2 More operations

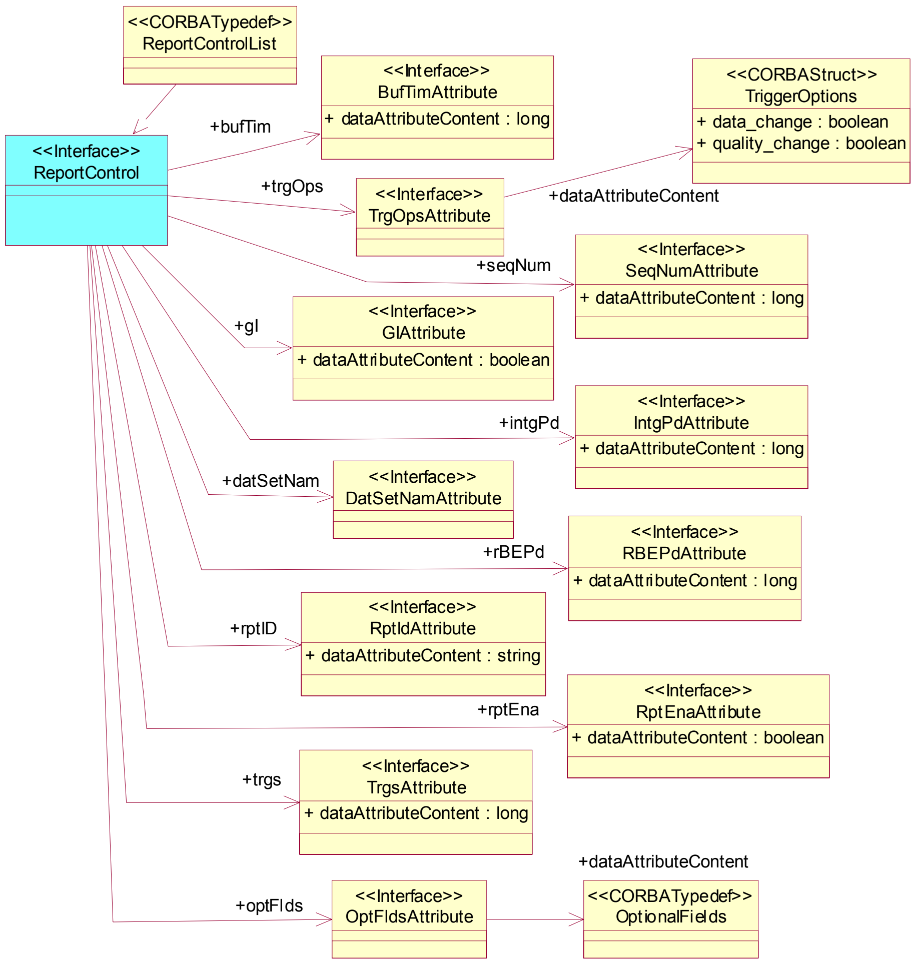
More operations would be needed for the proper handling of reports. Examples are a Subscription interface or a Factory interface for the server ???):

```
      void RegisterAsClient(in ReportClient client);
      void UnRegisterAsClient(in ReportClient client);

      // Create
      // Destroy
```

13.4 IDL

```
/*===================================================================*/
/* Report Control                                                    */
/*===================================================================*/

module ReportControl{

/* Attributes for Report Control class ------------------------*/

   interface RptEnaAttribute : DataAttribute {
      attribute boolean dataAttributeContent;
    };

   interface RptIdAttribute : DataAttribute {
      attribute string dataAttributeContent;
    };


   interface DatSetNamAttribute : DataAttribute {
      attribute ObjectName dataAttributeContent;
    };

   interface GIAttribute : DataAttribute {
      attribute boolean dataAttributeContent;
    };

   interface BufTimAttribute : DataAttribute {
      attribute long dataAttributeContent;
    };

   interface TrgsAttribute : DataAttribute {
      attribute long dataAttributeContent;
    };

   interface TrgOpsAttribute : DataAttribute {
      attribute TriggerOptions dataAttributeContent;
    };

   interface RBEPdAttribute : DataAttribute {
      attribute long dataAttributeContent;
    };

   interface IntgPdAttribute : DataAttribute {
      attribute long dataAttributeContent;
    };

   interface SeqNumAttribute : DataAttribute {
      attribute long dataAttributeContent;
    };
   interface OptFldsAttribute : DataAttribute {
      attribute OptionalFields dataAttributeContent;
     };

   typedef sequence <octet> OptionalFields;

   /* reserved                        (0),
      sequence-number          (1),
      report-time-stamp        (2),
      data-set-name                   (3),
      reason-for-inclusion            (4),
      dataObjectNames                 (5),
   */

   struct TriggerOptions {
      boolean data_change;
      boolean quality_change;
      };

   interface ReportControl {
      attribute ObjectName rcNam;
      attribute RptEnaAttribute rptEna;
      attribute RptIdAttribute rptID;
      attribute OptFldsAttribute optFlds;
      attribute DatSetNamAttribute datSetNam;
      attribute GIAttribute gI;
      attribute BufTimAttribute bufTim;
      attribute TrgsAttribute trgs;
      attribute TrgOpsAttribute trgOps;
      attribute RBEPdAttribute rBEPd;
      attribute IntgPdAttribute intgPd;
      attribute SeqNumAttribute seqNum;
      attribute AccessControl accessControl;
   };


   typedef sequence <ReportControl> ReportControlList;
   typedef sequence <DataAttribute> ListOfReportControlAttributes;

   interface ReportClientI{
      oneway void report(in ReportFormat data); // No raises
   };

   interface ReportControlI{
      ListOfReportControlAttributes GetReportControlValue(
         in ObjectName rcNam,
         in ListOfReportControlAttributeNames names)
            raises (GetFailed);
      ListOfConfirmations SetReportControlValue(
         in ObjectName rcNam,
         in ListOfReportControlAttributes names)
            raises (SetFailed);

      void RegisterAsClient(in ReportClient client);
      void UnRegisterAsClient(in ReportClient client);
      // Create
      // Destroy

   };


}; /* EO ReportControl  */
```

13.5 UML



<<CORBATypedef>> ReportControlList
<<Interface>> ReportControl
+bufTim
<<Interface>> BufTimAttribute
+ dataAttributeContent : long
<<CORBAStruct>> TriggerOptions
+ data_change : boolean
+ quality_change : boolean
+trgOps
<<Interface>> TrgOpsAttribute
+dataAttributeContent
+seqNum
<<Interface>> SeqNumAttribute
+ dataAttributeContent : long
+gI
<<Interface>> GIAttribute
+ dataAttributeContent : boolean
+intgPd
<<Interface>> IntgPdAttribute
+ dataAttributeContent : long
+datSetNam
<<Interface>> DatSetNamAttribute
+rBEPd
<<Interface>> RBEPdAttribute
+ dataAttributeContent : long
+rptID
<<Interface>> RptIdAttribute
+ dataAttributeContent : string
+rptEna
<<Interface>> RptEnaAttribute
+ dataAttributeContent : boolean
+trgs
<<Interface>> TrgsAttribute
+ dataAttributeContent : long
+dataAttributeContent
+optFlds
<<Interface>> OptFldsAttribute
<<CORBATypedef>> OptionalFields

Figure 13-1: ReportControl class UML diagram.

14. Mapping of Generic Object Oriented System-wide Event (GOOSE)

14.1 Introduction

The GOOSE is intended for the distribution of system wide events. The natural implementation for it is the CORBAservices Event Service. The main problem with the event service is its lack of real time features. Real-time even services have been described elsewhere [Harrison 1997].

The GOOSE does not specify who receives the events. System-wide seems to mean that the event reaches every systems in the SAS. We understand that this means all Servers in the SAS. It is obviously necessary to provide some way of receiver registration.

The same can be said about what events should be distributed for each receiver. The ACSI does not specify types of events nor classes of receivers. Should it be done, the natural implementation for events with filtering is the Notification Service.

From the specification it is not clear at all WHAT events are generated nor HOW they are generated. It is mentioned a power-up event and an associate GOOSE event, but this seem a little small utility for such kind of service.

14.2 Generic object oriented system-wide event control class

This is the controlling class for event generation:

```
interface GOOSEControl {
   attribute ObjectName gcNam;
   attribute GooseEna gooseEna;
   attribute ObjectName sndgLD;
   attribute UserDatNam userDatNam;
};
```

14.2.1 GcName

The name of the GOOSE control object.

```
    attribute ObjectName gcNam;
```

14.2.2 GooseEna

The Boolean controlling if the GOOSE controller wil generate events.

```
interface GooseEna : GenericDataAttribute{
    attribute boolean dataAttributeContent;
};
```

14.2.3 SndgLD

Name of sending logical device:

```
    attribute ObjectName sndgLD;
```

14.2.4 UserDatNam

An user data name to be included in GOOSE messages:

```
interface UserDatNam : GenericDataAttribute {
    attribute string dataAttributeContent;
  };
```

14.3 Generic object oriented system-wide event (GOOSE) message

The GOOSE message is the informational content of a GOOSE event.

```
struct GooseMessage{
    string sendingIED;
    TimeStamp t;
    long seqNum;
    long stNum;
    long usec;
    any userDat;
    };
```

This should be inserted into a type `CORBA::any` if the CORBAservices event channel is going to be used.

14.4 Service specification

The service specification provides two operations for getting and setting the GOOSE control state.

14.4.1 GetGOOSEControlValue

```
    GOOSEControl GetGOOSEControlValue()
        raises (GetFailed);
```

14.4.2 SetGOOSEControlValue

```
      GOOSEControl SetGOOSEControlValue(in GOOSEControl value)
         raises (SetFailed);
```

14.5 IDL

```
/*==================================================================*/
/* GOOSE                                                            */
/*==================================================================*/

module GOOSE{

   interface GooseEna : GenericDataAttribute{
      attribute boolean dataAttributeContent;
   };

   interface UserDatNam : GenericDataAttribute {
      attribute string dataAttributeContent;
    };


   interface GOOSEControl {
      attribute ObjectName gcNam;
      attribute GooseEna gooseEna;
      attribute ObjectName sndgLD;
      attribute UserDatNam userDatNam;
   };

   interface GOOSEControlI {
      GOOSEControl GetGOOSEControlValue()
         raises (GetFailed);
      GOOSEControl SetGOOSEControlValue(in GOOSEControl value)
         raises (SetFailed);
   };

   struct GooseMessage{
      string sendingIED;
      TimeStamp t;
      long seqNum;
      long stNum;
      long usec;
      any userDat;
      };

};
```

14.6 UML

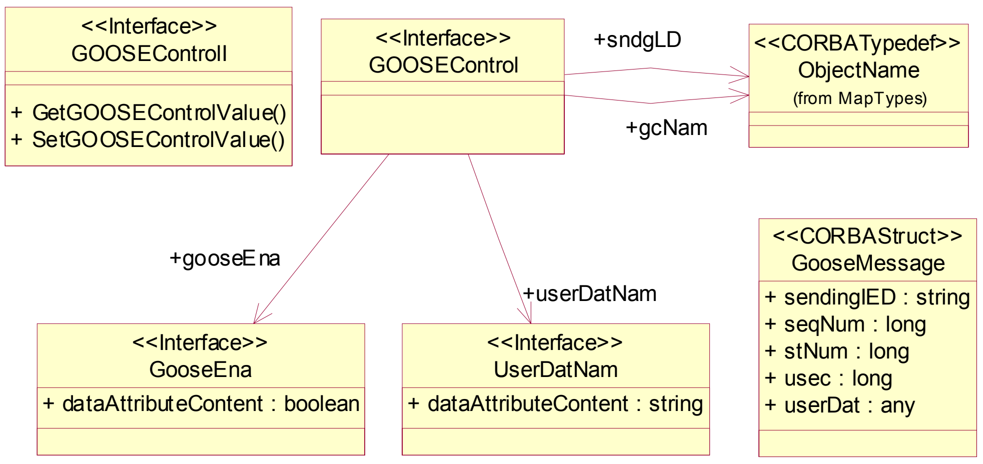


Figure 14-1: GOOSE UML model.

15. Mapping of Control Model

15.1 Control Introduction

The definition of the parameters of the Control interfaces is as follows:

```
typedef ObjectName      ControlObjectName;

typedef any Value;

typedef TimeStamp T;

typedef boolean Test;

struct Check{
    boolean synchrocheck;
    boolean interlock_check;
};
```

In case of Response-, additional cause diagnosis shall identify the reason of failure using the enumerated type `enum AddCause`[13].

15.2 Control services

15.2.1 Select

```
      void Select (in ControlObjectName name)
         raises (ControlFailed);
```

15.2.2 SelectWithValue

This service will return AddCause = CORBA::nil in case Response+.

```
      AddCause SelectWithValue (
         inout ControlObjectName name,
         inout Value value,
         inout T time,
         inout Test test,
         in Check check)
            raises (ControlFailed);
```

[13] Names of values have been changed to use underscores instead of hyphens. Also 1_to_n_control has been changed to one_to_n_control to comply with IDL naming rules.

15.2.3 Cancel

This service will return AddCause = NIL in case Response+

```
    AddCause Cancel (
       inout ControlObjectName name,
       inout T time,
       inout Test test)
```
raises (ControlFailed);

15.2.4 Operate

This service will return AddCause = NIL in case Response+

```
    AddCause Operate (
       inout ControlObjectName name,
       inout Value value,
       inout T time,
       inout Test test,
       in Check check)
          raises (ControlFailed);
```

15.2.5 CommandTermination

This service will return AddCause = NIL in case Response+

```
    AddCause CommandTermination (
       out ControlObjectName name,
       out T time,
       out Test test)
          raises (ControlFailed);
```

15.2.6 Synchrocheck

This service will return AddCause = NIL in case Response+

```
 AddCause Synchrocheck (
       inout ControlObjectName name,
       inout T time,
       inout Test test)
          raises (ControlFailed);
```

15.2.7 TimeActivatedOperate

This service will return TimOperRsp in case Response+ and AddCause in case Response-.

```
    any TimeActivatedOperate (
       inout ControlObjectName name,
       inout Value value,
       inout T time,
       inout Test test)
          raises (ControlFailed);
```

15.3 IDL

```
/*===================================================================*/
/* Control                                                          */
/*==================================================================*/

module Control{

// Service paramenters for control interfaces

typedef ObjectName      ControlObjectName;
typedef any Value;
typedef TimeStamp T;
typedef boolean Test;
struct Check{
    boolean synchrocheck;
    boolean interlock_check;
};

// Additional cause diagnosis shall identify the reason of failure

enum AddCause {
        not_supported,
        blocked_by_switching_hierarchy,
        blocked_by_event,
        blocked_by_interlocking,
        blocked_by_setpoint_command,
        target_exists,
        parameter_error,
        time_limit_over,
        address_error,
        hardware_error,
        one_of_n_control,
        system_crash,
        step_limit,
        command_already_in_execution,
        plausibility_error,
        blocked_by_synchrocheck,
        debounce_active,
        abortion,
        parameter_charge_in_execution,
        CB_alarm
        };


   interface Control {

      void Select (in ControlObjectName name)
        raises (ControlFailed);

// This service will return AddCause = NIL in case Response+


      AddCause SelectWithValue (
        inout ControlObjectName name,
        inout Value value,
        inout T time,
        inout Test test,
        in Check check)
           raises (ControlFailed);


// This service will return AddCause = NIL in case Response+

      AddCause Cancel (
        inout ControlObjectName name,
        inout T time,
        inout Test test)
          raises (ControlFailed);

// This service will return AddCause = NIL in case Response+

      AddCause Operate (
        inout ControlObjectName name,
        inout Value value,
        inout T time,
        inout Test test,
        in Check check)
          raises (ControlFailed);

// This service will return AddCause = NIL in case Response+

      AddCause CommandTermination (
        out ControlObjectName name,
        out T time,
        out Test test)
          raises (ControlFailed);

// This service will return AddCause = NIL in case Response+

    AddCause Synchrocheck (
        inout ControlObjectName name,
        inout T time,
        inout Test test)
          raises (ControlFailed);

// This service will return TimOperRsp in case Response+
// and AddCause in case Response-

      any TimeActivatedOperate (
        inout ControlObjectName name,
        inout Value value,
        inout T time,
        inout Test test)
          raises (ControlFailed);

    };


    typedef sequence <Control> ControlList;

};
```

15.4 UML

Not very useful but here it is.

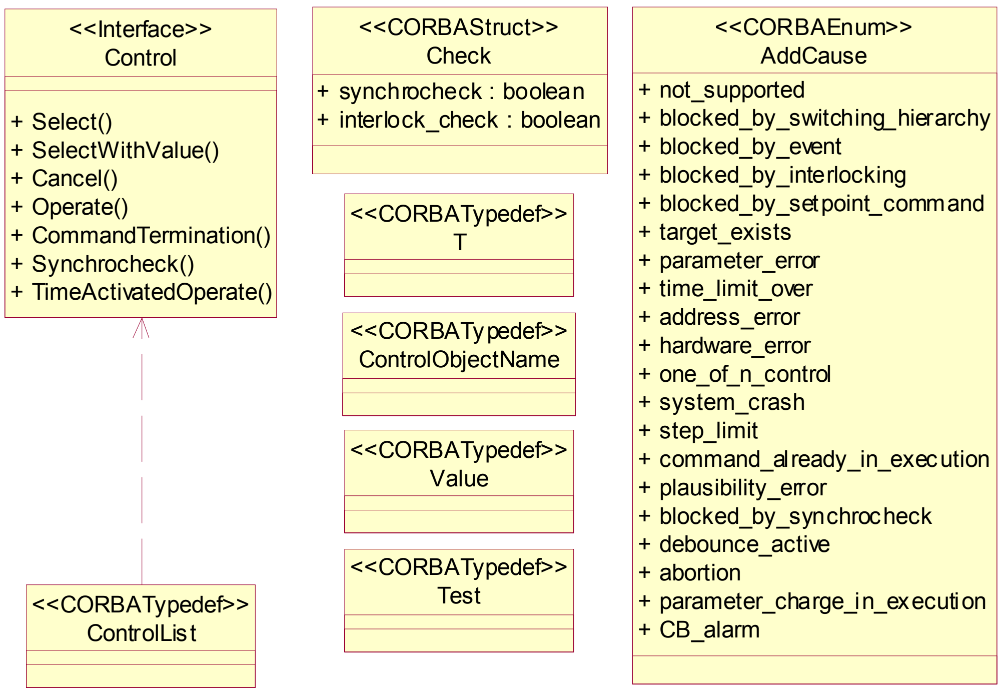


Figure 15-1: Control interface model in UML.

16. Mapping of Substitution Model

16.1 Substitution overview

Service parameters are very similar to parameters used in the Control Interface.

16.2 Service specification

16.2.1 Substitute

This service will return AddCause = NIL in case Response+

```
      AddCause Substitute (
         inout DataObjectName name,
         inout Value value,
         inout T time,
         inout Test test)
            raises (SubstituteFailed);
```

16.2.2 UnSubstitute

This service will return AddCause = NIL in case Response+

```
   AddCause UnSubstitute (
         inout DataObjectName name,
         inout T time,
         inout Test test)
            raises (SubstituteFailed);
```

};

16.3 IDL

```
/*================================================================*/
/* Substitution                                                   */
/*================================================================*/

module Substitution{

   interface Substitution {


// This service will return AddCause = NIL in case Response+

      AddCause Substitute (
        inout DataObjectName name,
        inout Value value,
        inout T time,
        inout Test test)
          raises (SubstituteFailed);

// This service will return AddCause = NIL in case Response+

   AddCause Substitute (
        inout DataObjectName name,
        inout T time,
        inout Test test)
          raises (SubstituteFailed);

   };


};
```

17. Mapping of Transmission of Sampled Measured Values

17.1 Overview

The model applies to the exchange of values of a data set (collection of data objects) in a tilaly manner. There exist a data sampler (that can make local buffering) an a remote buffer where data are pulled by the client.

It is not clear at all how connections between them are going to happen. This can be nicely implemented using a real-time even service. Multicasting of samplings is also possible.

17.2 Sampled Measured Value Control Class Definition

This is that class for the attributes controlling how sampleas are taken:

```
interface SampledValueControl {
   attribute ObjectNameAttribute svcNam;
   attribute SvEnaAttribute svEna;
   attribute DatSetNamAttribute datSetNam;
   attribute RefrRateAttribute refrRate;
   attribute NoOfSmpAttribute noOfSmp;
   attribute ComUpRateAttribute comUpRate;
   attribute SmpRateAttribute SmpRate;
   };
```

17.2.1 SvcNam

The name of the.

```
   attribute ObjectNameAttribute svcNam;
```

17.2.2 SvEna

Sampled Value Controller enabling:

```
interface SvEnaAttribute : GenericDataAttribute {
     attribute boolean dataAttributeContent;
    };
```

17.2.3 DatSetNam

Data set sampled:

```
  interface DatSetNamAttribute : GenericDataAttribute {
    attribute ObjectReference dataAttributeContent;
   };
```

17.2.4 RefrRate

Refresh rate:

```
  interface RefrRateAttribute : GenericDataAttribute {
    attribute long dataAttributeContent;
   };
```

17.2.5 NoOfSmp

Number of samples:

```
interface NoOfSmpAttribute : GenericDataAttribute {
    attribute long dataAttributeContent;
   };
```

17.2.6 ComUpRate

Communication rate:

```
interface ComUpRateAttribute : GenericDataAttribute {
    attribute long dataAttributeContent;
   };
```

17.2.7 SmpRate

```
  interface SmpRateAttribute : GenericDataAttribute {
    attribute long dataAttributeContent;
   };
```

17.3 Sampled Measured Value Service Definitions

The sampler is controlled using the SampledValueControlI interface.

17.3.1 GetSMVControlValues

```
    DataAttributeList GetSMVControlValue(
             in ObjectName name)
                    raises (GetFailed);
```

17.3.2 SetSMVControlValues

```
    void SetSMVControlValue(
             in ObjectName name,
             in DataAttributeList data)
                    raises (SetFailed);
  };
```

17.4 Sampled Measured Value Buffer Format

17.4.1 Buffer definition

The received buffer has a sequence of structs that can be pulled by clients:

```
   struct SampleData{
      any mVal;
      any q;
      };

   typedef sequence <SampleData> SampleDataList;

   struct Sample{
      SampleDataList listOfDataObjectValues ;
      long smpCnt; // Unnecessary (can be queried to the sequence)
      };

   typedef sequence <Sample> SampleList;


   struct SampledMeasuredValueBufferFormat{
      ObjectReference datSetNam;
      long noOfSmp;
      SampleList listOfSmp;
      TimeStam prefrTim;
      long confRev;
      };
```

17.4.2 Buffer interface

```
interface SampledMeasuredValueBufferI{
      Sample Pull() raises (PullFailed);
      boolean BufReceived();
      };
```

17.5 IDL

```
/*======================================================================*/
/* SampledValue                                                         */
/*======================================================================*/

module SampledValueControl{

   interface SvEnaAttribute : GenericDataAttribute {
      attribute boolean dataAttributeContent;
     };

   interface DatSetNamAttribute : GenericDataAttribute {
      attribute ObjectReference dataAttributeContent;
     };

   interface RefrRateAttribute : GenericDataAttribute {
      attribute long dataAttributeContent;
     };

   interface NoOfSmpAttribute : GenericDataAttribute {
      attribute long dataAttributeContent;
     };
   interface ComUpRateAttribute : GenericDataAttribute {
      attribute long dataAttributeContent;
     };

   interface SmpRateAttribute : GenericDataAttribute {
      attribute long dataAttributeContent;
     };

   interface SampledValueControl {
      attribute ObjectNameAttribute svcNam;
      attribute SvEnaAttribute svEna;
      attribute DatSetNamAttribute datSetNam;
      attribute RefrRateAttribute refrRate;
      attribute NoOfSmpAttribute noOfSmp;
      attribute ComUpRateAttribute comUpRate;
      attribute SmpRateAttribute SmpRate;
      };


   interface SampledValueControlI {
      DataAttributeList GetSMVControlValue(
               in ObjectName name)
                      raises (GetFailed);
      void SetSMVControlValue(
               in ObjectName name,
               in DataAttributeList data)
                      raises (SetFailed);
      };


   typedef sequence<SampledValueControl>  SampledValueControlList;

   struct SampleData{
      any mVal;
      any q;
      };

   typedef sequence <SampleData> SampleDataList;

   struct Sample{
      SampleDataList listOfDataObjectValues ;
      long smpCnt; // Unnecessary (can be queried to the sequence)
      };

   typedef sequence <Sample> SampleList;


   struct SampledMeasuredValueBufferFormat{
      ObjectReference datSetNam;
      long noOfSmp;
      SampleList listOfSmp;
      TimeStam prefrTim;
      long confRev;
      };

   interface SampledMeasuredValueBufferI{
      Sample Pull() raises (PullFailed);
      boolean BufReceived();
      };


};
```

17.6 UML

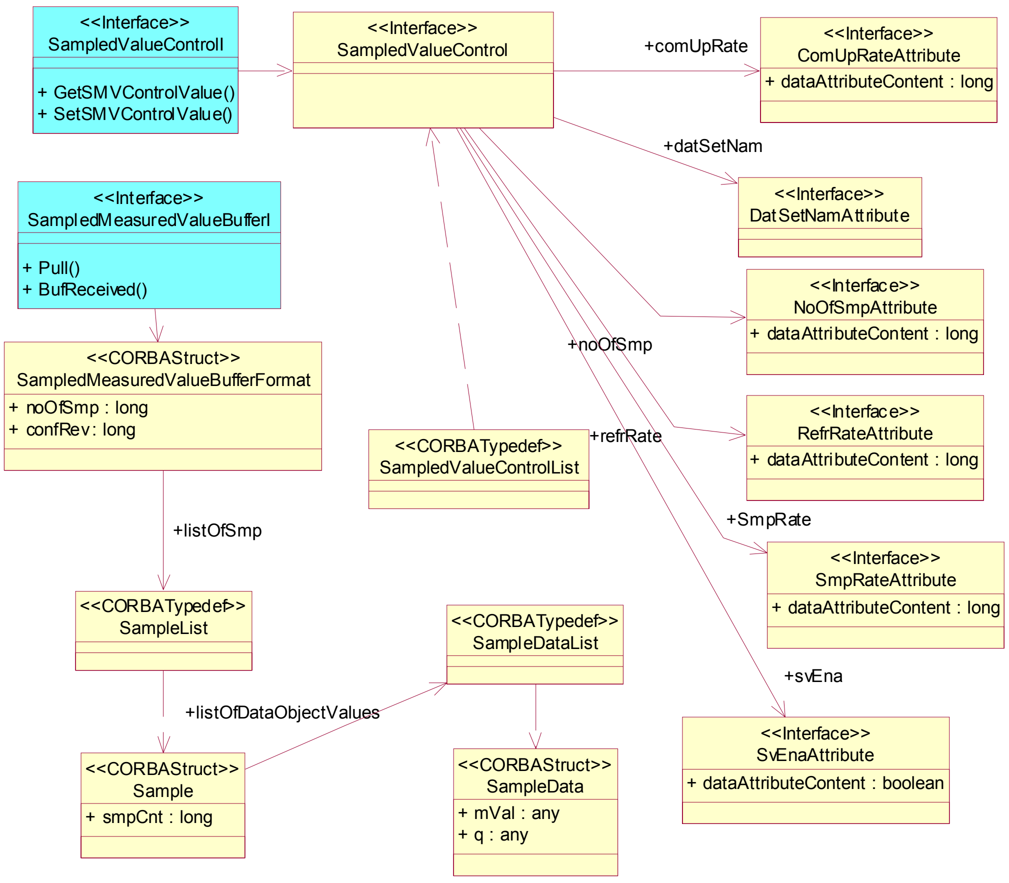


Figure 17-1: UML model of the sampled measure values service.

18. Mapping of Time Synchronisation Model

18.1 Introduction

The ACSI does not say very much about these services, mostly only their names. From the explanation it looks like these services are provided by a server to be invoked by a synchronization master, but no parameters and/or results are specified.

18.2 Services

18.2.1 Prepare

For what ?

```
void Synchro::Prepare();
```

18.2.2 Measure

Measure what ??

```
void Synchro::Measure();
```

18.2.3 Synchronise

Using what time ??

```
void Synchro::Synchronise();
```

18.3 IDL

```
Interface Synchro{
    void Synchro::Prepare();
    void Synchro::Measure();
    void Synchro::Synchronise();
};
```

19. File transfer

19.1 File transfer model

19.1.1 FileName

Name in the ACSI file store. What is the “ACSI File Store”?

```
typedef string FileName;
FileName name;
```

19.1.2 FileSize

Size in octets.

```
long fileSize;
```

19.1.3 LastModified

```
TimeStamp lastModified;
```

19.2 File services

19.2.1 GetFile service

```
FileData GetFile(in FileName name) raises (GetFailed);
```

Files can be large, an returning a file can produce proble with the stack if the file is big. This interface does not take this into account but it is possible to use an iterator to do this work avoiding the stack overflow problem.

19.2.2 SetFile service

Returns TRUE is success.

```
typedef sequence <octet> FileData;
boolean SetFile(in FileName name, in FileData data) raises (SetFailed);
```

19.2.3 DeleteFile service

Returns TRUE is success.

```
boolean DeleteFile(in FileName name) raises (DeleteFailed);
```

19.2.4 FileDirectory service

This service can be invoked passing the name of a file to get attributes or no name to get a full directory.

```
Struct FileDirectoryEntry{
    FileName name;
    long fileSize;
    TimeStamp lastModified;
}

typedef sequence <FileDirectoryEntry> FileDirectoryList;
typedef sequence <FileName> FileNameList;

FileDirectoryList FileDirectory(in FileNameList list)
    raises(DirectoryFailed);
```

19.3 IDL

The file interface is intended to be used by an ACSI server (as the rest of the ACSI, indeed).

```
typedef string FileName;
typedef sequence <octet> FileData;

struct FileDirectoryEntry{
   FileName name;
   long fileSize;
   TimeStamp lastModified;
};

typedef sequence <FileDirectoryEntry> FileDirectoryList;
typedef sequence <FileName> FileNameList;

interface File{
  FileData GetFile(in FileName name) raises (GetFailed);
  boolean SetFile(in FileName name, in FileData data) raises (SetFailed);
  boolean DeleteFile(in FileName name) raises (DeleteFailed);
  FileDirectoryList FileDirectory(in FileNameList list)
         raises(DirectoryFailed);
};
```

19.4 UML

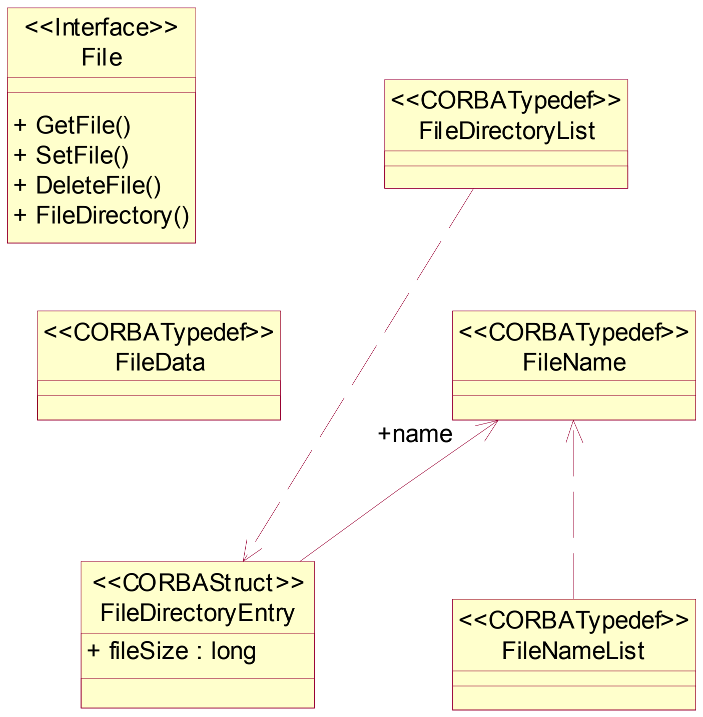


Figure 19-1: The file management interface for an ACSI server.

Part 3
Reference Material

20. Additional Bibliography

[Sanz 2000] Ricardo Sanz. *CORBA for Complex Control Systems*. Plenary Speech at IFAC Symposium on Algorithms and Architectures for Real Time Control. Palma de Mallorca, Spain, May 2000.

[Harrison 1997] T. H. Harrison, D. L. Levine, and D. C. Schmidt, "The Design and Performance of a Real-time CORBA Event Service," in *Proceedings of OOPSLA '97*, (Atlanta, GA), ACM, October 1997.

21. IDL Files

21.1 CORBA

21.1.1 COS Name Service

```
//*    Source file: CosNaming.idl   */

//
============================================================================
// DOTS Consortium (c) 2000
//
============================================================================

#ifndef _COS_NAMING_DEFINED
#define _COS_NAMING_DEFINED

#pragma prefix "omg.org"

module CosNaming
{

  typedef string Istring;  // The "I" is for future support for
Internationalization.

  struct NameComponent
  {
    Istring id;     // The name
    Istring kind;   // More info
  };

  typedef sequence <NameComponent> Name;

  enum BindingType
  {
    nobject,    // Object binding.
    ncontext    // Context binding.
  };

  struct Binding
  {
    Name binding_name;           // Object name
    BindingType  binding_type;    // Context or object
  };

  typedef sequence <Binding> BindingList;

  interface BindingIterator;

  interface NamingContext
    {
      enum NotFoundReason
```

```
       {
        missing_node,
        not_context,
        not_object
       };

       exception NotFound
     {
       NotFoundReason why;
     Name rest_of_name;
   };

       exception CannotProceed
   {
     NamingContext cxt;
     Name rest_of_name;
   };

       exception InvalidName
         {
        octet something_put_to_avoid_errors_from_Rose_Reverse_engineering;
         };

       exception AlreadyBound
         {
        octet something_put_to_avoid_errors_from_Rose_Reverse_engineering;
         };

       exception NotEmpty
         {
        octet something_put_to_avoid_errors_from_Rose_Reverse_engineering;
         };

// Operations ---------------------------------------

// Binding

       void bind (in Name n, in Object obj)
   raises(NotFound, CannotProceed, InvalidName, AlreadyBound);

       void rebind (in Name n, in Object obj)
   raises(NotFound, CannotProceed, InvalidName);

       void bind_context (in Name n, in NamingContext nc)
   raises(NotFound, CannotProceed, InvalidName, AlreadyBound);

       void rebind_context (in Name n, in NamingContext nc)
   raises(NotFound, CannotProceed, InvalidName);

// Resolution

       Object resolve (in Name n)
   raises(NotFound, CannotProceed, InvalidName);

// Unbinding

       void unbind (in Name n)
   raises(NotFound, CannotProceed, InvalidName);

// Context creation and destruction

       NamingContext new_context ();

       NamingContext bind_new_context (in Name n)
   raises(NotFound, AlreadyBound, CannotProceed, InvalidName);

      void destroy ()
   raises (NotEmpty);
```

```
      void list (in unsigned long how_many,
       out BindingList bl,
                  out BindingIterator bi);
    };

  interface BindingIterator
    {
      boolean next_one (out Binding b);

      boolean next_n (in unsigned long how_many,
           out BindingList bl);

      void destroy ();
    };
};

#endif /* _COS_NAMING_DEFINED */
```

21.1.2 COS Event Service

21.1.2.1 COS Event Channel

```
//*    Source file: CosEvenComm.idl   */

/****************************************************************************
**
                     Copyright (c) 2000 DOTS Consortium
*****************************************************************************
**/

#ifndef _COS_EVENTCOMM_DEFINED
#define _COS_EVENTCOMM_DEFINED

#pragma prefix "omg.org"

module CosEventComm
{
  exception Disconnected
    {
     octet to_put_something_and_avoid_errors_from_Rose_Reverse_engineering;
    };

  interface PushConsumer
    {
      void push (in any data) raises (Disconnected);
      void disconnect_push_consumer ();
    };

  interface PushSupplier
    {
      void disconnect_push_supplier ();
    };

  interface PullSupplier
    {
      any pull () raises (Disconnected);
      any try_pull (out boolean has_event) raises (Disconnected);
      void disconnect_pull_supplier ();
    };

  interface PullConsumer
    {
      void disconnect_pull_consumer ();
    };
};

#endif /* _COS_EVENTCOMM_DEFINED */
```

21.1.2.2 COS Event Channel Administration

```
/* Source file: CosEvenChannelAdmin.idl */

/*****************************************************************************
*
                    Copyright (c) 2000 DOTS Consortium
******************************************************************************
*/


#ifndef _COS_EVENTCHANNELADMIN_DEFINED
#define  _COS_EVENTCHANNELADMIN_DEFINED

#include "CosEventComm.idl"

#pragma prefix "omg.org"

module CosEventChannelAdmin
{

  exception AlreadyConnected {};
  exception TypeError {};

  interface ProxyPushConsumer: CosEventComm::PushConsumer
    {
      void connect_push_supplier (in CosEventComm::PushSupplier push_supplier)
        raises (AlreadyConnected);
    };

  interface ProxyPullSupplier : CosEventComm::PullSupplier
    {
      void connect_pull_consumer (in CosEventComm::PullConsumer pull_consumer)
        raises (AlreadyConnected);
    };

  interface ProxyPullConsumer : CosEventComm::PullConsumer
    {
      void connect_pull_supplier (in CosEventComm::PullSupplier pull_supplier)
        raises (AlreadyConnected, TypeError);
    };

  interface ProxyPushSupplier : CosEventComm::PushSupplier
    {
      void connect_push_consumer (in CosEventComm::PushConsumer push_consumer)
        raises (AlreadyConnected, TypeError);
    };

  interface ConsumerAdmin
    {
      ProxyPushSupplier obtain_push_supplier ();
      ProxyPullSupplier obtain_pull_supplier ();
    };

  interface SupplierAdmin
    {
      ProxyPushConsumer obtain_push_consumer ();
      ProxyPullConsumer obtain_pull_consumer ();
    };

  interface EventChannel
```

```
    {
      ConsumerAdmin for_consumers ();
      SupplierAdmin for_suppliers ();

      void destroy ();
    };
};

#endif /* _COS_EVENTCHANNELADMIN_DEFINED */
```

21.2 ACSI Mapping

The ACSI-CORBA mapping is specified in a single file ACSI.IDL:

```
/*************************************************************************
        CORBA Specific Communcation Service Mapping for IEC 61850
                     Copyright (c) 2001 DOTS Consortium
************************************************************************
Revision: 1.9.6
Date:     July 16, 2001
Author:   Ricardo Sanz, Mariano Alonso
************************************************************************

This file contains an IDL definition of the IEC 61850 Abstract
Communication Service Interface. This interface is documented in part
7-2 of the aforementionaed IEC specification.

Throughout this document, references to specific parts of IEC 61850
are done in the form $$61850-a-b/x.y.z$$ where "a" and "b" refer to parts
of the specification, and "x", "y" and "z" refer to specific clauses,
sections and subsections of the document.

The mapping is structured in several sections :

"Types"      Contains the mapping of basic types used in the
              IEC 61850 specification.

"Exceptions" Contains the collection of exceptions thrown by the ACSI
                 services.


"Services"   Contains the specific parts of 7-2.

This ACSI module contains the mappings of the ACSI services. Each set
of services is mapped as an specific CORBA module. Specific interfaces
are defined to manage specific collections of services in each module.
The services provided by the ACSI are the following:

   - Server
   - Association
   - Logical Device
   - Logical Node
   - Data
   - Data Set
   - Substitution
   - Setting group control
   - Reporting and logging
   - GSE
   - Transmission of Sampled Values
   - Control
   - File Transfer

   - Time: It is not a real service so it does not have a separate section

************************************************************************* */
```

```
#ifndef  _ACSI_DEFINED
#define  _ACSI_DEFINED
#pragma  prefix "dots"

#include "CosNaming.idl"

   /* ========================================================================
   ACSI Types
   ===================================================================== */

   module acsitypes {

      /* Foundation types =============================================
      $$61850-7-2/5.3.2$$ */

      typedef long INTEGER;

      typedef unsigned long ULong;

      typedef float REAL;

      typedef sequence <octet> BIT_STRING;

      typedef string OCTET_STRING;

      typedef string VisibleString;

      typedef sequence <octet> RawData;


      /* Common types ==================================================
      // $$61850-7-2/5.3.3$$

      /*ObjectName ---------------------------------------------------- */

      typedef VisibleString ObjectName;

      typedef sequence <acsitypes::ObjectName> ObjectNameList;

      typedef acsitypes::ObjectNameList ListOfDataObjectNames;

      /* ObjectReference ---------------------------------------------- */

      // Do not confuse these references with CORBA object references !!!!

      typedef CosNaming::Name ObjectReference;

      typedef sequence <ObjectReference> ObjectReferenceList;

      typedef ObjectReferenceList LogicalDeviceRefList;

      typedef ObjectReferenceList LogicalNodeRefList;

      typedef ObjectReferenceList DataRefList;

      typedef ObjectReferenceList DataSetRefList;

      typedef ObjectReferenceList ReportControlRefList;

      typedef ObjectReferenceList LogRefList;

      typedef ObjectReferenceList LogControlRefList;

      typedef ObjectReferenceList SettingGroupControlRefList;

      typedef ObjectReferenceList GOOSEControlRefList;
      typedef ObjectReferenceList MulticastSMVCRefList;

      typedef ObjectReferenceList UnicastSMVCRefList;

      typedef ObjectReferenceList DataAttributeRefList;

      typedef ObjectReferenceList URCRefList;

      typedef ObjectReferenceList BRCRefList;


      /* Time ------------------------------------------------------- */
      // $$61850-7-2/18$$

      enum TimeAccuracyType {

         T0,
         T1,
         T2,
         T3,
         T4,
         T5
      };

      struct TimeQualityType {
         boolean faultClockFailure;
         boolean clockNotSynchronized;
         TimeAccuracyType timeAccuracy;
      };

      struct Time {
         ULong secondOfCentury;
         ULong fractionOfSecond;
         TimeQualityType timeQuality;
         boolean leapSecondPending;
         boolean leapSecondOccured;
         boolean leapSecondNegative;
         boolean daylightSavingsPending;
         boolean daylightSavingsTime;
         ULong daylightSavingsDateSpring;
         ULong daylightSavingsDateFall;
         ULong localTimeOffset;
      };

      /* Time Stamp ---------------------------------------------------- */

      struct TimeStamp {
         TimeQualityType timQ;
         boolean leapSecondNegative;
         boolean leapSecondOccured;
      };

      /* Selectors ---------------------------------------------------- */

      /* Selectors serve the purpose of specifying filters for collections
      of objetcs. We use ACSI ObjectReferences for this purpose,
      indicating in the <id> field of each CosNaming::NameComponent
      the identifier and in the field <kind> the type of the scope.

      The main difference between selectors and ObjectReferences is
      in their use because selectors can contain information that
      include scopes our of the hierarchy of LogicalDevice/LogicalNode.xxx
      (for example they can include functional constraints
      $$61850-7-2.11.3.6.3.1$$) */

      typedef ObjectReference Selector;

      typedef sequence <Selector> SelectorList;
```

```
    /* Test status (GSE, Control) ---------------------------------- */

    enum TestStatus {

      TEST,
      NO_TEST
    };

    /* DataAttributes ============================================== */

    enum DataAttributeCharacteristic {
      OP, ST, CO, MX, SP, CT, DC, CF, RC, LC, GC, SC, AX
    };

    typedef sequence <DataAttributeCharacteristic>
       DataAttributeCharacteristicList;

    typedef RawData DataAttributeValue;

    interface DataAttribute {
      attribute ObjectName dataAttributeName;
      attribute DataAttributeCharacteristicList dataAttributeCharacteristic;
      attribute DataAttributeValue value;
    };

    typedef sequence <DataAttribute> DataAttributeList;

    typedef sequence <DataAttributeValue> DataAttributeValueList;

    typedef sequence <DataAttributeValue> ListOfDataValues;

    typedef string ListOfConfirmations;

  };

  /*=================================================================
  ACSI Exceptions
  ================================================================== */

  module acsiexceptions {

    exception FailedDirectory {
      long reason;
    };

    exception NoSuchData {
      acsitypes::ObjectName name;
    };

    exception GetFailed {
      long reason;
    };

    exception SetFailed {
      long reason;
    };

    exception CreateFailed {
      long reason;
    };

    exception DeleteFailed {
      long reason;
    };

    exception ControlFailed {
      long reason;
```

```
     };

     exception PullFailed {
        long reason;
     };

     exception AssociateFailed {
        long reason;
     };

     exception NoSuchAssociation {
        long reason;
     };

     exception NoSuchClient {
        long reason;
     };

     exception NoData {
        long reason;
     };

     exception NoAvailableUSMVC {
        long reason;
     };

     exception ActivateFailed {
        long reason;
     };

     exception SubstituteFailed {
        long reason;
     };

  };



  /* ===========================================================================
  ReportLog   $$ 61850-7-2/14 $$
  ========================================================================= */

  module acsireportlog {

     struct TriggerOptions {
        boolean data_change;
        boolean filtered_data_change;
        boolean quality_change;
        boolean cyclic_integrity;
        boolean general_interrogation;
        boolean data_set_directory;
     };

     struct ReportOptionalFields {
        boolean reserved;
        boolean sequence_number;
        boolean report_time_stamp;
        boolean data_set_name;
        boolean reason_for_inclusion;
        boolean dataReference;
     };

     /* Buffered Report Control ========================================= */

     enum BRCStateType {DISABLED,HOLD,ENABLED};

     /* Report Client
==========================================================
     This callback interface is to be supported by a report client */

     enum ReasonForInclusion {
        DATA_CHANGE_REASON,
        QUALITY_CHANGE_REASON,
        PERIODIC_REASON,
        INTEGRITY_REASON,
        GENERAL_INTERROGATION_REASON,
        NOT_VALID_REASON
     };

     struct ReportDataItem {
        acsitypes::ObjectReference dataName;
        string value;
        /* FALSE in optFlds */
        ReasonForInclusion reasonCode;
     };

     struct ReportFormat {
        acsitypes::VisibleString rptId;
        ReportOptionalFields optFlds;
        /* Only Valid if Sequence_Number == TRUE in optFlds */
        acsitypes::ULong seqNum;
        acsitypes::ULong subSeqNum;
        boolean bufOvfl;
        boolean reqBufAck;
        /* Only Valid if Report_Time_Stamp == TRUE in optFlds */
        acsitypes::TimeStamp rptTim;
        /* Only Valid if  DataSet_Name  == TRUE in optFlds */
        acsitypes::ObjectReference datSetRef;
        sequence <ReportDataItem> listOfData;
     };

     interface ReportClient {
        void Report (
           in ReportFormat reportFmt
           )
           raises (acsiexceptions::NoSuchClient);
     };

     /* Log Control class ---------------------------------------------- */

     interface LogControl {
        /* Attributes ...................................... */

        attribute acsitypes::ObjectName lCName;
        attribute boolean logEna;
        attribute acsitypes::ObjectReference dataSetRef;
        attribute TriggerOptions trgOps;
        attribute acsitypes::ULong intgPd;
        attribute acsitypes::ObjectReference logRef;
        /* Services ........................................
        GetLogControlValue and SetLogControlValue are implemented by
        the IDL compiler for each attribute. */

     };

     struct EntryData {
        acsitypes::ObjectReference datRef;
        acsitypes::DataAttributeValue datValue;
        ReasonForInclusion inclusionReasonCode;
     };

     struct LogEntry {
        acsitypes::TimeStamp timeOfLog;
        acsitypes::ULong entryId;
        EntryData entryDat;
```

```
};

typedef sequence <LogEntry> LogEntryList;

typedef unsigned long long EntryNo;

interface Log {
  /* Attributes .................................. */

  attribute acsitypes::ObjectReference logRef;
  attribute acsitypes::TimeStamp oldEntrTim;
  attribute acsitypes::TimeStamp newEntrTim;
  attribute EntryNo oldEnt;
  attribute EntryNo newEnt;
  /* This attribute should be internal to the implementation of
  the Log and not manifest in the interface !!! */

  attribute LogEntryList logEntries;

  /* Services .................................... */

  // GetLogStatusValue() is implemented by the IDL Compiler

  LogEntryList QueryLogByTime (
    in acsitypes::TimeStamp rangeStartTime,
    in acsitypes::TimeStamp rangStopTime
    )
    raises (acsiexceptions::NoData);

  LogEntryList QueryLogByEntry (
    in EntryNo startEntry,
    in EntryNo stopEntry
    )
    raises (acsiexceptions::NoData);

};

interface ReportControl {
  attribute acsitypes::ObjectName RCName;
  attribute acsitypes::ObjectReference dataSetReference;
  attribute acsitypes::ULong bufTim;
  attribute TriggerOptions trgOps;
  attribute acsitypes::ULong intgPd;
  attribute acsitypes::VisibleString rptId;
  attribute ReportOptionalFields optFlds;
};

interface BufferedReportControl : ReportControl {
  /* Attributes ........................... */

  attribute BRCStateType bRCstate;
  attribute boolean discBuf;

  /* Services .............................. */

  // "Report" service should be implemented by clients

    void AckReport (
    in acsitypes::VisibleString rptId,
    in long seqNum,
    in long subSeqNum
    );

  // GetReportControlValue and SetReportControlValue are automatically
  // generated by the IDL compiler

  void RegisterAsClient (
    in ReportClient client
```

```
         );

      void UnRegisterAsClient (
         in ReportClient client
         );

   };

   typedef sequence <ReportControl> ReportControlList;

   /* Unbuffered Report Control ======================================= */

   interface UnbufferedReportControl : ReportControl {
      /* Attributes ............................. */

      attribute boolean rptEna;

      /* Services ................................ */

      // "Report" is implemented by clients and used by ReportControl

      // GetReportControlValue and SetReportControlValue are automatically
      // generated by the IDL compiler

      void RegisterAsClient (
         in ReportClient client
         );

      void UnRegisterAsClient (
         in ReportClient client
         );

   };

   typedef sequence <LogControl> LogControlList;

   typedef sequence <Log> LogList;

};

/* =====================================================================
DataSet   $$ 61850-7-2/11 $$
================================================================== */

module acsidataset {

   interface DataSet {
      attribute acsitypes::ObjectName dataSetName;
      attribute acsitypes::SelectorList listOfSelectors;

      acsitypes::ListOfDataValues GetDatSetValue (
         in boolean specificationWithResult
         )
         raises (acsiexceptions::GetFailed);

      acsitypes::ListOfConfirmations SetDatSetValue (
         in acsitypes::ListOfDataValues listOfDataValues
         )
         raises (acsiexceptions::SetFailed);

      void DeleteDataSet ()
         raises (acsiexceptions::DeleteFailed);

      acsitypes::SelectorList GetDataSetDirectory ()
         raises (acsiexceptions::FailedDirectory);

   };
   /* A Factory object should be created before to perform
   DataSet creation (And deletion?) */

   interface DataSetFactory {
      acsitypes::ObjectReference CreateDataSet (
         in acsitypes::ObjectName dataSetName,
         in acsitypes::SelectorList listOfSelectors
         )
         raises (acsiexceptions::CreateFailed);

      void DeleteDataSet (
         in acsitypes::ObjectReference dataSetReference
         )
         raises (acsiexceptions::DeleteFailed);
   };

   typedef sequence <DataSet> DataSetList;

};


/* ========================================================================
Setting Group Control  $$ 61850-7-2/13 $$
===================================================================== */

module acsisettinggroupcontrol {

   interface SettingGroupControl {
      attribute acsitypes::ObjectName sGCControlName;
      attribute acsitypes::SelectorList listOfSelectors;
      attribute acsitypes::ULong numOfSG;
      attribute acsitypes::ULong activeSG;

      boolean ActivateSG (
         in acsitypes::ULong sGNumber
         )
         raises (acsiexceptions::ActivateFailed);

      acsitypes::ListOfConfirmations SetSGValues (
         in acsitypes::ULong sGNumber,
         in acsitypes::ListOfDataValues dataValues
         )
         raises (acsiexceptions::SetFailed);

      acsitypes::ListOfDataValues GetSGValues (
         in acsitypes::ULong sGNumber
         )
         raises (acsiexceptions::GetFailed);

      boolean GetSGControlValues (
         out acsitypes::ULong activeSG,
         out acsitypes::ULong numOfSG,
         out acsitypes::SelectorList listOfSelectors
         )
         raises (acsiexceptions::GetFailed);

   };

   typedef sequence <SettingGroupControl> SettingGroupControlList;

};



/* =======================================================================
GSE
===================================================================== */
```

```
module acsigse {

   typedef sequence <long> ElementOffsets;

   struct DataItem {
      acsitypes::ObjectReference dataReference;
      acsitypes::DataAttributeValue value;
   };

   struct GOOSEMessage {
      acsitypes::ObjectReference datSetRef;
      acsitypes::VisibleString appId;
      acsitypes::TimeStamp t;
      acsitypes::ULong stNum;
      acsitypes::ULong sqNum;
      acsitypes::TestStatus test;
      long confRev;
      boolean needsCommissioning;
      boolean dataRefIncluded;
      sequence <DataItem> listOfData;
   };

   /* GSSE ============================================================ */

   typedef sequence <acsitypes::VisibleString> LabelList;

   typedef sequence <octet> Data;

   struct GSSEMessage {
      acsitypes::VisibleString appId;
      acsitypes::TimeStamp t;
      acsitypes::ULong stNum;
      acsitypes::ULong sqNum;
      acsitypes::TestStatus test;
      long confRev;
      Data listOfData;
   };

   interface GSEControl {
      attribute acsitypes::ObjectReference gCRef;
      attribute acsitypes::VisibleString appId;
   };

   /* GOOSE =========================================================== */

   interface GOOSEControl : GSEControl {
      /* Attributes ............................................... */

      attribute acsitypes::ObjectReference dataSetRef;
      attribute boolean needsCommissioning;

      /* Services ................................................. */

      // GetGSEControlValue() and SetGSEControlValue() equivalents are
      // generated by the IDL compiler

      acsitypes::ObjectReferenceList GetReference (
         in ElementOffsets listOfElementOffsets
         )
         raises (acsiexceptions::NoSuchData);

      ElementOffsets GetGSEElementNumber (
         in acsitypes::ObjectReferenceList listOfReferences
         )
         raises (acsiexceptions::NoSuchData);

   };
```

```
        interface GSSEControl : GSEControl {
          /* Attributes ........................................ */

          attribute LabelList dataLabels;
          attribute Data lastSentData;

          /* Services ........................................ */

          // GetGSEControlValue() and SetGSEControlValue() equivalents
          // are generated by the IDL compiler

          LabelList GetReference (
            in ElementOffsets listOfElementOffsets
            )
            raises (acsiexceptions::NoSuchData);

          ElementOffsets GetGSEElementNumber (
            in LabelList listOfLabels
            )
            raises (acsiexceptions::NoSuchData);

        };

        typedef sequence <GSEControl> GSEControlList;

      };

      /* ==================================================================
      SampledValue
      ================================================================== */

      module acsisampledvalue {

        /* SMV buffer ================================================== */

        struct SMVOptionalFields {
          boolean reserved;
          boolean refresh_time;
          boolean configuration_revision;
          boolean sample_rate;
        };

        /* The buffer definition contains places for indication
        of optional content but no place for the content itself !!! */

        struct SMVBuffer {
          acsitypes::VisibleString sMVId;
          SMVOptionalFields optFlds;
          acsitypes::ObjectReference dataSetRef;
          acsitypes::ListOfDataValues listOfSmp;
          acsitypes::ULong smpCnt;
          acsitypes::TimeStamp refrTime;
          acsitypes::ULong confRev;
          acsitypes::ULong smpRate;
        };

        interface SMVControl {
          attribute acsitypes::ObjectName sMVCNam;
          attribute acsitypes::VisibleString sMVId;
          attribute acsitypes::ObjectReference DatSetRef;
          attribute acsitypes::ULong smpRate;
        };

        /* MulticastSMVC ================================================== */

        interface MulticastSMVC : SMVControl {
          /* Services ........................................*/
          // GetMSMVCValue() and SetMSMVCValue() equivalents are generated
          // by the IDL compiler */

        };

        /* UnicastSMVC ================================================== */

        interface UnicastSMVC : SMVControl {
          /* Attributes ........................................... */

          attribute boolean sMVEna;

          /* Services ............................................. */

          // GetMSMVCValue() and SetMSMVCValue() equivalents are generated
          // by the IDL compiler

          // The GetNextUSMVC() service has been put in the LogicalDevice
          // interface (see $$61850-7-2/16.3.2.3$$). Perhaps it should be
          // put in the LogicalNodeZero interface :-(

        };

        typedef sequence <SMVControl> SMVControlList;

      };

      /* =====================================================================
      File
      ================================================================== */

      module acsifile {

        /* Note on getting big files: If files are large, returning a file
        can produce problems with the stack. This interface does not take
        this into account but it is possible to use an iterator to do this
        work. */

        typedef sequence <octet> FileData;

        struct FileEntry {
          acsitypes::VisibleString name;
          acsitypes::ULong fileSize;
          acsitypes::TimeStamp lastModified;
        };

        typedef sequence <FileEntry> FileEntryList;

        // Forward  declaration
        interface File;

        typedef sequence <File> FileList;

        interface File {

          /* Attributes ........................................... */

          attribute acsifile::FileEntry fileEntry;

          /* Services ............................................. */

          acsifile::FileData GetFile (
            in acsitypes::VisibleString fileName
            )
            raises (acsiexceptions::GetFailed);

          boolean SetFile (
            in acsitypes::VisibleString name,
            in FileData data
            )
            raises (acsiexceptions::SetFailed);

         boolean DeleteFile (
            in acsitypes::VisibleString name
            )
            raises (acsiexceptions::DeleteFailed);

         // The FileDirectory operation should return a full list
         // in the case of empty argument.

         acsifile::FileList FileDirectory (
            in acsitypes::VisibleString fileName
            )
            raises (acsiexceptions::FailedDirectory);
      };

   };



   /* ===================================================================
   LogicalNode  $$ 61850-7-2/9 $$
   ================================================================== */

   module acsilogicalnode {

      interface LogicalNode {
         /* Attributes .................................................... */

         attribute acsitypes::ObjectName logicalNodeName;
         attribute acsitypes::DataAttributeList listOfData;
         attribute acsidataset::DataSetList listofDataSets;
         attribute acsireportlog::ReportControlList lisOfReportControls;
         attribute acsireportlog::LogControlList listOfLogControls;
         attribute acsireportlog::LogList listofLogs;

         /* Services ....................................................
*/
         // The polimorphic service LogicalNodeDirectory is expanded into a
         // set of non-polimorphic services

         acsitypes::DataRefList LogicalNodeDataDirectory ()

            raises (acsiexceptions::FailedDirectory);

         acsitypes::DataSetRefList LogicalNodeDataSetDirectory ()

            raises (acsiexceptions::FailedDirectory);

         acsitypes::BRCRefList LogicalNodeBRCDirectory ()
            raises (acsiexceptions::FailedDirectory);

         acsitypes::URCRefList LogicalNodeURCDirectory ()
            raises (acsiexceptions::FailedDirectory);

         acsitypes::LogControlRefList LogicalNodeLogControlDirectory ()

            raises (acsiexceptions::FailedDirectory);

         acsitypes::LogRefList LogicalNodeLogDirectory ()
            raises (acsiexceptions::FailedDirectory);

         acsitypes::SettingGroupControlRefList
            LogicalNodeSettingGroupControlDirectory ()
            raises (acsiexceptions::FailedDirectory);
      acsitypes::GOOSEControlRefList LogicalNodeGOOSEControlDirectory ()

        raises (acsiexceptions::FailedDirectory);

      acsitypes::MulticastSMVCRefList LogicalNodeMulticastSMVCDirectory ()

        raises (acsiexceptions::FailedDirectory);

      acsitypes::UnicastSMVCRefList LogicalNodeUnicastSMVCDirectory ()

        raises (acsiexceptions::FailedDirectory);

    };

    interface LogicalNodeZero : acsilogicalnode::LogicalNode {
      attribute acsisettinggroupcontrol::SettingGroupControlList
              listOfSettingGroupControls;
      attribute acsigse::GSEControlList listOfGSEControls;
      attribute acsisampledvalue::SMVControlList listOfSampledValueControls;
    };

    interface LogicalNodePHD : acsilogicalnode::LogicalNode {
    };

    typedef sequence <LogicalNode> LogicalNodeList;

  };

  /* ==================================================================
  LogicalDevice  $$61850-7-2/8$$
  ================================================================== */

  module acsilogicaldevice {

    interface LogicalDevice {
      attribute acsitypes::ObjectName logicalDeviceName;
      attribute acsilogicalnode::LogicalNodeList logicalNodes;
      attribute acsilogicalnode::LogicalNodeZero logicalNodeZero;
      attribute acsilogicalnode::LogicalNodePHD logicalNodePHD;

      acsitypes::LogicalNodeRefList LogicalDeviceDirectory ()

        raises (acsiexceptions::FailedDirectory);

      // This service is an LD service but is described as a
      // service of the Unicast SMVC $$61850-7-2/16.3.2.3$$

      acsitypes::ObjectName GetNextUSMVC ()
        raises (acsiexceptions::NoAvailableUSMVC);

    };

    typedef sequence <LogicalDevice> LogicalDeviceList;

  };

  /* ================================================================
  Association  $$61850-7-2/7$$
  ==================================================================

  A simple userid/passwd authentication scheme is used. This is the level
  required by IEC 61850. Futher security can be attained by means of a
  CORBA Securiy Service Level 1 or Level 2.

  Views are implemented by means of selectors. */

  module acsiassociation {
    typedef acsitypes::VisibleString ViewId;

    struct View {
      ViewId id;
      acsitypes::Selector selector;
    };

    struct AuthenticationType {
      string userId;
      string passwd;
      string viewId;
    };

    typedef acsitypes::ObjectName AssociationIdent;

    typedef sequence <AssociationIdent> ClientAssociationList;

    /* Two-party association class (interface) ------------------------- */

    interface Association {
      /* attributes */

      attribute AssociationIdent associationId;
      attribute AuthenticationType authenticationParameter;

      /* Services */

      AssociationIdent Associate (
        in AuthenticationType authentication
        )
        raises (acsiexceptions::AssociateFailed);

      boolean Abort (
        in AssociationIdent association
        )
        raises (acsiexceptions::NoSuchAssociation);

      // Ignored "indication" argument in  61850

      boolean Release (
        in AssociationIdent association
        )
        raises (acsiexceptions::NoSuchAssociation);

    };

    /* Multicast association class (interface) --------------------------
    We cannot fullly understand how this class is goin to be used !!! */

    interface MulticastAssociation {
      /* attributes */

      attribute AuthenticationType authenticationParameter;
    };

  };

  /* ==================================================================
  Server  $$61850-7-2/6$$
  =============================================================== */

  module acsiserver {

    interface Server {
      /* Attributes */

      attribute acsitypes::ObjectReference serviceAccessPoint;
      attribute acsitypes::VisibleString configurationVersion;
      attribute acsilogicaldevice::LogicalDeviceList LogicalDevices;
      attribute acsifile::FileList Files;
      attribute acsiassociation::ClientAssociationList ClientAssociations;

      /* Services */

      acsilogicaldevice::LogicalDeviceList ServerLogicalDeviceDirectory ()

         raises (acsiexceptions::FailedDirectory);

      acsifile::FileList ServerFileDirectory ()
         raises (acsiexceptions::FailedDirectory);

      acsiassociation::ClientAssociationList
         ServerClientAssociationDirectory ()
         raises (acsiexceptions::FailedDirectory);

   };

};


/* ===========================================================================
Data   $$ 61850-7-2/10 $$
========================================================================= */

module acsidata {

   interface Data {
      attribute acsitypes::ObjectName dataName;
      attribute acsitypes::DataAttributeList dataAttributes;
      attribute acsitypes::RawData rawdata;

      acsitypes::DataAttributeValueList GetDataValues (
         in boolean specificationWithResult,
         inout acsitypes::SelectorList listOfSelectors
         )
         raises (acsiexceptions::GetFailed);

      acsitypes::ListOfConfirmations SetDataValues (
         in acsitypes::SelectorList listOfSelectors,
         in acsitypes::DataAttributeValueList dataValues
         )
         raises (acsiexceptions::SetFailed);

      acsitypes::DataAttributeRefList GetDataDirectory (
         in acsitypes::Selector selector
         )
         raises (acsiexceptions::FailedDirectory);

      acsitypes::DataAttributeRefList GetDataDefinition (
         in acsitypes::Selector selector
         )
         raises (acsiexceptions::FailedDirectory);

   };

};



/* ===========================================================================
Substitution   $$ 61850-7-2/12 $$
========================================================================= */

module acsisubstitution {

   interface Substitution {
      boolean Substitute (
         in acsitypes::ObjectReference dataAttributeReference,
         in acsitypes::DataAttributeValue substitutionValue
         )
         raises (acsiexceptions::SubstituteFailed);

      boolean UnSubstitute (
         in acsitypes::ObjectReference dataAttributeReference
         )
         raises (acsiexceptions::SubstituteFailed);
   };
};



/* ======================================================================
Control
====================================================================== */

module acsicontrol {

   /* As stated in $$61850-7-2/17.1$$ control objects can be any data class
   one of the following common data classes: SPC, DPC, IST, BST, IST, ASP */

   /* Service parameters for control interface ================= */

   struct Check {
      boolean synchrocheck;
      boolean interlock_check;
   };

   enum TimeActivatedOperateType {

      TIMER_ACTIVATED,
      COMMAND_EXECUTED
   };

   /* Additional cause diagnosis shall identify the reason of failure */

   enum AddCause {

      NOT_SUPPORTED,
      BLOCKED_BY_SWITCHING_HIERARCHY,
      BLOCKED_BY_EVENT,
      BLOCKED_BY_INTERLOCKING,
      BLOCKED_BY_SETPOINT_COMMAND,
      TARGET_EXISTS,
      PARAMETER_ERROR,
      TIME_LIMIT_OVER,
      ADDRESS_ERROR,
      HARDWARE_ERROR,
      ONE_OF_N_CONTROL,
      SYSTEM_CRASH,
      STEP_LIMIT,
      COMMAND_ALREADY_IN_EXECUTION,
      PLAUSIBILITY_ERROR,
      BLOCKED_BY_SYNCHROCHECK,
      DEBOUNCE_ACTIVE,
      ABORTION,
      PARAMETER_CHARGE_IN_EXECUTION,
      CB_ALARM
   };

   interface Control {
      void Select (
         in acsitypes::ObjectName name
         )
         raises (acsiexceptions::ControlFailed);
      // This service will return AddCause="No_error" in case Response-

      AddCause SelectWithValue (
         inout acsitypes::ObjectName name,
         inout acsitypes::DataAttributeValue value,
         inout acsitypes::TimeStamp time,
         inout acsitypes::TestStatus test
         )
         raises (acsiexceptions::ControlFailed);

      // This service will return AddCause="No_error" in case Response-

      AddCause Cancel (
         inout acsitypes::ObjectName name,
         inout acsitypes::TimeStamp time,
         inout acsitypes::TestStatus test
         )
         raises (acsiexceptions::ControlFailed);

      // This service will return AddCause="No_error" in case Response-

      AddCause Operate (
         inout acsitypes::ObjectName name,
         inout acsitypes::DataAttributeValue value,
         inout acsitypes::TimeStamp time,
         inout acsitypes::TestStatus test,
         in acsicontrol::Check check
         )
         raises (acsiexceptions::ControlFailed);

      // This service will return AddCause="No_error" in case Response-

      AddCause CommandTermination (
         out acsitypes::ObjectName name,
         out acsitypes::TimeStamp time,
         out acsitypes::TestStatus test
         )
         raises (acsiexceptions::ControlFailed);

      // This service will return AddCause="No_error" in case Response-

      AddCause Synchrocheck (
         inout acsitypes::ObjectName name,
         inout acsitypes::TimeStamp time,
         inout acsitypes::TestStatus test
         )
         raises (acsiexceptions::ControlFailed);

      // This service will return TimeActivatedOperateType in case Response+
      //  and AddCause in case Response-

      AddCause TimeActivatedOperate (
         inout acsitypes::ObjectName name,
         inout acsitypes::DataAttributeValue value,
         inout acsitypes::TimeStamp time,
         inout acsitypes::TestStatus test,
         out acsicontrol::TimeActivatedOperateType type
         )
         raises (acsiexceptions::ControlFailed);

   };

};

/* =================================================================
Time  $$61850-7-2/18$$
====================================================================
   61850-7-2/18 defines not a service but two types (Time and TimeQuality)
   that are so basic they have been moved to the "Types" section. */


#endif
```

22. ACSI Document Quick Reference

Chapters 5 to 19 of this mapping have a one-to-one correspondence with the IEC 61850 Part 7-2. As a fast access guide the following tables indicate the page numbers of the sections in the IEC 61850 Part 7-2 document (ACSI Page). Corresponding pages in this document can be found in the table of contents.

Section	Content	ACSI Page
5	ACSI communication service model	14
5.1	Abstract modelling techniques	14
5.2	Overview on ACSI models	16
5.3	Class model	18
5.4	Service tables	19

Section	Content	ACSI Page
6	Server model	20
6.1	Server class definition	20
6.1.1	ServiceAccessPoint	20
6.1.2	LogicalDevices	21
6.1.3	Files	21
6.1.4	ClientAssociations	21
6.2	Server services	21
6.2.1	ServerDirectory	21
6.3	Access control	22
6.3.1	Access control definition	23
6.3.2	Use of access control	25

Section	Content	ACSI Page
7	Application association model	27
7.1	Concept of abstract associations	27
7.2	Association class	28
7.2.1	Trigger policy	28
7.2.2	Queues and buffers	29
7.2.3	Connection and connectionless protocols	29
7.2.4	Establishing associations	29
7.2.5	Role	29
7.2.6	Cardinality	29

7.3	Association Types	30
7.3.1	Queued User Triggered Unidirectional (QUU)	30
7.3.2	Queued User Triggered Bi-directional Connection Oriented (QUB-CO)	30
7.3.3	Queued User Triggered Bi-directional Connectionless (QUB-CL)	30
7.3.4	Queued User Triggered Bi-directional Flow Control (QUB-FC)	30
7.3.5	Queued User Triggered Bi-directional Segmenting (QUB-Seg)	30
7.3.6	Buffered Network-Scheduled Uni-directional (BNU)	30
7.4	Definitions of association end points	31
7.4.1	Association end point state	31
7.4.2	Max outstanding requests calling	31
7.4.3	Max outstanding requests called	31
7.4.4	Max outstanding unconfirmed requests client (maxUCSC)	31
7.4.5	Max outstanding unconfirmed requests server (maxUCSS)	31
7.4.6	Maximum outstanding requests client (maxOSCC)	31
7.4.7	Maximum outstanding requests server (maxOSCS)	31

Section	Content	ACSI Page
8	Logical device model	32
8.1	Logical device class definition	32
8.1.1	logicalDeviceObjectName	33
8.1.2	LogicalNodes	33
8.1.3	AccessControl	33
8.2	Logical device access services	33
8.2.1	LogicalDeviceDirectory	33

Section	Content	ACSI Page
9	Logical node model	35
9.1	Logical node class definition	35
9.1.1	LnState	35
9.1.2	LogicalNodeObjectName	36
9.1.3	DataObjects	36
9.1.4	DataSets	36
9.1.5	ReportControls	36
9.1.6	LogControls	36
9.1.7	Logs	36

9.1.8	gOOSEControl	36
9.1.9	SampledValueControls	36
9.1.10	AccessControl	37
9.2	Logical node access services	37
9.2.1	LogicalNodeDirectory	37

Section	Content	ACSI Page
10	Data object model	39
10.1	Data object class definition	39
10.1.1	dataObjectName	39
10.1.2	DataAttributes	39
10.1.3	AccessControl	39
10.2	Data object class services	39
10.2.1	GetDataObjectValues	39
10.2.2	SetDataObjectValues	41
10.2.3	GetDataObjectDefinition	42
10.2.4	DataObjectDirectory	43

Section	Content	ACSI Page
11	Data attribute model	44
11.1	Data attribute class definition	44
11.1.1	DataAttributeName	44
11.1.2	DataAttributeContent	44
11.1.3	DataAttributeCharacteristic	44
11.2	DataAttribute Services	45
11.2.1	SetDataAttributeValues	45
11.2.2	SetDataAttributeValues	45
11.3	Attribute and parameter types	45
11.4	Common types	45
11.4.1	ObjectName	45
11.4.2	ObjectReference	45
11.4.3	TimeStamp	46
11.4.4	RelativeTime	47

Section	Content	ACSI Page
12	Data set model	48
12.1	Data set class definition	48
12.1.1	DataSetObjectName	48
12.1.2	DataObjectNames	48
12.1.3	AccessControl	48
12.2	Data set services	49
12.2.1	GetDataSetValue	49

12.2.2	SetDataSetValue	49
12.2.3	CreateDataSet	49
12.2.4	DeleteDataSet	50
12.2.5	DataSetDirectory	51

Section	Content	ACSI Page
13	Publish and subscribe data transfer (reporting and logging)	52
13.1	Overview	52
13.2	Report model	53
13.2.1	Report control class definition	53
13.2.2	Reporting services	59
13.3	Log model	64
13.3.1	Overview	64
13.3.2	Log control class definition	64
13.3.3	Log control attribute definition	65
13.3.4	Log class definition	66
13.3.5	Log attribute definition	67
13.3.6	Log and log control services	68

Section	Content	ACSI Page
14	Generic Object Oriented System-wide Event (GOOSE)	74
14.1	Introduction	74
14.2	Generic object oriented system-wide event (GOOSE) control class	74
14.2.1	GcName - GOOSE control name	75
14.2.2	GooseEna – GOOSE enable	75
14.2.3	SndgLD – Name of sending logical device	75
14.2.4	UserDatNam – user data name	75
14.3	Generic object oriented system-wide event (GOOSE) message	75
14.4	Service specification	76

Section	Content	ACSI Page
15	Control	77
15.1	Control Introduction	77
15.2	Control with normal security	77
15.2.1	Direct control with normal security	78
15.2.2	SBO control with normal security	78
15.3	Control with enhanced security	79

15.3.1	Direct control with enhanced security	79
15.3.2	SBO control with enhanced security	79
15.4	Typical control operations	80
15.4.1	Direct control with normal security	80
15.4.2	Select before operate with enhanced security	81
15.4.3	Control with synchrocheck	82
15.4.4	Time activated control	84
15.5	Control services	85
15.5.1	Service parameter definition	85
15.5.2	Service specification	86

Section	Content	ACSI Page
16	Substitution model	90
16.1	Substitution overview	90
16.2	Substitution services	90
16.2.1	Service parameter definition	90
16.2.2	Service specification	91
16.2.3	Service procedure	92

Section	Content	ACSI Page
17	Transmission of sampled measured values	93
17.1	Overview	93
17.2	Sampled Measured Value Control Class Definition	94
17.2.1	SvcNam	94
17.2.2	SvEna	94
17.2.3	DatSetNam	95
17.2.4	RefrRate	95
17.2.5	NoOfSmp	95
17.2.6	ComUpRate	95
17.2.7	SmpRate	95
17.3	Sampled Measured Value Service Definitions	95
17.3.1	GetSMVControlValues	95
17.3.2	SetSMVControlValues	95
17.4	Sampled Measured Value Buffer Format	96

Section	Content	ACSI Page
18	Time Synchronisation Model	98

Section	Content	ACSI Page
19	File transfer	99
19.1	File transfer model	99

23. DOTS General Model Update

23.1 Introduction

This annex reflects changes in DOTS general model on Communication Networks and Systems in Substations, from v04.mdl to v07.mdl. Principally, modifications are due to updates from revision of January 2001 to March 2001, in documents IEC 61850-7-3 and IEC 61850-7-4.

Parts 7-3 and 7-4 specify the basic communication structure for substations and feeder equipment. The first one includes Common Data Classes (attribute types and class specifications) while the second one delas with Compatible Loginal Node Classes and Data Classes.

23.2 Common Data Attribute Types

Main types remain the same. Only a few more types have been added.

23.3 Common Data Class Specifications

The following common data classes were specified in the old version:

Status Information

- Single Point Status (**SPS**)
- Double Point Status (**DPS**)
- Step Position Information (**SPI**)
- Status Indication Group (**SIG**)
- Integer Status (**ISI**)
- Protection Activation Information (**ACT**)
- Security Violation Counting (**SEC**)
- Name Plate (**PLATE**)
- Binary Counter Reading (**BCR**)

Measurand Information

- Measured Value (**MV**)
- WYE (**WYE**)
- Delta (**DEL**)

- Sequence (**SEQ**)
- Harmonics for WYE (**HVWYE**)
- Harmonics for DEL (**HVDEL**)

Controllable Status Information

- Controllable Single Point (**SPC**)
- Controllable Double Point (**DPC**)
- Controllable Integer Status (**ISC**)
- Binary Controlled Step Position Information (**BST**)
- Integer Controlled Step Position Information (**IST**)

Controllable Analogue Information (Set Point)

- Analogue Set Point (**ASP**)
- Pick up Group (**PUG**)
- Setting Curve (**CURVE**)

In the new version, HV has been added while DPS, SPI and PUG have been removed.

23.3.1 Data Classes

In the old version we had:

Measurands and metered values

- Measured values and analogue setpoints
- Metered values

Commands

- System commands and system return information
- LN specific commands and their related return information

Status

- System information
- LN status indications
- LN monitoring indications
- LN functional indications (fault indications for protection)

In the new version, the data has been reorganised as follows:

System information

- Basic LN class information
- Physical device information

Measured values and analogue setpoints

Measurand Identification

Metered values

Controllable data

Controllable data

Status Information

Settings Information

Additionally, all the inherited object from these data classes have drastically changed. This is due to the fact that they are components of Logical Node classes, and these have been completely redifined. Changes in DOTS model have been made restarting from the beginning.

23.3.2 Data Attribute Class

Data Attributes have also changed a bit due to modifications in Logical Nodes. This does not strongly affect Logical Nodes definition.

23.3.3 Logical Node Classes

Many logical nodes have been kept in the new release. Nevertheless, the main modifications follow:

- Logical nodes kept have seen their attributes completed and/or importantly changed.
- New logical nodes have appeared.
- Logical nodes classification has been substantially changed.

First type of modification has been the most important in model updating. Changes in structure follow.

Old structure:

Logical node zero
Protection related
Protection functions
Supervisory control
Switchgear
Instrument transformers
Power transformer related
Further Power System Equipment

New structure:

System Logical Nodes
Control
Switchgear
Instrument Transformers
Power Transformer Related
Further Power System Equipment
Protection Related
Protection Functions
Generic References
Interfacing and archiving
Automatic Control
Metering and Measurement

23.4 Conclusion

Due to the important modifications carried out in the model, it is not possible to compare both versions with simple tables indicating corresponding changes. This means that it is not recommendable to try to adapt any old developement to the new version, but to redevelope everything. The most important changes have been made in Data Classes, and this affects all Logical Nodes structure.

24. Comparison between IEC 61850 Draft Standard and IEEE 1525 Draft Standard

24.1 The IEEE 1525 specification

The reference document is Draft IEEE 1525, Standard for Substation Protection, Control
and Data Acquisition Communications (P1525 Draft 4r3, December 2000).

24.2 The IEEE 1525

This standard applies to systems used to communicate between intelligent electronic devices (IEDs) for substation integrated protection, control and data acquisition. The requirements of this standard are in addition to those contained in standards for individual devices (e.g., relays, switchgear).

This standard establishes the requirements for developing detailed object models for substation IEDs. The object models define the data structures and methods required for communication transaction processing between IEDs. Abstract Syntax Notation One (ASN.1) is used to define the data structures. Functional and performance requirements for communication are also defined.

Use this standard as part of a specification to develop new, or modify existing substation automation capability to ensure open system communication interfaces for substation IEDs.

This standard specifies a suite of communication protocols based on TCP/IP for connection-oriented communications and UDP/IP for connectionless oriented protocols. Other protocol suites may be used to provide communication services that are functionally equivalent to those specified for TCP/IP and UDP/IP.

24.3 Comparison with IEC 61850

Both draft standards are orientated to the same subject, Communications in Substations, but the focus is quite different.

The approach of IEEE 1525 is beginning in the physical layer, specifying bottom-up the different requirements. First in Clause 4, this norm specifies Internet Protocols and performance criteria to accomplish. Requirements for deploying multicast in a substation are described in Clause 5. Network security is specified in Clause 6, based in encryption and authentication using Internet Protocol Security (IPSec) protocols. Substation data model requirements are described in Clause 7. Substation communication capabilities required are described in Clause 8 and system requirements are described in Clause 9.

On the contrary, the point of view of IEC 51850 is the system developers one, and so, is a top-down functional approach. The starting point is Part 3, devoted to the General Requirements of the global system: quality requirements as reliability, availability, maintainability and security, environmental conditions and auxiliary services. Part 4 establishes specifications for system and project management. Part 5 describes the communication requirements for functions and device models. The Substation Configuration Language, SCL, is specified in Part 6 for describing IED configurations and communications systems.

Part 7 is the centre of IEC standard. It describes the Basic communication structure for substation and feeder equipment, defining the Abstract Communication Service Interface, ACSI, Common data classes, Compatible logical node classes and data classes. As already was said in point 1.2 of this document, ASCI is an abstract interface and, so, this part of IEC 61850 does not specify individual implementations and does not specify the mapping of the abstract functionality to standard application layers. In consequence ACSI needs specific communication service mapping, SCSM, as those described in Parts 8 and 9 of IEC draft, or the Mapping to CORBA described in this document.

In conclusions, IEC 61850 and IEEE 1525 could be consider as complementary specifications: IEC covers high levels of the substation application while IEEE covers low ones. From this point of view it is possible to think in developing a specific communication service mapping from IEC 61850 ACSI to IEEE 1525.

25. Comparison between IEC 61850 Draft Standard and OMG UMS-DAF

25.1 Contents

This chapter contains a comparison between "Utility Management System Data Access Facility UMS-DAF" of the Object Management Group, Inc. with the Specific Communication Service Mapping "SCSM" developed in *Distributed Object Telecontrol Systems and Networks DOTS, IST Project 1999-10258.*

Main documents consulted were:

- "Utility Management System (UMS) Data Access Facility". OMG, Version 1.0, June 2001
- "Specific Communication Service Mapping. Mapping to CORBA". July 2001. IST 10258-DOTS. Version 1.8

25.2 UMS Introduction (from the referred document)

Utilities operate their water, gas, or power assets through control systems. The scope of control may include production facilities, bulk transmission networks, distribution networks, and supply points.

The most basic control system provides Supervisory Control and Data Acquisition (SCADA) functions. More sophisticated systems provide simulation and analysis applications that help the operators optimize performance, quality, and security of supply.

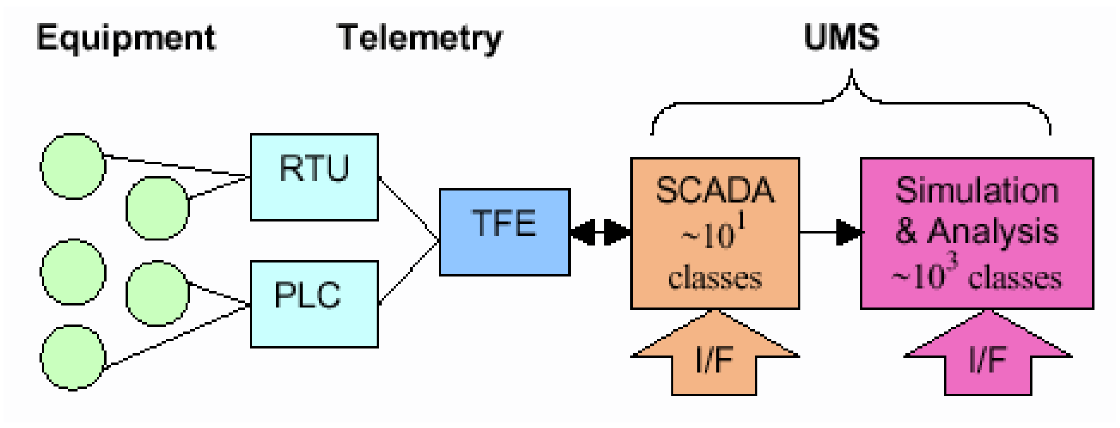


Figure 2: A global view of an Utility Management System.

We will use the umbrella term Utility Management System (UMS) to refer to this class of system. It covers Water Quality and Energy Management Systems (WQEMS) in the water sector, and Energy Management Systems (EMS) or Distribution Management Systems (DMS) in the power sector.

Figure 1-1 illustrates a UMS. Two interfaces are shown as block arrows: SCADA interface (which will be covered in other RFPs) and the analysis data interface (the subject of this specification).

The applications in a UMS employ extensive physical models representing networks, production facilities, and demand behavior among other things. These models distinguish a UMS from other types of control system.

For example, a power system model in an EMS may contain several hundred classes representing both physical and circuit theoretical concepts. Parts of this model must be understood by any external application that hopes to interpret the simulation and analysis results.

However, in all extant systems, this model is implemented in one or another proprietary database management system. There are no standard query languages or APIs for today’s EMS, DMS, or WQEMS resident data.

This sets the UMS data access problem apart from conventional database access on the one hand, and from SCADA data access on the other where just a few classes are involved representing generic concepts such as measure and device.

Notwithstanding the difficulties, the need for inter-operation between the UMS and other applications or systems is evident. The UMS automates key utility activities that touch on many other parts of the business. Moreover, UMS functions are being redefined as the utility business environment is reshaped. Consequently, new inter-operation requirements are steadily emerging.

The goal of the Utility Management System Data Access Facility is to improve the interoperability of these UMS applications with other applications and systems.

The goal of the UMS-DAF is to improve the interoperability of these UMS applications with other applications and systems.

The DAF is distinguished from some other database APIs because it can be applied to very simple systems, which may not contain a full-blown database management system.

Accordingly, the DAF defines very few interfaces, and does not require implementations to manage large, dynamic populations of CORBA objects. Most activity centers on the interface **ResourceQueryService**, which defines a small but sufficient set of queries as methods. The queries defined by the DAF are simple enough to be implemented in any UMS database and many related systems and applications.

The information made available through DAF interfaces can be described by one o more *schema.* A schema has the following contents:

- Simple Values
- Resource Descriptors
- Resource Query Service
- Resource Identifiers
- Query Sequence
- DAF Events
- Current Version and Transactions

When DAF is used to access power system model data, a schema derived from EPRI CIM (Electric Power Research Institute Common Information Model) will be employed. This schema is undergoing standardization in the IEC under Technical Committee TC57.

25.3 Characteristics of UMS-Data Access Facility

The mentioned document explain that DAF formulates queries and their results in terms of resources, properties, and values.

The *ResourceQueryService* performs single queries by means of following operations:

- get_values()
- get_extent_values()
- get_related_values()
- get_descendent_values()

The class is given by its **ClassID** and the source by a **ResourceID.**

25.4 Comparison with IEC 61850 and DOTS

UMS-DAF is the "analysis data interface" for Off-line applications with Complex Data Models and Complex algorithms or techniques and not cover the interface with other RFPs.

The advantages of UMS-DAF are well described in the mentioned document.

IEC 61850 and DOTS are interfaces with RFPs. this difference of goals is owing to that in real-time applications in the Energy Management Systems (EMS) o Distribution Management Systems (DMS):

- Applications needs data interchange, not only data access. The meaning of client and server is less restrictive
- Remote Call Procedures is necessary.
- Complete Object Oriented functionality is needed.
- In Real-Time Systems in order to reduce data volume, some times you only ask for values recently modified

DOTS covers UMS-DAF functionality, the *ResourceQueryService* of UMS-DAF performs single queries by means of following operations:

- get_values()
- get_extent_values()
- get_related_values()
- get_descendent_values()

The class is given by its **ClassID** and the source by a **ResourceID.**

DOTS offers access to Data Object Model, Data Attribute Model, Server Model, etc. IEC 61850 Model are not pure Object Oriented and present the difficulties mentioned in 4.3 paragraph of this document:

- Partial O.O. implementation
- Violations of encapsulation
- IEC 61850 is not yet finished
- Loose typing and unnecessary polymorphism

DOTS proposal solve this disadvantages and may be considered as equilibrated task force to move IEC 61850 towards a standard interoperability.

In this way, is possible to consider UMS-DAF a subset of DOTS and some works of normalization of names (ex. Path or Directory) and criteria (errors treatment, some structures) may produce a rational standard for EMS and DMS systems.

26. Index