INTERNATIONAL TELECOMMUNICATION UNION

ITU-T G.851.1

(11/96)

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

SERIES G: TRANSMISSION SYSTEMS AND MEDIA, DIGITAL SYSTEMS AND NETWORKS

Digital transmission systems – Digital networks – Telecommunications management network

---

Management of the transport network – Application of the RM-ODP framework

ITU-T Recommendation G.851.1

(Previously CCITT Recommendation)

ITU-T G-SERIES RECOMMENDATIONS

TRANSMISSION SYSTEMS AND MEDIA, DIGITAL SYSTEMS AND NETWORKS

INTERNATIONAL TELEPHONE CONNECTIONS AND CIRCUITS	G.100–G.199
INTERNATIONAL ANALOGUE CARRIER SYSTEM	
GENERAL CHARACTERISTICS COMMON TO ALL ANALOGUE CARRIER-TRANSMISSION SYSTEMS	G.200–G.299
INDIVIDUAL CHARACTERISTICS OF INTERNATIONAL CARRIER TELEPHONE SYSTEMS ON METALLIC LINES	G.300–G.399
GENERAL CHARACTERISTICS OF INTERNATIONAL CARRIER TELEPHONE SYSTEMS ON RADIO-RELAY OR SATELLITE LINKS AND INTERCONNECTION WITH METALLIC LINES	G.400–G.449
COORDINATION OF RADIOTELEPHONY AND LINE TELEPHONY	G.450–G.499
TRANSMISSION MEDIA CHARACTERISTICS	G.600–G.699
DIGITAL TRANSMISSION SYSTEMS	
TERMINAL EQUIPMENTS	G.700–G.799
General	G.700–G.709
Coding of analogue signals by pulse code modulation	G.710–G.719
Coding of analogue signals by methods other than PCM	G.720–G.729
Principal characteristics of primary multiplex equipment	G.730–G.739
Principal characteristics of second order multiplex equipment	G.740–G.749
Principal characteristics of higher order multiplex equipment	G.750–G.759
Principal characteristics of transcoder and digital multiplication equipment	G.760–G.769
Operations, administration and maintenance features of transmission equipment	G.770–G.779
Principal characteristics of multiplexing equipment for the synchronous digital hierarchy	G.780–G.789
Other terminal equipment	G.790–G.799
DIGITAL NETWORKS	G.800–G.899
General aspects	G.800–G.809
Design objectives for digital networks	G.810–G.819
Quality and availability targets	G.820–G.829
Network capabilities and functions	G.830–G.839
SDH network characteristics	G.840–G.849
Telecommunications management network	**G.850–G.859**
DIGITAL SECTIONS AND DIGITAL LINE SYSTEM	G.900–G.999
General	G.900–G.909
Parameters for optical fibre cable systems	G.910–G.919
Digital sections at hierarchical bit rates based on a bit rate of 2048 kbit/s	G.920–G.929
Digital line transmission systems on cable at non-hierarchical bit rates	G.930–G.939
Digital line systems provided by FDM transmission bearers	G.940–G.949
Digital line systems	G.950–G.959
Digital section and digital transmission systems for customer access to ISDN	G.960–G.969
Optical fibre submarine cable systems	G.970–G.979
Optical line systems for local and access networks	G.980–G.999

For further details, please refer to ITU-T List of Recommendations.

ITU-T RECOMMENDATION G.851.1

MANAGEMENT OF THE TRANSPORT NETWORK-APPLICATION OF THE RM-ODP FRAMEWORK

Source

ITU-T Recommendation G.851.1 was prepared by ITU-T Study Group 15 (1993-1996) and was approved under the WTSC Resolution No. 1 procedure on the 8th of November 1996.

FOREWORD

ITU (International Telecommunication Union) is the United Nations Specialized Agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of the ITU. The ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Conference (WTSC), which meets every four years, establishes the topics for study by the ITU-T Study Groups which, in their turn, produce Recommendations on these topics.

The approval of Recommendations by the Members of the ITU-T is covered by the procedure laid down in WTSC Resolution No. 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

INTELLECTUAL PROPERTY RIGHTS

The ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. The ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, the ITU had/had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

CONTENTS

Page

Page

Recommendation G.851.1

MANAGEMENT OF THE TRANSPORT NETWORK – APPLICATION OF THE RM-ODP FRAMEWORK

(Geneva, 1996)

1 Scope

This Recommendation provides a description of the modelling concepts upon which the ITU-T Study Group 15 network level model is based. The model uses the Reference Model of Open Distributed Processing (RM-ODP) framework [1], [2], [3] and [4] as a starting point to define a prescriptive methodology for defining the network level model.

A process used to develop a complete set of specifications based on this methodology is described in Appendix V.

1.1 Objectives

This Recommendation has the following objectives:

– to define a methodology for specifying a network management model to support interfaces to a network operations system within the TMN architecture, initially using OSI management with other infrastructures (e.g. CORBA IDL and ODP functions) to be used, as the standards for them become available;

– to produce models that can be flexibly distributed amongst different management system architectures (e.g. using OSI management, or distributed processing based systems);

– the work shall provide for maximum reusability of specifications and processing entities;

– this work shall retain maximum compatibility with existing OSI management based Recommendations and should be seen as an extension to that installed base.

NOTE – Implementation independent enterprise, information and computational viewpoints are being defined by this methodology. In this way the definitions which are used for the definition of OSI management, for example, are equally applicable to subsequent definitions using other infrastructures. Therefore, while the initial Recommendations which use this methodology will be targeted for the OSI management infrastructure, the models defined in these Recommendations for the enterprise, information and computational viewpoints will be independent of infrastructure.

1.2 Principles

The RM-ODP has been selected in order to provide a framework for a rigorous specification technique which is traceable to requirements.

The selection of techniques is guided by the objective of retaining maximum compatibility with OSI management.

Maintain compatibility with existing OSI management based Recommendations (e.g. M.3100, G.774-X, X.700-Series on OSI Management, Q.821, Q.822, etc.).

Maintain a balance between the requirements for human readability, machine readability and precision (e.g. use of formal notations).

This Recommendation should facilitate the interworking of applications based on implementations using different infrastructures.

1.3 Structure of this Recommendation

Clause 2 lists the references used by this Recommendation; clause 3 contains definitions which have been introduced in this Recommendation; and clause 4 defines the abbreviations used. Clause 5 provides an overview of the methodology described in the remainder of this Recommendation. Clauses 6, 7, 8 and 9 provide specific details of the RM-ODP viewpoints used in the methodology which includes the enterprise, information, computational and engineering viewpoints respectively. Clause 10 describes the use of ensembles in the definition of network management applications.

Annexes A, B, C and D contain descriptions of the templates used to define the enterprise, information, computational and engineering viewpoints respectively. Annex E defines the labelling syntax used within the methodology, and Annex F describes the template used for defining a management ensemble.

Appendices I through V contain various examples of the application of the methodology to solving network management problems and ways of using the methodology to develop standard specifications. Appendix VI defines the mapping between the various RM-ODP viewpoints used in the methodology. Appendix VII provides some guidelines for the use of Z in the information viewpoint.

2 References

The following ITU-T Recommendations contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations are subject to revision; all users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations. A list of the currently valid ITU-T Recommendations is regularly published.

[1] ITU-T Recommendation X.901[1] | ISO/IEC 10746-1[1], *Information technology – Basic reference model of open distributed processing: Overview.*

[2] ITU-T Recommendation X.902 (1995) | ISO/IEC 10746-2:1996, *Information technology – Open Distributed Processing – Reference Model: Foundations.*

[3] ITU-T Recommendation X.903 (1995) | ISO/IEC 10746-3:1996, *Information technology – Open Distributed Processing – Reference Model: Architecture.*

[4] ITU-T Recommendation X.904[1] | ISO/IEC 10746-4[1], *Information technology – Open Distributed Processing – Reference Model: Archtiectural semantics.*

[5] CCITT Recommendation X.722 (1992) | ISO/IEC 10165-4:1992, *Information technology – Open Systems Interconnection – Structure of Management Information: Guidelines for the definition of managed objects.*

[6] ITU-T Recommendation X.725 (1995) | ISO/IEC 10165-7:1996, *Information technology – Open Systems Interconnection – Structure of Management Information: General Relationship Model.*

[7] SPIVEY (J.M.): The Z Notation – A Reference Manual, 2nd Edition, *Prentice Hall International*, ISBN 0-13-978529-9, 1992.

[8] NM Forum 025, OMNIPoint 1: The "Ensemble" Concepts and Format, August 1992.

[9] ITU-T Recommendation X.724 (1996) | ISO/IEC 10165-6:1997, *Information technology – Open Systems Interconnection – Structure of Management Information: Requirements and*

[1] Presently at the stage of draft.

guidelines for implementation conformance statement proformas associated with OSI management.

[10] ITU-T Recommendation G.852.1 (1996), *Management of the transport network – Enterprise viewpoint for simple subnetwork connection management.*

[11] ITU-T Recommendation G.774 (1996), *Synchronous Digital Hierarchy (SDH) management information model for the network element view.*

[12] CCITT Recommendation X.721 (1992) | ISO/IEC 10165-2:1996, *Information technology – Open Systems Interconnection – Structure of Management Information: Definition of management information.*

3 Definitions

This Recommendation defines the following terms.

3.1 contract type: An expression of all the contract features (i.e. required, subject to negotiation or optional).

3.2 contract instance: The result of a negotiation between a provider and a given client.

4 Abbreviations used in this Recommendation

This Recommendation uses the following abbreviations.

ASN.1	Abstract Syntax Notation One
BEO	Basic Engineering Object
BNF	Backus-Naur Form
CMIP	Common Management Information Protocol
CMISE	Common Management Information Service Element
CORBA	Common Object Request Broker Architecture
DCE	Distributed Computing Environment
DPE	Distributed Processing Environment
GDIO	Guideline for the Definition of Information Objects
GDMO	Guidelines for the Definition of Managed Objects
GRM	General Relationship Model
IDL	Interface Definition Language
ITU-T	International Telecommunication Union – Telecommunication Standardization Sector
MOCS	Managed Object Conformance Statement
ODL	Object Definition Language
ODP	Open Distributed Processing
OMG	Object Management Group
OSI	Open Systems Interconnection
PICS	Protocol Implementation Conformance Statement
SDH	Synchronous Digital Hierarchy
SMI	Systems Management Information
SNC	Subnetwork Connection

SNMP Simple Network Management Protocol

TMN Telecommunications Management Network

5 Methodology overview

5.1 Introduction

From a network management point of view, the RM-ODP provides an object oriented framework (for distributed management systems) that allows user requirements for each management application (e.g. configuration management), the information (or data) related to the resources to be managed and the way that the information may be accessed and manipulated to be defined in a way that is essentially independent of the technology and distribution used in the implementation of a management system.

The RM-ODP framework provides five viewpoints of the system and resources being managed. This is illustrated in Figure 1.

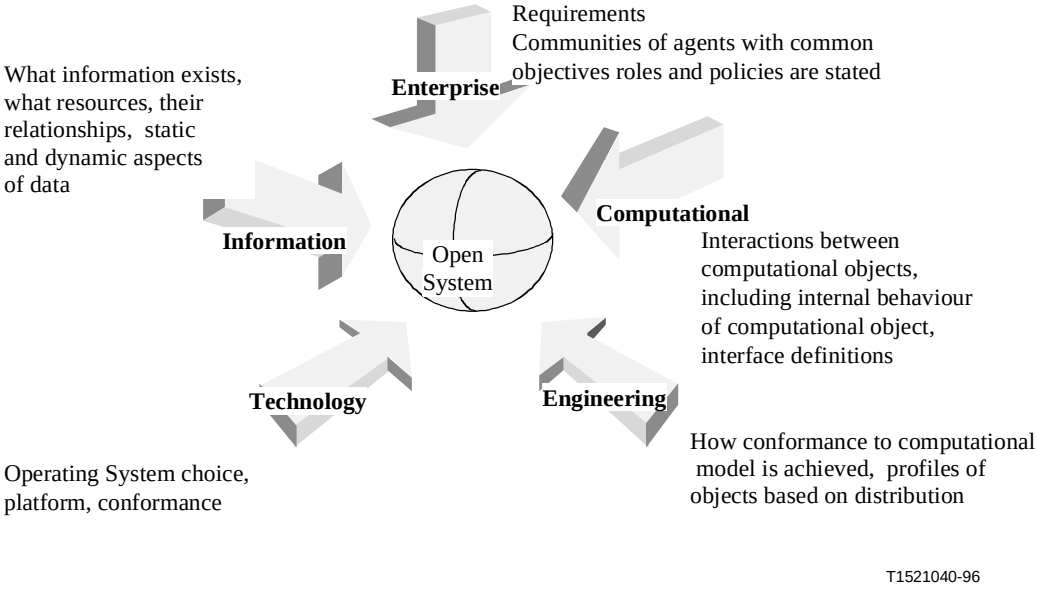


Figure 1/G.851.1 – RM-ODP viewpoints

The RM-ODP provides:

- a set of concepts to describe open distributed systems;
- independence with regard to any existing analysis and/or design and/or programming method/language.

The initial network level modelling work uses only the enterprise, information, computational and engineering viewpoints from RM-ODP. The technology viewpoint is not currently considered to be within the scope of the Telecommunication Standardization Sector of the ITU.

The information and computational viewpoints of the network level model focus on the semantics rather than the syntax of the information that must be exchanged for each individual or set of management applications. This level of definition is sufficient to allow more detailed specification of the system in the engineering viewpoint where the interoperability of the system is defined by the external interfaces. Conformance requirements will typically be defined in relation to these external interfaces. The engineering viewpoint shall be defined for a specific communications technology, such as OSI systems management.

This methodology document defines the viewpoints used to define the network level model.

5.1.1 RM-ODP viewpoint descriptions

Complete descriptions of the RM-ODP viewpoints can be found in the references listed in clause 2. A brief description of RM-ODP viewpoints follows:

- Enterprise viewpoint: A viewpoint on an ODP system and its environment that focuses on the purpose, scope and policies of that system. An enterprise specification enables the client of an ODP system to express its requirements and policies thereby establishing a contract between the client and the provider of the ODP system. Thus, an enterprise specification must be expressed in terms that are understandable by both parties.
- Information viewpoint: A viewpoint on an ODP system and its environment that focuses on the semantics of information and information processing activities in that system. An information viewpoint focuses on the definition of information object types, together with their relationships, their state values and permitted state changes.
- Computational viewpoint: A viewpoint on an ODP system and its environment that focuses on functional decomposition into structures suitable for distribution. A computational specification describes computational object types and their interface types. Instances of these object types will cooperate with each other through interfaces. Operational interface types are defined by specifying their operation signatures and associated behaviour. All computational objects are potentially distributed into separate systems. In the context of this methodology, any computational objects that are defined cannot be further distributed unless further computational interfaces are defined.
- Engineering viewpoint: A viewpoint on an ODP system and its environment that focuses on the functions required to support distribution in that system. An engineering specification is used to make actual decisions regarding distribution of the computational objects and, in addition, determine the infrastructure components required to support the distributed objects.
- Technology viewpoint: A viewpoint on an ODP system and its environment that focuses on the choice of technology in that system.

5.2 Using RM-ODP viewpoints in a specification design methodology

Although the reference model for open distributed processing provides an approach to modelling (i.e. the RM-ODP viewpoints), it does not provide a prescriptive methodology that can be followed in developing a system. This subclause introduces the methodology, using the RM-ODP viewpoints, for use in the standardization of transport network management models, whereby the application components are viewed as objects communicating through well-defined interfaces, and for which only the externally observable behaviour is described in an implementation-independent manner. The methodology uses the following steps:

a) requirements are identified;

b) information to describe the system is defined;

c) processes that manipulate the information and provide the services are described; and

d) distribution and implementation decisions are made.

Each of these design steps is associated with an RM-ODP viewpoint. In practice the temporal ordering of the viewpoints is not required. What is more important is the separation of concerns provided by the RM-ODP framework. An example of a specification process based on this methodology is shown in Appendix VI and the relationship between the viewpoints is shown in Appendix VII.

The set of resources that form a transport network architecture can be described using the RM-ODP enterprise and information viewpoints only, whereas a network management service requires the addition of the computational viewpoint. Both the transport network architecture and service definition will be merged during the engineering process to develop management capabilities at, for example, a Q3 interface.

5.3 Tracing between the viewpoints

This methodology allows tracing between requirements and the resulting engineering specifications. This is achieved by structuring the document in a way that allows a direct relationship between viewpoints for each enterprise community. For example, tracing from the engineering specification back to the requirements is also assured by use of a labelling scheme described below.

5.3.1 Labelling

A label will be provided to specific elements of specification allowing it to be referenced from other specifications. This reference may be made in the same viewpoint, across several viewpoints of the same service specification or across several service specifications (e.g. when using an IMPORT clause).

5.3.1.1 Label declaration structure

The structure of labels uses a BNF-based structure.

The label reference structure is organized as a tree. Each node in the tree will be either a pre-defined keyword (as defined in this Recommendation, e.g. ATTRIBUTE, COMMUNITY, ROLE), or an element label that is provided to identify the element of specification, e.g. networkTTP, operational_state. When defined the element label must be unique within the context of its immediately superior node. When used as a reference the label must be unique within the context. The label reference must be provided to the level that provides a unique pointer to the section of the specification that is being referenced. The element label is defined as a text string, the label reference is enclosed by <>. For example, for label references within a single recommendation, the recommendation element of the label reference need not be provided, or within a viewpoint specification, reference to other entities within that viewpoint need not include the viewpoint element label.

The BNF definition for the label structure is specified in Annex E.

6 Enterprise viewpoint

The enterprise viewpoint is defined in RM-ODP Part 3 (ITU-T Rec. X.903 | ISO/IEC 10746-3 [3]), it defines the purpose, scope and policies of an ODP system. Its objective is to specify the requirements of the system from the perspective of all participants. These requirements are expressed in terms of the interactions between the system and the user environment. Any interaction between management applications that impacts the way that the system behaves must be defined in the enterprise viewpoint.

The enterprise viewpoint is included in the network level model in order to:

– document the use of the various portions of the model based on **communities** of common functions (applications); and

– provide a way of specifying the requirements upon which the model is defined.

The templates and the language used in this specification conform to the requirements defined in Part 3 of the RM-ODP [3].

The templates are used to describe **communities** which represent a related set of functions to meet a specific objective (management application) such as connection management. The enterprise community specifies the scope of a specific problem being addressed. The community comprises a set of roles, a set of actions and a set of policies to satisfy the cooperative objective, or contract, that is shared between the roles. A community does not specify objects – only the roles they play. Hence a given object may play a number of different roles in different communities. A community does not specify transparencies. One community supports a number of different instances of the community contract between roles. Each role (or set of roles) has an individual contract with another role (or set of roles). This instance of the contract will be a particular selection of service features from the available set.

Associated with each community description are the following:

- Contract.
- Community roles.
- Community policies.
- Community actions.
- Community activities.
- Service.
- Service feature.

The following subclauses describe these enterprise components in more detail. The templates and language to define them are described in Annex A.

6.1 Scope of the enterprise viewpoint

The client/provider relationship is provided by a contract establishment. A given contract reflects a service with an associated quality that is offered by the provider to its client and accepted by it.

A service contract will include all the features that enable the service type definition. It includes:

- a community definition;
- the list of roles involved in the community;
- the policies applicable to the community;
- each action and associated policies;
- each activity, if any.

6.2 Concepts

6.2.1 Community

The requirements are captured by first identifying the communities in the ODP system where a community is a group of roles that have come together for some objective. The objective of the community needs to be expressly articulated. Typically, the objective of the community is the provision of a particular service, e.g. subnetwork connection management community, resource management community, etc.

RM-ODP Part 3 [3] gives the following definition:

"Community: A composition of objects formed to meet an objective. The objective is expressed as a contract which specifies how the objective can be met."

The community is defined by its purpose (i.e. the common objective of the roles involved in the community), the definition of each role implied in the community and the policy applicable to the entire community.

6.2.2 Contract

The result of the negotiation of a service is a contract instance which reflects the agreed selection from the feature set of the supplier. These service features can be captured as a set of supported actions, activities and where applicable supported policies.

RM-ODP Part 2 [2] gives the following definition for a contract:

"Contract: An agreement governing part of the collective behaviour of a set of objects. A contract specifies obligations, permissions and prohibitions for the objects involved. The specification of a contract may include:

a) a specification of the different roles that objects involved in the contract may assume, and the interfaces associated with the roles;

b) quality of service attributes;

c) indications of duration or periods of validity;

d) indications of behaviour which invalidates the contract;

e) liveness and safety conditions."

Some contract features will be required for all the clients of a given provider (constraining the provider and/or the clients behaviour), others will be negotiated between the provider and each client before contract establishment, leading to supported or non-supported features as a result of this negotiation process. In addition, some contract features may stay optional after the contract establishment and be used by clients and/or the provider on a negotiated or local policy basis (such as "best effort").

To reflect this policy, an enterprise specification will have two parts. The first will reflect the expression of all the contract features (i.e. required, subject to negotiation or optional). This first part will then be used by a provider to establish the second part as the result of a negotiation between the provider and a given client. In this sense, the first part can be considered as a "service contract type" while the second part can be considered as a "service contract instance".

The contract negotiation process is beyond the scope of the methodology.

NOTE – This methodology does not distinguish between interfaces and roles in the enterprise viewpoint.

6.2.3 Roles

This methodology uses the enterprise role concept as is defined in Part 2 of RM-ODP [2]. For each service all the roles will be defined.

A caller role represents the behaviour of an enterprise object that defines the service requests of a given service.

A provider role represents the behaviour of an enterprise object that performs the service requests of a given service.

The other roles of a service represent the behaviour of enterprise objects reflecting the involvement of the resources in the context of this service.

6.2.4 Policies

The community policy is specified as a set of permissions, obligations, prohibitions and exceptions applicable for either the client or the provider with regard to the community RM-ODP Part 2 [2] provides the following definitions:

"Policy: A set of rules related to a particular purpose. A rule can be expressed as an obligation, a permission or a prohibition.

NOTE – Not every policy is a constraint. Some policies represent an empowerment."

"Obligation: A prescription that a particular behaviour is required. An obligation is fulfilled by the occurrence of the prescribed behaviour."

"Permission: A prescription that a particular behaviour is allowed to occur. A permission is equivalent to there being no obligation for the behaviour not to occur."

"Prohibition: A prescription that a particular behaviour must not occur. A prohibition is equivalent to there being an obligation for the behaviour not to occur."

6.2.5 Actions

According to RM-ODP Part 2 [2], an action is defined by:

Action: Something which happens.

Each community has a set of actions which support the community purpose. The actions will be used to express the service requests and associated responses that are exchanged between the client and the provider. Typical actions include: connection setup, connection release, and modify connection. These actions can be grouped into activities which specify the order in which actions can occur.

Each action is defined by the action-name and the specification of the action policy.

Associated with each action are a set of **action policies** which are described as either an obligation, permission, or prohibition applicable for either the client or the provider with regard to the community. These policies state the role and information involved in each policy. However, it is not the intent of the enterprise viewpoint to be prescriptive about the information.

6.2.6 Activities

Within a community a set of activities can be defined which include descriptions of the activities and an activity name. Associated with each activity, an **action graph** can be defined which includes a list of atomic actions associated with the activity. Also, associated with each activity is a set of **activity policies** which, along with the action graph, assure that dependencies such as the ordering of actions within an activity is clearly stipulated as a policy.

According to the RM-ODP definition, activities are generally part of a service realization and the only exchange between a client and a provider that is part of the contract specification is the heading action of a provider activity. In fact, it is on a service realization basis to fix whether the heading action is followed or not followed by subsequent actions. However, for some cases, the client has to address several related actions to the provider as part of a service feature; in this case, these related actions are part of an activity.

RM-ODP Part 2 [2] provides the following definition:

"Activity: A single-headed acyclic graph of actions, where occurrence of each action in the graph is made possible by the occurrence of all immediately preceding actions (i.e. by all adjacent actions that are closer to the head)."

6.2.7 Services

A service is a negotiation of a particular set of service features between the service provider and the service user, depending on the requirements of the user and the abilities of the provider. A service is NOT service management in the TMN sense. A service is a set of operations to accomplish a particular objective and may exist at any level of the logical layered architecture.

6.2.8 Service feature

A service feature is a set of community policies, activities and/or actions and their associated policies.

6.2.9 Composition of services

The definition of contracts based on existing contracts is for further study.

6.3 Extension of communities

The method of extending communities is for further study.

6.4 Domains

The RM-ODP concept of domains, as defined in 10.3 of RM-ODP Part 2 [2], is related to the establishment of communities. It is intended that domains be a part of the methodology; however, the concept needs to be refined in terms of the network level model. Therefore, the application of this concept in this methodology is for further study.

7 Information viewpoint

The information viewpoint is defined in RM-ODP Part 3 [3], it defines the information types within a distributed system. From the point of view of the network level modelling work the information viewpoint reflects the information aspects (including states and significant transitions) of the managed resources and management applications.

The information viewpoint defines information object types, the relationships between these object types their attributes and states along with their permitted state transitions. From the point of view of the network level model, the information viewpoint defines static and invariant information sets (schema) related to the managed resources (e.g. connections, links, network termination points, etc.) represented by the model. It also defines the dynamic schema for the resources. The dynamic schema define the allowable state changes of one or more information objects.

The information viewpoint is specified in three parts:

– an informal description which is specified in natural language with appropriate label keywords (e.g. DEFINITION, INVARIANTS, etc.);

– a semi-formal description which is specified using a subset of the guidelines for the definition of managed objects (GDMO) [5] and the general relationship model (GRM) [6] as used in OSI management; and

– formal definitions provided using the Z notation [7].

In addition, in the common information viewpoint the potential relationships are listed to enhance readability.

A complete description of the information viewpoint templates is provided in Annex B.

The method of expression in the information viewpoint does not constrain the access provided in the computational viewpoint or the representation in the engineering viewpoint

7.1 Information viewpoint

Recommendation G.853.1, Common Elements of the Information Viewpoint for the Management of a Transport Network, or common information viewpoint contains the definition of the information objects and relationships that represent the G.805 resources, independent of any particular management service. Common information attributes and states are also specified.

The common information viewpoint provides the basis for the development of management specific application information viewpoints.

When requirements are identified for a specific management application (e.g. Connection management) they are defined in an enterprise community, the corresponding management application specific information viewpoint is then developed. Recommendation G.853.1 provides the base from which such a management application specific viewpoint is developed.

Management application specific information objects may be created by subclassing from the objects in the common information viewpoint, and extending them for that application. In this case the new management application specific subclass may include other attributes from the common information viewpoint, in addition to those defined in its superclass. Additional relationships and attributes may also be created as needed for that management application. New objects, inherited from networkInfomationTop, can also be added.

If attribute definitions are compatible with attributes from existing GDMO managed object models (e.g. in Recommendation G.774 [11]), then reference to these attributes shall be informally provided. In this case the information viewpoint specification imports the semantics of the attribute but not its syntax (which can be imported into the corresponding computational viewpoint).

Modified General Relationship Model (GRM) templates have been included in this Recommendation to indicate how objects relate to each other. Each GRM template identifies roles in the relationship, and identifies information objects that may play each of these roles. In the common information viewpoint specification, the initially defined relationships that an information object may take part in are listed in the potential relationships part of the object description. When a common information viewpoint object is subclassed for a management application-specific information viewpoint, the relationships that are considered to be required for that application are declared to be mandatory.

Recommendation G.853.1 also contains common attributes that may be included when the management application specific subclasses are created; examples of these attributes include operationalState and userLabel.

8 Computational viewpoint

The computational viewpoint is defined in RM-ODP Part 3 [3], it defines the functional decomposition of the system into objects which interact with each other at specified interfaces in order to facilitate distribution. In the computational viewpoint, applications consist of configurations of interacting objects. The interfaces defined by the computational viewpoint define the maximum level of object distribution that may be supported. The final decision on the actual level of distribution supported in an "open" system is defined in the engineering viewpoint. In addition, the computational specification of object interactions is specified in terms of the detailed interfaces provided by each object, their operational signatures and their behaviour specifications. These interface specifications refer, through the utilization of parameter matching, to the state transitions of the information objects.

A complete computational viewpoint design requires the definition of computational objects as well as interfaces and operations. The definition of objects (as opposed to only interfaces) allows for the interaction between interfaces to be defined as well as the explicit definition of the number and types (client/server) of interfaces to be supported by a single object. This provides for a complete definition of the application related to the object. The definition of a single object with multiple interfaces precludes the need to deal with the replication of the state of a specific resource or other managed entity in multiple objects. If an object's interfaces need to be distributed then that object may be decomposed into multiple computational objects.

The computational viewpoint specifies computational objects, interfaces and operations which are defined as follows:

- **Computational objects** are defined as a specific view of information defined in information objects for a specific application purpose. A computational object specifies the server interfaces and client interfaces associated with the object and the behaviour (i.e. constraints between the interfaces) of the object. The behaviour clause describes the relationship between the object's interfaces. It may also describe the role of this object in providing service via each of its interfaces as well as any specific initial states within the object.
- **Interfaces** define a set of operations which may be invoked at the interface and the behaviour of the interface. The behaviour clause describes the service provided by the interface. It may also specify the ordering (or sequencing) constraints on the operations defined by the interface.
- **Operations** are defined as operational signatures. The operational signature includes input and output parameters, pre- and post-conditions, raised exceptions and operation behaviour which defines the semantics of the operation. Operations may reference data which has been defined in information objects.

8.1 Computational concepts

- Computational objects interact via interfaces.
- A given computational object may have multiple interfaces and multiple instances of the same interface type.
- Changes in the state of a system occurring as the result of operations at one interface may be viewed at other interfaces of the same object or interfaces to other objects.
- Computational objects are only defined for an application if the required interactions between the interfaces need to be standardized.

- A computational interface may allow operations which provide for the observation and manipulation of information about the resources being managed. This information may have been defined in the information viewpoint and is referenced by parameters of the operation. The data types involved are the information objects defined in the information viewpoint. These data types are grouped as decided by the particular applications, instantiated and made accessible through operation interfaces in the computational viewpoint.

8.1.1 Communications domain-independent computational viewpoint

From the point of view of this methodology the computational viewpoint can be divided into two parts: a part which is independent of the underlying engineering environment or communications domain, and that part which is closely linked to the communications domain used in the engineering viewpoint, as shown in Figure 2. The method of specifying the communications-independent computational viewpoint is defined in Annex C. This Recommendation is as far as possible independent of the communications domain (e.g. CMIP/OSI, CORBA, etc.) chosen for the underlying engineering viewpoint realization.

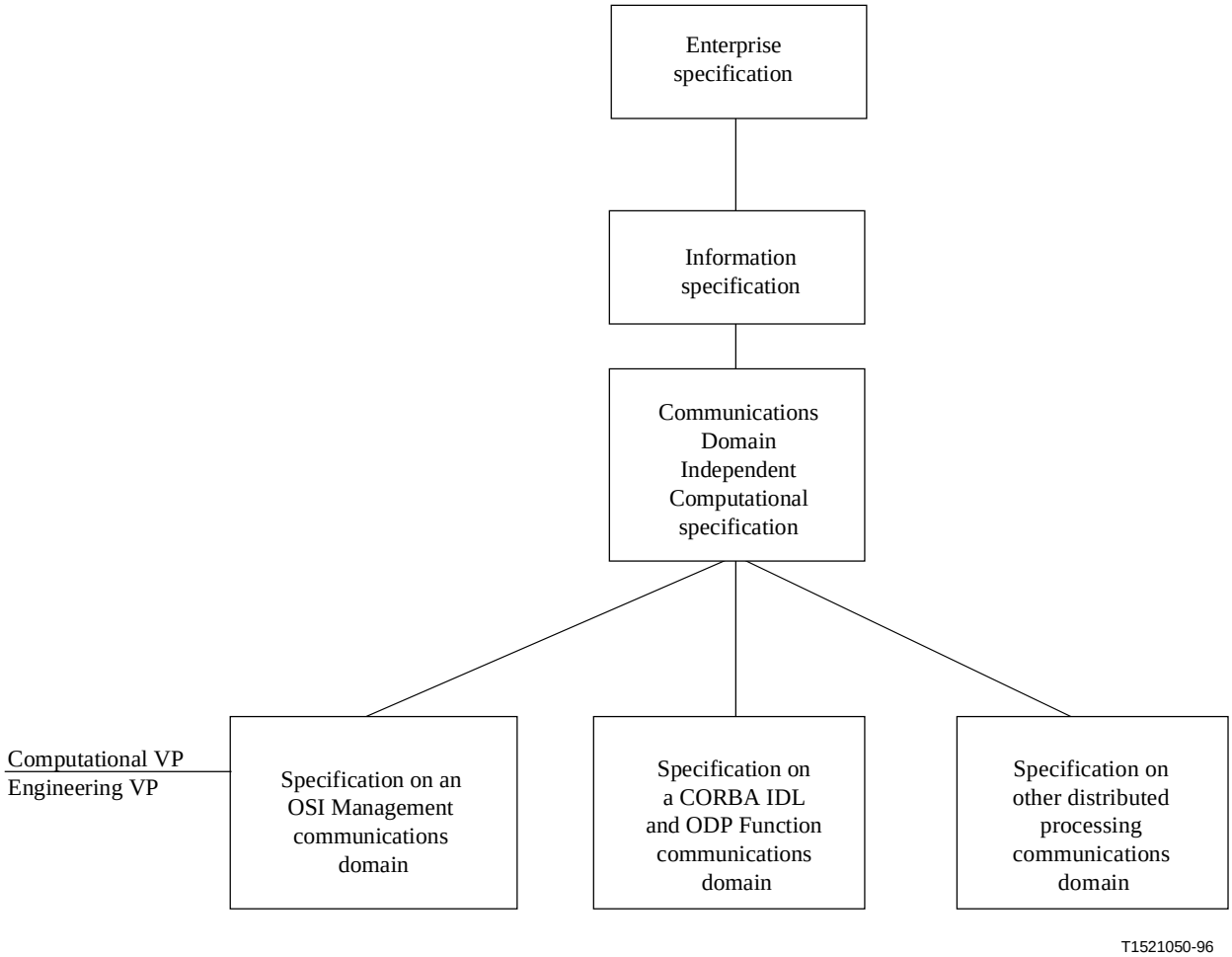


Figure 2/G.851.1 – Alternative communications domains specifications

8.1.2 Mappings to the communications domain-dependent computational viewpoint specification

The templates defined in Annex C must be mapped to the templates for the communication domain used in the engineering realization. A subset of ASN.1 abstract syntax is used to define the syntax of

the computational operation parameters. This abstract syntax representation and the computational operations must be mapped to the particular syntax and protocol for the communication domain used in the engineering realization. This process results in a computational specification that is specific for each intended communication domain.

9 Engineering viewpoint

9.1 Introduction

In RM-ODP, an engineering specification defines the infrastructure to support the functional distribution of an ODP system. This is based on the assumption that distribution transparency is achieved by assembling ODP transparency functions and objects. This methodology initially specifies engineering processes to achieve the implementation of the functional distribution for two communications domains. They include: use of OSI management, and use of CORBA IDL and ODP functions. In the future other communications domains (infrastructures) may be considered.

These alternatives are illustrated in Figure 2.

9.2 Engineering concepts

The RM-ODP engineering viewpoint provides the mechanisms by which the transparencies of the computational viewpoint are supported. Where the computational specification is protocol and infrastructure independent, the engineering viewpoint is infrastructure specific and satisfies scenarios where interfaces have been selected and transparency demands are met.

If, during the definition of the engineering viewpoint, additional requirements are discovered which affect the other viewpoints, the enterprise, information and computational viewpoints have to be updated accordingly.

The engineering viewpoint must state what scenarios are to be supported. A scenario first reflects the enterprise requirements which are to be met, and, the considerations which result in interface choices. The decision to combine computational objects may be a result of environment constraints such as real-time requirements (e.g. a maximum time period for an interaction between objects).

The ensemble technique discussed in clause 10 can be used as the basis to specify engineering objects which meet enterprise requirements and environmental constraints of a given scenario. The ensemble technique is appropriate for defining a scenario which meets enterprise requirements.

The run-time versions (Basic Engineering Object – BEO) of computational objects may be grouped into clusters which satisfy the ensemble requirements. A cluster can be viewed as a basic building block which migrates, recovers and initializes, and is replicated, as a single entity. An ODP cluster executes on one physical system. A cluster offers both client and server interfaces, which are supported by protocol specific stubs and binders. While a BEO is not necessarily equivalent to a managed object, a mapping does exist.

Relationship mapping templates: The complete specification of a relationship is only given when the GRM relationship mapping templates are defined. Because these bindings are infrastructure specific, they are found in the engineering viewpoint.

9.3 OSI management based engineering viewpoint

This subclause considers only OSI management based implementations and clusters. The OSI management information model defines a manager-agent relationship between the manager and the agent located in two different OSI systems. The computational specification makes no assumption on the physical distribution of objects; however, this engineering specification must be considered in the

TMN context. The implication of this is that the defined set of computational objects must be partitioned in two parts, each part being located on its own OSI system. In this case, the computational objects can be grouped into two clusters with only those computational interfaces which need to be exposed made externally visible to the associated cluster as OSI systems management interfaces, as shown in Figure 3.

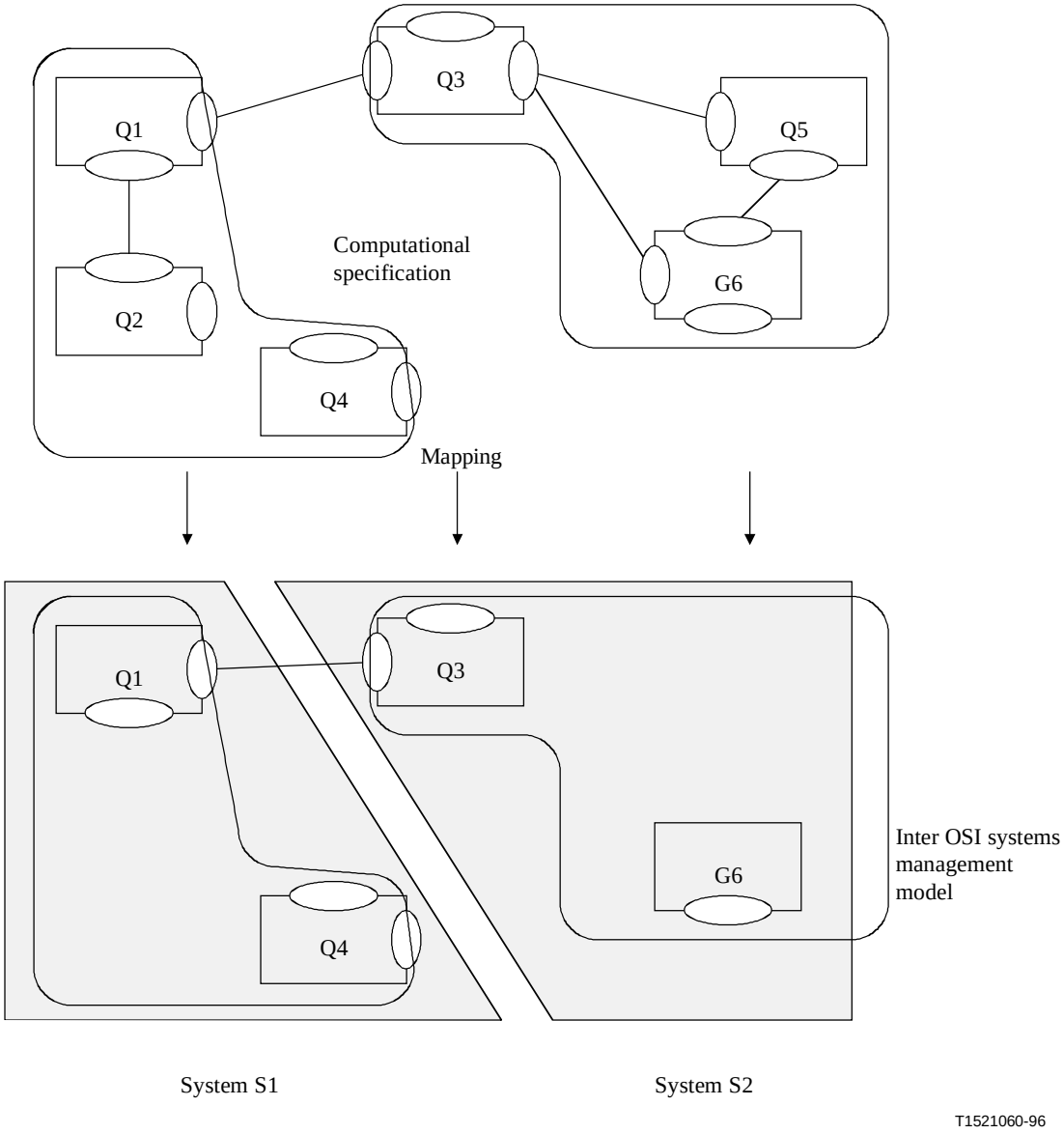


Figure 3/G.851.1 – Grouping of computational objects to define OSI management system interfaces

9.3.1 Impact of protocols on information and computational viewpoints

The engineering and computational models must be as independent as possible. However, the computational level will be influenced by engineering constraints (e.g. use of SMI). As a consequence, the information viewpoint may also need to take into account the use of SMI (e.g. with the production of GRM relationships mapping to pointers in managed objects). This will allow a mapping from computational operations to CMISE actions or attribute operations. This information expansion is part of the engineering process and leads to the production of an information model at engineering interfaces developed from the knowledge of:

– the specification of independent computational interfaces;
– distribution requirements;
– communication capabilities (e.g. CMISE).

9.3.2 Enterprise viewpoint engineering constraints

Distribution of management services across several systems may be expressed as an enterprise concern which imposes engineering constraints (e.g. definition of a management interface to a standardized network element that constitutes an abstraction of distributed resources). For an enterprise specification of engineering constraints, different scenarios may be described, resulting in different profiles and ensembles.

9.3.3 The use of GDMO packages and managed objects

Prescriptions are required to choose between ways to express information at the engineering level using GDMO.

Resources (transport or managed resources) can be modelled as managed objects, since they represent entities that may be manipulated by all management services.

There are two ways to model management services:

1) A given management service may be modelled as a managed object and the composition with the associated managed resource can be made by inheritance in a managed object deriving from both the managed resource and the management service object classes; this solution leads to a great number of managed object classes. In addition, it is different from the composition of interfaces defined in RM-ODP (particularly in the case of dynamic binding). Use of combined pointers between the managed resource and the management service is consistent with the notion of composition of interfaces; however, this solution results in a large number of pointers.

2) A given management service may be modelled as a package and the composition with the associated managed resource can be made by inclusion of these packages as conditional packages in the managed object class reflecting the resource. This solution is consistent with the use of a pre-defined engineering library. This library will contain managed objects representing transport resources (that is, corresponding to common information viewpoint information objects), and for each one, a package per potential supported service. This will avoid rewriting and provide a more homogeneous view of a system.

Note that multiple instances of the same type may be defined in the computational viewpoint. The use of GDMO packages in this case is not appropriate.

9.3.4 Support of multiple management services at an engineering interface

Packages may be fully independent of each other and in this case they can be independently included in the managed object when specifying the corresponding services at the engineering interface. However, some packages may express dependencies. In this case the best way is to define a compound package, but in this case the corresponding compound service must be defined (with the dependencies expressed in enterprise, information and computational viewpoints). In addition, some packages may have to be mutually exclusive in a managed object. This must be expressed in the managed object behaviour statement.

9.3.5 Identification

In accordance with the RM-ODP, computational interfaces must be identified to be referenced (for example for binding). Computational interfaces may be mapped onto packages. The desired packages will be instantiated at the same time as the managed objects that support them. A unique

engineering reference will be made to the managed object. This engineering reference may be independent of any information concern, and then produced on a policy basis only known by the engineering system that supports it; or an attribute will be defined which carries the identification: instanceId.

9.3.6 Relationship mapping

GRM relationships will map to ROLE BINDING templates in the engineering viewpoint. The representation may be in terms of pointers, name bindings, operations, or inheritance, for example.

9.3.7 The use of actions versus attribute operations for relationships and state modifications

CMISE allows two ways to modify the state of a system:

- REPLACE operations on attributes;
- actions.

Actions will be used when multiple attributes are involved and the operation is considered to be atomic or if significant behaviour is required. If only one attribute is modified and there is a need to provide read access to the attribute, then a REPLACE operation may be prescribed.

9.4 The distributed processing environment

9.4.1 Overview

The use of a Distributed Processing Environment (DPE) is for further study. Some considerations are:

– Since many implementation choices exist (e.g. OSI, CORBA, DCE based DPEs), the set of engineering models must use a common DPE which allows these domains to interact.

– Local forms of naming, protocols and reliability schemes may be used within each domain; however, a set of common support services are required for inter-working. For example, only one trader should be endorsed by Study Group 15.

– Basic engineering objects may be supported by stub and protocol objects, to facilitate inter-working of selected objects by various protocols.

There is also a need to define transparency requirements for the DPE which is used by transport network management applications.

10 The use of ensembles in the definition of network management applications

10.1 Scope

The methodology in this Recommendation produces application specific specifications for the enterprise, information and computational viewpoints which are, in the main, distribution infrastructure dependent. The ensemble technique is used to allow definition of an engineering solution that satisfies a specific application scenario. This is illustrated in Figure 4.

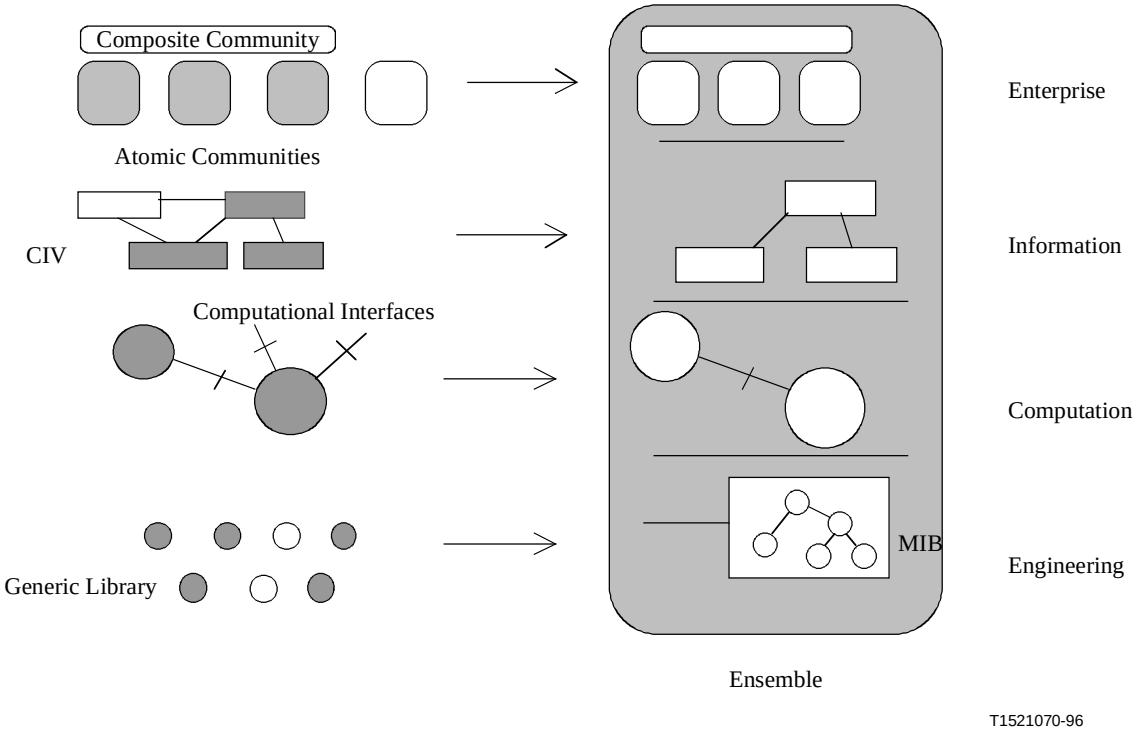


Figure 4/G.851.1 – Use of an ensemble to produce an application specific interface

The ensemble identifies the specific set of management applications to be supported (in terms of the enterprise communities) together with other implementation constraints such as the degree of distribution to be provided, the level of visibility and the interface protocol.

The ensemble is not part of any RM-ODP viewpoint, but references the enterprise, information, computational, and engineering viewpoints to allow the development of a solution that satisfies the requirements of a set of enterprise communities. The ensemble is a guide for using the material in the RM-ODP viewpoints to solve a particular management problem. This means that the ensemble provides a particular solution in terms of an engineering interface specification dictating a particular distribution of management functionality. Conformance will be tested at the engineering viewpoint.

The ensemble template is defined in Annex F.

ANNEX A

Template and guidelines for the enterprise viewpoint specification

This Annex provides an explanation of an enterprise viewpoint specification.

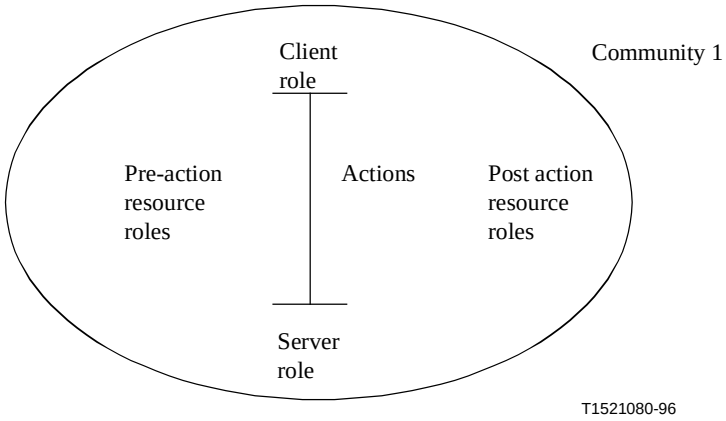


Figure A.1/G.851.1 – Members of a community have roles in regard to actions

A.1 Informal definition of the enterprise template

X COMMUNITY <community_label> "Name"

X.1 PURPOSE

community_definition -- *This clause introduces the purpose of the community.*

X.2 ROLE

<community label><Role Label>

role_definition -- *The list of member roles that are involved in the community are provided with a brief description.*

X.3 POLICY

PERMISSION <label_string>|OBLIGATION <label_string>|PROHIBITION <label_string>|EXCEPTION <label_string>

policy_definition -- *Policies applicable to the entire community are provided.*

X.4 ACTION

X.4.1 <action label> "Action Name"

action_definition -- *This provides the list of actions which support the community purpose.*

ACTION POLICY: -- *A short clear concise description of the policy as either an OBLIGATION, PERMISSION, PROHIBITION or EXCEPTION is provided here.*

PERMISSION <label_string>|OBLIGATION <label_string>|PROHIBITION <label_string>|EXCEPTION <label_string>

policy_definition -- *These policies should each state the role and information involved. It is the intent of the enterprise viewpoint not to be prescriptive about the information.*

X.5 ACTIVITY

Each activity is defined in a subsection titled by the activity name and introduced by a definition of the activity.

Activity Name <activity label>
activity_definition

WITH ACTION GRAPH -- *A list of atomic actions*.

Start
<action label>
<nth action label>
End

ACTIVITY POLICY -- *The action graph needs to assure that dependencies such as the ordering of actions is clearly stipulated as a policy*.

PERMISSION <label_string>|OBLIGATION <label_string>|PROHIBITION <label_string>|EXCEPTION <label_string>

policy_definition

X.6 CONTRACT

A service is a negotiation of a particular set of service features between the service provider and the service user, depending on the requirements of the user and the abilities of the provider. The result of the negotiation is a contract which reflects the agreed selection from the feature set of the supplier. This contract establishment phase is for further study; however, these service features can be captured as a set of supported actions and, where applicable, supported policies.

A.2 Formal definition of the enterprise template

This subclause provides a formal description of the enterprise description in BNF.

NOTE – Extensions to BNF for this specification.

[...] – optional item, 0 or 1 occurrences

* list item 1 or more occurrences

-- *remainder of line is a comment as per ASN.1*

<community_template> ::=
"COMMUNITY" <label> <name>

<doc_heading> "PURPOSE" <community_definition>

<doc_heading> "ROLE"
<role_definition>*

<doc_heading> "POLICY"
<policy_definition>*

<doc_heading> "ACTION"
<action_description>*

<doc_heading> "ACTIVITY"
<activity_definitions>*

["WITH ACTION GRAPH"
"Start"
<action_label>*
"End"]

"CONTRACTS" <label_string>

<action_description> ::= <doc_heading> <label> <label_string>
"ACTION POLICY" ":"
<policy_definitions>

<action_label> ::= <label>

<activity_definition> ::= <doc_heading> <label>
<label_string>
"ACTIVITY POLICY"
<policy_definitions>

<activity_definitions> ::= "None"
| <activity_definition>*

<community_definition> ::= <label_string>

<doc_heading> ::= <text>
-- doc_heading represents a heading when the template
-- is printed. It is publishing system specific and would
-- be considered White space in an implementation

<label> ::= "[A-Za-z][-A-Za-z0-9.]*"

<label_string> ::= -- quoted string | label

<name> ::= <label>

<policy_definitions> ::= "None"
| <policy_definition>*

<policy_definition> ::= { "PERMISSION" | "OBLIGATION" | "PROHIBITION" |
"EXCEPTION" } <label> <label_string>

<role_definition> ::= <role_label> <label_string>

<role_label> ::= <label>

ANNEX B

Information viewpoint structure

B.1 Introduction

The information viewpoint section is structured into the following subsections:

- A list of information entities which have been defined in other specifications and are referenced by the corresponding enterprise community, including other information entities from other specifications for inheritance purposes. When information entities are imported into an information specification in this way, all other related specifications relevant to this

entity are also imported (e.g. informal, semi-formal and formal specifications are imported when one of these specifications is listed).

- Diagrams of information object and relationship types as required to provide readability:
  - contains role[2]-relationship diagrams and class-role diagrams. The separation between roles and classes is based on the use of GDMO and GRM notations where relationships are defined between roles that can be played by instances of object classes;
  - inheritance diagrams;
  - relationship diagrams;
  - entity-relationship diagrams.
- Information object classes:
  - defines the object classes. The semi-formal description uses an INFORMATION OBJECT CLASS template which is similar in structure and syntax to the GDMO MANAGED OBJECT CLASS template, defined in CCITT Rec. X.722 | ISO/IEC 10165-4 (GDMO) [5]. The formal definitions are in the Z notation [7].
- Information relationships:
  - defines the relations in which the information objects may participate. Uses the RELATIONSHIP CLASS template defined in ITU-T Rec. X.725 | ISO/IEC 10165-7 (GRM) [6] for the semi-formal part and Z notation for the formal part.
- Static schema definitions:
  - defines the global and generic system states, described in schemas. This subsection has informal, semi-formal, and formal parts.
- Schema transition definitions (dynamic schemas):
  - labels particular transitions between compound states, described by static schema, so that these transitions can be referenced as triggers of notifications. This subsection has informal, semi-formal, and formal parts.
- Attribute type definitions:
  - definition of the attributes for the concerned service. Uses the ATTRIBUTE template defined in GDMO for the semi-formal part and Z notation for the formal part.

B.2 Model descriptions

For any information definition in the information viewpoint section, three specifications are provided:

- An informal (natural language) description which is intended to be human readable. In order to be as concise as possible, this specification is structured, using labels to introduce each clause (e.g. <definition>, <invariants>, ...) and its sub-clauses (<inv_1>, <operationalState>, ...). These labels are used to provide cross-references within a particular service, across several viewpoints of the same service or between several services.
- A semi-formal description using a notation derived from GDMO for object classes and attributes. This notation, GDIO (Guideline for the Definition of Information Objects), is based on the use of the GDMO managed object class and attribute template structures. In this formal specification, all natural language (normally used in the BEHAVIOUR clause) will be replaced by reference to the equivalent content of the informal description. For the

[2] In the GRM sense.

relationship templates, this specification uses the GRM notation. Again, the BEHAVIOUR clause refers to the informal part.

- A formal description using the Z notation.

B.3 Structure of the specification

B.3.1 Information object classes

B.3.1.1 Informal description

The informal part of the object class specification is structured into four parts:

- The first one, introduced by the keyword <definition> contains a definition of the information object class.
- The second one, introduced by the keyword <attributes> lists the attributes of the information object class by inclusion of a reference, which can be local if the attribute is locally defined or global if the attribute is imported from another service.
- The third one, introduced by the keyword <invariants>, describes the permitted states for an instance of the information object class. These may be expressed in terms of individual attribute values, or the combination of attribute values if they are interdependent. Attribute value dependencies may also be described in static schema (see B.3.3). If the state of an attribute is independent of the states of the others, its permitted values can be described independently. If the specification of permitted values is missing, by default, all possible attribute values are permitted.
- The fourth part, introduced by the keyword <transitions> expresses the state transitions of the information object. This clause defines the relevant, valid transitions between all combined states of the information object. The state transitions are required to be specified in this part may be influenced by the specification of invariants as specified in the <invariants> part. For example, an invariant may restrict the visibility of need for some possible state transitions.

  If an attribute is independent of the others, i.e. its value does not influence the state transition of the other attributes, its transitions can be described independently. If this description is missing, by default, all state transitions are permitted for that attribute.

B.3.1.2 Semi-formal description

This description uses the GDMO notation with the "MANAGED OBJECT CLASS" keyword changed to "INFORMATION OBJECT CLASS" and the following restrictions:

- no access-specifier can be assigned to attributes;
- the REGISTERED AS clause is not required;
- no ACTIONS or NOTIFICATIONS can be specified;
- the WITH ATTRIBUTE SYNTAX clause shall not be used in the ATTRIBUTE template;
- the GDMO "BEHAVIOUR" construct shall be structured, following the subsections of the informal description, as follows:
  - the definition part is referenced by including the <definition> keyword;
  - the <invariants> and the <transitions> parts are copied into the BEHAVIOUR clause;
  - in application-specific subclasses the actual relationships which are used shall be listed.

The <attributes> part of the informal description is converted to an ATTRIBUTES construct of the information object template, without any *access-specifier*.

B.3.1.3 Formal description

The formal description uses the Z notation. See Appendix VII for an example of how Z can be applied to this description.

B.3.1.4 Potential relationships

This part of the information object definition is required for the common information viewpoint specification only. It defines the possible relationships in which the information object or subclasses can participate in. It is included in the common information viewpoint specification to enhance the clarity of the specification only and is not a list of relationships which the object or its subclasses must support, nor is it necessarily an exclusive list.

When the common information viewpoint object is subclassed for a management application-specific information viewpoint, the relationships that are necessary for that application and which affect the behaviour of the object are listed in the BEHAVIOUR clause of the semi-formal description of the new subclass and in the relationship part of the informal description. These relationships may be a subset of those listed for the common information viewpoint superclass as well as any additional required relationships.

B.3.1.5 Example

networkCTP

Informal description

DEFINITION

"The networkCTP information object represents an extremity of a linkConnection."

Semi-formal description

```
networkCTP INFORMATION OBJECT CLASS
DERIVED FROM networkInformationTop;
CHARACTERIZED BY
    networkCTPPacakage PACKAGE
    BEHAVIOUR
    networkCTPPackageBehaviour BEHAVIOUR
    DEFINED AS
    "<DEFINITION>";;;;
```

Formal description

networkCTP_Static

networkCTP : **F** OBJECT

networkInformationTop_Static

networkCTP ⊆ networkInformationTop

networkCTP_Dynamic

Δ networkCTP_Static

networkInformationTop_Dynamic

Potential relationships

<clientServer>
<extremitiesTerminateTransportEntity>
<networkTTPAdaptsNetworkCTP>
<subnetworkTPIsRelatedToExtremity>

B.3.2 Information relationships

B.3.2.1 Informal description

The informal part of the relationship class specification is structured into four parts:

- The first one, introduced by the keyword DEFINITION contains a definition of the relationship class.
- The second one, introduced by the keyword ROLE describes the roles of the relationship class.
- The third one, introduced by the keyword INVARIANT gives the invariants applicable to the information objects playing the roles of the relationship during the lifetime of this relationship. These invariants can be expressed using an informal language, e.g. inv_2: "The container and the elements must have the same directionality."
- The last part, introduced by the keyword TRANSITION, expresses the restriction on the state transitions of the information object playing the roles of the relationship.

B.3.2.2 Semi-formal description

The semi-formal part of the information relationship specification is written using extended GRM. The extension to GRM is in the "COMPATIBLE WITH" clause which has been modified to accept more than one class label[3]. The restriction of referencing at most one object type in the clause "COMPATIBLE WITH" leads to duplicate the relationships. This extension of the GRM notation helps to provide a document which is easier to read.

The GRM "BEHAVIOUR" construct shall be structured, following the parts of the informal description, as follows:

- the definition part is referenced by including the keyword DEFINITION;
- the INVARIANT part is copied into the BEHAVIOUR construct, except for those invariants concerning cardinalities which are translated into the "PERMITTED-ROLE-CARDINALITY-CONSTRAINT" construct;
- the TRANSITION part is copied into the BEHAVIOUR construct, except for the invariants concerning dynamic evolution of a relationship regarding the departure and arrival of object instances which have been translated using the construct BIND-SUPPORT and UNBIND-SUPPORT.

In addition, the ROLE part of the informal specification is translated into the ROLE construct, including the COMPATIBLE WITH construct.

B.3.2.3 Formal description

The formal description uses the Z notation. See Appendix VII for an example of how Z can be applied to this description.

B.3.2.4 Example

linkHasLinkConnections

---

[3] It is possible to define a managed object class derived from a common supertype of the list of object classes, with its behaviour class stating that its instances are compatible with the object classes in the list. This can be used in the actual GRM in the engineering viewpoint.

Informal description

DEFINITION

"The linkHasLinkConnections relationship class describes the relationship that exists between a link and the linkConnections that are part of it.
This relationship type is a subtype of setOf."

ROLE

container

"Played by an instance of the link information object type or subtype."

element

"Played by an instance of the linkConnection information object type or subtype."

Semi-formal description

linkHasLinkConnections RELATIONSHIP CLASS
DERIVED FROM setOf;

BEHAVIOUR
linkHasLinkConnectionsBehaviour BEHAVIOUR
DEFINED AS
"<DEFINITION>";;

ROLE container

COMPATIBLE WITH link AND SUBCLASSES;

ROLE element

COMPATIBLE WITH linkConnection AND SUBCLASSES;

Formal description

_____ linkHasLinkConnections_Static _____

linkHasLinkConnections : **F** RELATIONSHIP
setOf_Static
link_Static
linkConnection_Static

---

linkHasLinkConnections $\subseteq$ setOf

$\forall$ R : linkHasLinkConnections $\bullet$container(R) $\in$ link $\wedge$ elementSet(R) $\in$ linkConnection

---

_____ linkHasLinkConnections_Dynamic _____

Δ linkHasLinkConnections_Static
setOf_Dynamic
link_Dynamic
linkConnection_Dynamic

B.3.3 Static schema definition

The static schema is used to define a set of states for a system at an instant in time. It defines global and generic states for a system, i.e. states that involve one or more attributes that are in one or more information objects. This is needed to describe the system state before a global system transition (pre-conditions of a computational operation) and the system state after this transition (post-conditions)[4].

---

[4] The transitions between these states will be described in the computational viewpoint, although they are part of the information viewpoint.

A system state is expressed with the following rules:

- attribute value constraints (within an object or between objects through the relationship constraints);
- object existence[5].

It is only necessary to specify those static schema which are of interest.

B.3.3.1 Informal description

The informal description of a state definition schema is structured in three parts:

- The first, introduced by the keyword DEFINITION, gives the semantic of the global state.
- The second, introduced by the keyword ROLE, lists all the roles onto which a constraint or invariant is defined.
- The third, introduced by the keyword INVARIANT, lists the rules that define the state.

B.3.3.2 Semi-formal description

The definition of the specification method for the semi-formal description is for further study. The use of the RELATIONSHIP CLASS from the GRM has been proposed as one method of specification.

B.3.3.3 Formal description

The definition of the specification method for the formal description is for further study. A formal description in Z has been proposed.

B.3.3.4 Example

ssccNotConnected

This information concept is related to the following enterprise entities:

<"Rec. G.852.1",COMMUNITY:sscc,ACTION:sscc1,OBLIGATION:OBLG_2>,
<"Rec. G.852.1",COMMUNITY:sscc,ACTION:sscc1,OBLIGATION:PROH_1>,
<"Rec. G.852.1",COMMUNITY:sscc,ACTION:sscc2,OBLIGATION:OBLG_2>.

Informal description

DEFINITION

"The ssccNotConnected schema defines a schema type with two non-connected subnetworkTPinformation objects subtypes candidates to the point-to-point connection management service."

ROLE

involvedSubnetwork

"Played by an instance of the ssccSubnetwork information object type or subtype."

potentialAEnd

"Played by an instance of the ssccSubnetworkTPSink, ssccSubnetworkTPSource or ssccSubnetworkTTPBidirectional object types or subtypes."

potentialZEnd

"Played by an instance of the ssccSubnetworkTPSink, ssccSubnetworkTPSource or ssccSubnetworkTTPBidirectional object types or subtypes."

INVARIANT

---

[5] Note that this clause may be expressed with the previous rule taken into account that an object exists if and only if its object_id attribute value is different from null.

inv_1

"The objects playing the potentialAEnd and potentialZEnd roles are involved in an instance of the subnetworkIsDelimitedBy relationship type with the object playing the role involvedSubnetwork."

inv_2

"The object playing the potentialAEnd role is not involved in any instance of the subnetworkConnectionIsTerminatedByPointToPoint relationship type and subtypes."

inv_3

"The object playing the potentialZEnd role is not involved in any instance of the subnetworkConnectionIsTerminatedByPointToPoint relationship type and subtypes."

B.3.4 Dynamic schemas

A dynamic schema is used to express the valid transitions among two or more static schema.

B.3.4.1 Informal description

A set of static schema definitions can be provided as pre-conditions to a specification of a transition to a set of static schema as post-conditions. The informal specification will have the following format:

Label DEFINITION "text"

PRE_CONDITION <static_schema_label> or "text string"

POST_CONDITION <static_schema_label> or "text string"

Where "text string" shall only be used to express constraints on attribute values of a single object.

B.3.4.2 Semi-formal description

For further study.

B.3.4.3 Formal description

For further study.

B.3.5 Attributes

B.3.5.1 Informal description

The informal part of the attribute definition is structured in four parts:

- The first one, introduced by the keyword DEFINITION, contains a definition of the attribute.
- The second one, introduced by the keyword STATE, lists all the values that an attribute can have, with associated semantics.
- The third one, introduced by the keyword INVARIANT, lists the invariants valid for that attribute, if any.
- The fourth one, introduced by the keyword TRANSITION, lists all the possible transitions between the states of the attribute. This can be done by defining a transition table.

Attribute definitions only provide the semantics of their values, not their syntax. If attribute definitions are compatible with attributes from existing GDMO Managed Object models (e.g. in Recommendation G.774) then reference to these attributes should be provided. In this case the information viewpoint specification imports the semantics of the attribute but not its syntax (which can be imported into the corresponding computational viewpoint).

B.3.5.2 Semi-formal description

The semi-formal description uses the ATTRIBUTE template from GDMO excluding the WITH ATTRIBUTE SYNTAX clause.

B.3.5.3 Formal description

The formal description uses the Z notation. See Appendix VII for an example of how Z can be applied to this description.

B.3.5.4 Example

userLabel

Informal description

DEFINITION

"The userLabel attribute type assigns a user friendly name to the associated resource. The semantic of this attribute is imported from M:3100: 1994 userLabel attribute."

Semi-formal description

userLabel ATTRIBUTE
BEHAVIOUR
DEFINED AS
"DEFINITION";

Formal description

[UserLabel]

_____ userLabel_Static _____

userLabel: OBJECT **F** UserLabel

_____ userLabel_Dynamic _____

Δ*userLabel _Static*

∀ object : OBJECT | object ∈ **dom** userLabel ∪ **dom** userLabel' •
userLabel'(object) =userLabel(object)

ANNEX C

Computational viewpoint template description

C.1 Introduction

The templates defined in this Annex must be mapped to the templates for the communication domain used in the engineering realization. A subset of ASN.1 abstract syntax is used to define the syntax of the computational operation parameters. This abstract syntax representation and the computational operations must be mapped to the particular syntax and protocol for the communication domain used in the engineering realization. This process results in a computational specification that is specific for each intended communication domain.

An interface may have many operations. Each operation is defined using the operation templates.

The object template is for the computational object class. Each instance of the class may have multiple instances of any given interface.

ASN.1 is used as the data definition language. The following restrictions are recommended for ease of translation to IDL/ODL type definitions:

– use SEQUENCE OF instead of SET OF

– avoid use of ANY or ANY DEFINED BY

– limit use of recursive type definitions to:

  • name1 ::= SEQUENCE {
    .
    .
    .
    SEQUENCE OF name1}

  • where name1 = name1

NOTE – This construct supports the CMIS_Filter.

C.2 Guidelines

This subclause provides guidelines on how to complete the templates which are formally defined in C.3. It provides information on the mapping to information entities in the information viewpoint.

In the description of an operation in the computational viewpoint, pre- and post-conditions on the invocation of the operation can be specified as invariants directly in the behaviour clause of the operation template or they can be specified by references to the schemas defined in the information viewpoint.

These schemas can be referenced to define the pre-conditions that the system must verify so that this operation can be performed, and the post-conditions that the system must verify after this operation. The static schema specifies a set of systems by providing:

- objects involved in all the instances of the systems class (through the use of the <role> clause in the information_relationship clause in the information viewpoint);
- constraints on their states (through the use of <invariant> clause);
- relationships involved in the systems class (through the use of <invariant> clause);
- constraints on combined states described in relationships (through the use of <invariant> clause).

The operation parameter will select which system instance will be addressed by the operation in the pre-condition and which system instance will be provided as the result of the operation in the post-condition. Therefore the behaviour may include:

- a pre-condition clause which may reference a template in the information viewpoint specification;
- a post-condition clause which may reference a template in the information viewpoint specification;
- a parameter matching rule clause or text to give information about this selection.

A single instance of a computational interface type cannot support both client and server interfaces. Therefore, it likewise cannot support both PRE/POST-CONDITIONS and the TRIGGERING CONDITION.

C.2.1 Guidelines specific to the operation template

This subclause provides guidelines on the content of the entities in the OPERATION template.

C.2.1.1 Parameter template

A computational operation parameter is associated with an information object. It has a type associated with it. A parameter cannot be declared without an associated parameter matching clause.

C.2.1.1.1 Operation parameters used to identify objects

In the computational viewpoint, interfaces are defined to allow access to the (computational viewpoint) abstraction of a resource that is identified in an enterprise viewpoint. Each computational interface provides access for a different management purpose. Within the computational viewpoint, the enterprise resource that is being addressed may be identified by referencing any one of its interfaces. The only restriction is that the computational interface used to identify a resource must always be present on the resource. For example, a port in the sscc enterprise community can be referenced by the use of reference to its associated ssccSnTpIfce in the corresponding computational viewpoint.

Computational interface types, that are communication domain independent, are mapped to communication domain dependent interface types (or "objects") in the engineering viewpoint. For example, when mapping to the OSI Management communication domain, each interface may be mapped onto a managed object class or package in an object class.

RM-ODP Part 3 states the following: "A formal parameter that is an identifier for a computational interface is qualified by a computational interface signature type. The corresponding actual parameter must reference an interface with that interface signature type (or one of its subtypes). The actual parameter can only be used as if it referenced a computational interface with the same signature type as the formal parameter (or one of the formal parameter's supertypes)."

Therefore, when passing a parameter in an operation, the type of that parameter must be the same as the type of the computational interface referenced in the definition of the operation. In the communication domain independent templates, a typed interface reference is used to indicate an interface reference parameter in computational operation signatures. The notation:

"<param_label> : <type> ::= (<interface_type_name>)"

is used to specify a parameter that is a reference to an instance of an interface conforming to the interface type named <interface_type_name>.

For example in the following:

INPUT_PARAMETERS
snpa : SnTPID ::= (ssccSnTpIfce);

where:

snpa -- *defines the input parameter name as snpa*
: SnTPID -- *defines the parameter type name to be SnTPID*
::= (ssccSnTpIfce) -- *defines an SnTPId to be of type ssccSnTpIfce, which may be satisfied by a ssccSnTpIfce interface or one of its subtypes*.

When a reference to an interface is used as a parameter in a computational operation, the signature of that computational operation must indicate the type of interface that is passed as the actual parameter at run time. Therefore, the actual parameter being passed must contain the identity of an object instance that represents the defined interface type (or one of its subtypes), in the run time invocation, in an engineering realisation of the operation.

At specification time, a reference to an interface as a parameter must, by convention, unambiguously define the type for the (formal) parameter in question.

NOTE – The value of such an interface reference parameter may be subsequently used to bind to the associated interface instance to invoke operations to manipulate or query the state of a resource.

The mapping of the identifier for an interface, in specific communication domains, can be done in a manner most appropriate for the mapping of each interface type to the underlying type associated

with the engineering access mechanism (e.g. Managed Object instance name, or CORBA Object Reference).

C.2.1.1.2 Computational parameter types

The BNF definition of the operation template in C.3 defines the syntax specification for a computational parameter as either a single ASN.1 type specification, or an interface reference specification.

The ASN.1 type specification uses either an ASN.1 in-line type production, or an ASN.1 type production reference.

An interface reference specification uses the keyword "REF" [e.g. REF (<commonNTPIfce>)].

The interface type at the highest level of the interface hierarchy is generally used as the named interface type in the REF specification, and it implies a reference to an interface instance of that type or any of its compatible subtypes derived from inheritance. The interface name inside the REF specification could be replaced, at run time, with any interface instance of a type derived from the named interface in the reference.

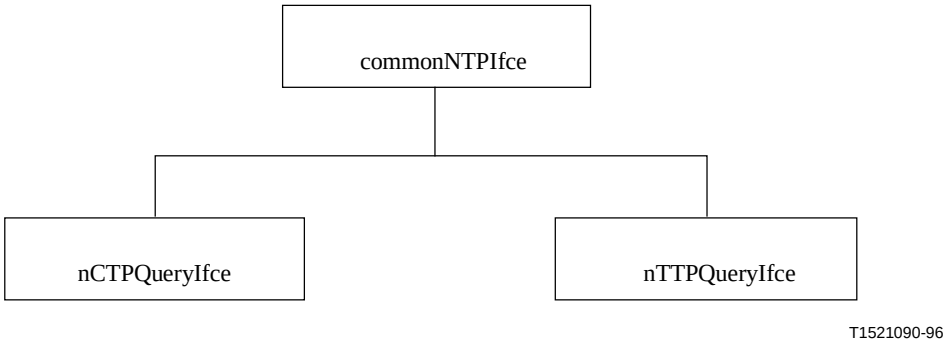


Figure C.1/G.851.1 – Example interface hierarchy

For example, the simplified inheritance hierarchy in Figure C.1 shows an interface type "commonNTPIfce", which is a common supertype of two derived interface types, "nCTPQueryIfce", and "nTTPQueryIfce". The operations of the supertype are included, through inheritance, in the derived interfaces. Thus a reference to an instance of either of the two derived interfaces can serve as a satisfactory substitute for a reference to an instance of the supertype interface. However, holding a reference to the supertype interface (commonNTPIfce) will not necessarily work as a reference to an instance of one of the derived interface types.

C.2.1.2 Parameter matching template

A parameter matching clause specifies the set of information objects or attributes that are intended to be bound to the parameter. It specifies an information object, either directly or as a ROLE played in an information relationship, or an attribute value of an information object. The parameter matching clause constrains the acceptable parameter more than the type declaration of the parameter declaration. The optional term "ELEMENTS" is used to indicate that the matching expression is applied to each element of a composite parameter.

C.2.1.3 Pre- and post- conditions

Pre- and post- conditions on the invocation of the operation are specified as invariants directly in the behaviour clause of the operation template, or they can be specified by references to a schema defined in the information viewpoint. These invariants define the conditions the system must verify

so that this operation can be performed, and the post-conditions are asserted to ensure that the operation is completed successfully.

C.2.1.4 Exceptions

Exceptions are raised against the invariants of the pre- and post-conditions, and specify what is to happen if an invariant evaluates to "false". As there is no implied execution environment, all exceptions must be specified explicitly. To raise an exception on a parameter matching failure, an invariant referencing the parameter matching rule should be provided.

C.2.2 Client/server interface definitions

Computational objects can interact through a connection established between a client interface and a server interface. A computational interface can be an operational interface or a notification interface. The roles of these interfaces relative to the interface types are:

– An operational client role: which can invoke operations on operational server interfaces.

– An operational server role: which receives operations from operational client interfaces.

– A notification client role: which invokes notifications on notification server interfaces.

– A notification server role: which receives notifications from notification client interfaces.

Operations are emitted by an operational client interface and received by an operational server interface. Notifications are emitted by a notification client interface and received by a notification server interface.

For an operational server interface the input parameters contain the request information from the operational client interface. See Figure C.2.

For a notification client interface the input parameters contain the actual content of the notification. See Figure C.3. For this reason the actual contents of the notification are specified as the input parameters of the operation which defines the notification.

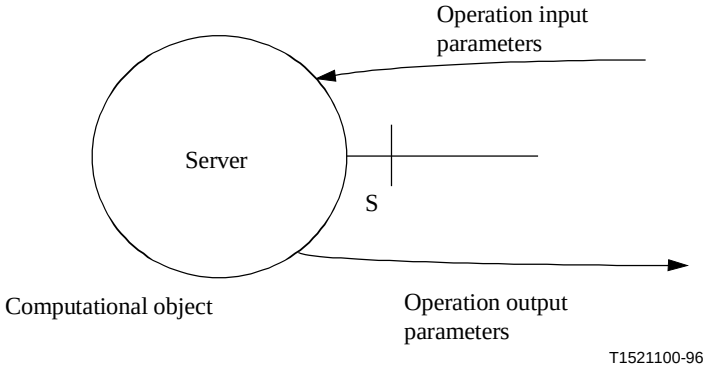


Figure C.2/G.851.1 – Server operational interface on computational objects

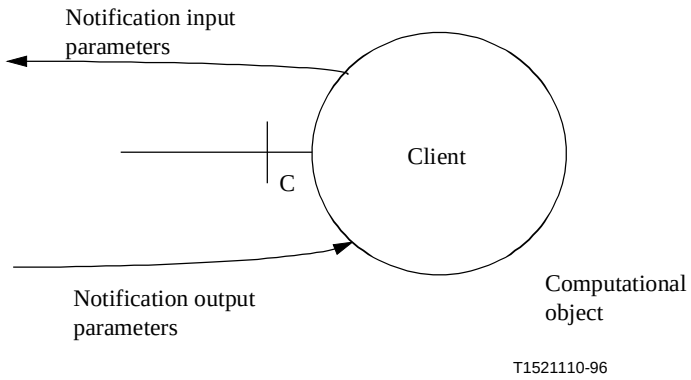


Figure C.3/G.851.1 – Client notification interface on computational objects

C.2.3 Considerations for mapping to different communications domains

The domain independent computational interfaces are not all derived from a common "top" object. Thus, in mapping the domain independent computational templates to GDMO, underived interface definitions must be replaced with a managed object definition derived from the GDMO "top" managed object class. Each REF specification can be translated to an ASN.1 type derived from "ObjectInstance", since CMIP engineering solutions use managed object names as references to managed object server interfaces.

In general, the mapping from the communication domain independent computational interface templates to communication domain specific interface definitions (e.g. CORBA IDL or GDMO/CMISE) for particular engineering realizations, an appropriate "top" object is inserted at the top of the interface hierarchy. In addition, the REF type is mapped to an appropriate type suitable for interface referencing.

The attributes of the GDMO "top" managed object class allow a client to determine the actual class which the managed object was instantiated as. In particular, a client may invoke a CMIP M-get on the managedObjectClass attribute, with the managed object instance CMIP parameter set to the reference value, and the managed object class CMIP parameter set to "Recommendation X.721"::actualClass value [12].

Likewise, CORBA IDL has a common supertype interface called "Object" which has an operations (called narrow) allowing a client to determine what derived interface types the actual instantiated reference is compatible with. Since CORBA IDL has an interface reference type, the REF specification can map directly onto the CORBA interface reference in mappings to CORBA domain specific interface templates.

C.3 Formal template definitions

This subclause defines the computational templates using BNF.

C.3.1 Object class template

<computational_object_template> ::= <computational_object_header>
"{"<computational_object_body> "}"

<computational_object_header> ::= "COMPUTATIONAL_OBJECT_CLASS
<object_name>

<object_name> ::= <identifier>

<computational_object_body> ::= **[<server_interface_definitions>]**
[<client_interface_definitions>]
[<behaviour_definition>]

<server interface_definitions> ::= **"SERVER_INTERFACES"**
{<server_interface_label> ";" }*

<server_interface label> ::= **<label_reference>**

<client interface_definitions> ::= **"CLIENT_INTERFACES"**
{<client_interface_label> ";" }*

<client_interface label> ::= **<label_reference>**

<behaviour_definition> ::= **"BEHAVIOUR" {<text_delimiter> <string_literal>**
<text_delimiter>
<string_literal> }";"

<identifier>::= [a-zA-Z][-a-zA-Z0-9_:.]*

-- This accepts identifiers as starting with a letter and containing
-- letters, digits, underscores, hyphens, colons and points.

<text_delimiter> ::= **! | " | # | $ | % | ^ | & | * | ’ | ‘ | ~ | ? | @ | **

NOTE – If a text delimiter is used, the same character shall be used at the start and end of the string, and whenever that text_delimiter character appears in the body of the text string, it shall be replaced by two occurrences of that character. If a text_delimiter character is not used, then the text string shall not contain any punctuation character that is a valid successor to the text string in the BEHAVIOUR template (i.e. ";").

C.3.2 Interface template

<computational_interface_template> ::= **<computational_interface_header>**
"{" <computational_interface_body> "}"

<computational_inteface_header> ::= **"COMPUTATIONAL_INTERFACE"**
<interface_name>

<interface_name> ::= **<identifier>**

<computational_interface_body> ::= **["DERIVED FROM" <interface_label>]**
<operation_definitions>
[<behaviour_definition>]

<interface_label> ::= **<server_interface_label> | <client_interface_label>**

<operation_definitions> ::= **"OPERATION"**
{<operation_label> ";"}*

<operation_label> ::= **<label_reference>**

C.3.3 Operation template

<operation_template> ::= **<operation_header> "{" <operation_body> "}"**

<operation_header> ::= **"OPERATION" <operation_name>**

<operation_name> ::= **<identifier>**

<operation_body> ::=
{INPUT_PARAMETERS" [{<param_label> ":" <syntax_label> ";"}*]}]

[{OUTPUT_PARAMETERS" [{<param_label> ":" <syntax_label>";"}*]}]

[{RAISED_EXCEPTIONS" [{<exception_label>":" <syntax_label>";"}*]}]
<opn_behaviour_definition> ";"]

<param_label> ::= **<identifier>**

<exeption_label> ::= **<identifier>**

<syntax_label> ::= **<primitive_asn1_type_name> | <module_name> "::" <production_name>**
<type_production> | <production_name>

<production_name ::= **<identifier>**
<module_name>::= <identifier>

<primitive_asn1_type_name>::= <identifier>

<asn1_production> ::= **<type> "::=" <comp_type_def>**

<comp_type_def> ::= **<interface_reference> | <sinagleASN1typedef>**
-- Imported from Recommendation X.208
<ref_spec> ::= **<identifier>**

<opn_behaviour_definition> ::= "BEHAVIOUR"
["INFORMAL" [{<text_delimiter> <string_literal> <text_delimiter>
|<string_literal> ";"}]]
"SEMI-FORMAL"
{"PARAMETER_MATCHING"
{<param_label> ["ELEMENTS"] ":" <parameter_matching_expression> ";"}*}
{[{"PRE_CONDITIONS" [{<schema_label> | {<text_delimiter>
<string_literal> <text_delimiter>}";"}]}]
[{"POST_CONDITIONS" [{<schema_label> | {<text_delimiter>
<string_literal> <text_delimiter>}";"}]}]} |
{TRIGGERING_CONDITIONS
[{<transition_label> | <text_string> |
<state_label> TRANSITION_TO <state_label> ";" }]}]

[{EXCEPTIONS
[{"IF <exception_invariant_label> "NOT_VERIFIED"
"RAISE_EXCEPTION" <exception_label> ";"}*]}]

<schema_label> ::= **<label_reference>**

<invariant_label> ::= **<label_reference>**

<exception_invariant_label> ::= {"PRE_CONDITION" <label_reference>}
| {"POST_CONDITION" <label_reference>}
<transition_label> ::= **<label_reference>**

<text_string> ::= **<string_literal>**

<type>::= **<identifier>**

<string_literal>::= [a-zA-Z] [-a-zA-Z0-9_:.]*

<state_label> ::= **<label_reference>**

parameter_matching_expression::=
<label_reference>
|<parameter_matching_expression> "AND" <parameter_matching_expression>
|<parameter_matching_expression> "OR" <parameter_matching_expression>
|"NOT" <parameter_matching_expression>
|"{" <parameter_matching_expression>"}"

<param_reference> ::= <label_reference> | "NOT {" <label_reference> "}"

C.4 Example

This example shows the computational interface and an operation template which provides the setup of a subnetwork connection in the subnetwork connection management community.

simple SNC performer interface

The simple Sub-network performer manages the setup and release of Sub-network Connections

The simple SNC performer interface is required to satisfy the enterprise requirements stated in:

<"Rec. G.852.1", COMMUNITY:sscc, ACTION:sccc1 > ,

<"Rec. G.852.1", COMMUNITY:sscc, ACTION:sccc2 > .

The simple subnetwork connection performer interface provides basic connection setup functionality. The operation ssccSetupSubnetworkConnection sets up a subnetwork connection, and the operation ssccReleaseSubnetworkConnection removes the subnetwork connection.

COMPUTATIONAL_INTERFACE simpleSncPerformerIfce {
OPERATION <setupSubnetworkConnection>;

<ssccReleaseSubnetworkConnection>;
}
sscc set up SNC

This operation sets up a simple subnetwork connection between a single A-End snTP or nTP, and a single Z-end snTP or nTP.

OPERATION ssccSetupSubnetworkConnection {

INPUT_PARAMETERS
subnetwork : SubnetworkId ::= (ssccSnIfce);
-- The subnetwork parameter is used to indicate the subnetwork across which the performer is
-- setting up the SNCs. This parameter is used, for example, when a given performer can set
-- up SNCs in many subnetworks. If the performer is associated with a single subnetwork, the
-- subnetwork parameter of this operation is redundant and may be removed as an
-- engineering optimization.
snpa : SnTPId ::= (snTPIfce);
snpz : SnTPId ::= (snTPIfce);
dir : Directionality;
suppliedUserLabel : UserLabel;

-- zero length string implies none supplied
serviceCharacteristics: CharacteristicsId ::= (serviceCharacteristicsIfce);
-- reference can be used to determine any QOS or routing characteristics);

OUTPUT_PARAMETERS
newSNC : SNCId ::= (sncIfce);
agreedUserLabel : UserLabel;

RAISED_EXCEPTIONS
invalidTransportServiceCharacteristics: NULL;

incorrectSubnetworkTerminationPoints : SEQUENCE OF SnTPId;
-- the list contains one element when only point is incorrect.
subnetworkTerminationPointsConnected : SEQUENCE OF SnTPId;
-- the list contains one element when only one subnetworkTerminationPoint
-- remains connected.
failure : Failed;
wrongDirectionality : Directionality;
userLabelInUse : UserLabel;
BEHAVIOUR
INFORMAL
!

This operation sets up a subnetwork connection between a given A-End snTP or nTP and a given Z-End snTP. The subnetwork termination points or network termination points to be connected are specified by explicitly identifying the subnetwork network termination points or network termination points.

The client may supply a unique user label. If not supplied (i.e. string length zero) the provider assigns a user label for the connection.

A subnetwork connection may only be established in the state 'connected'.

A single point to point unidirectional, or point to point bidirectional, unpartitioned subnetwork connection object will be created. The subnetwork connection object will have one A-End and one Z-end.

The subnetwork connection will have a directionality (unidirectional or bi-directional) as specified in the operation parameters.

If used, the service characteristics specify one pre-determined set of transport parameters which the server may offer.

The operation replies for set-up includes full information about the reasons in case the request could not be satisfied.

PRE_CONDITIONS

The snTP or nTPs must be in existence before a subnetwork connection can be made within any given subnetwork.

This operation will fail if any of the subnetwork termination points or network termination points specified is already involved in a subnetwork connection. The exception 'subnetworkTerminationPointsConnected' will be generated.

This operation will fail if the subnetwork termination points or network termination points are not contained within the domain of the subnetwork. The exception 'incorrectSubnetworkTerminationPoints' will be generated.

This operation will fail if the serviceCharacteristics requested is not supported by the computational object which executes the operation. The exception 'invalidTransportServiceCharacteristics' will be generated.

POST_CONDITIONS

If any subnetwork connection input parameters cannot be met by the server, the operation will fail.

This operation will fail if the value of the userLabel of the SubnetworkConnection is zero or is not unique within the domain of the containing subnetwork. The exception 'userLabelInUse' will be generated.
!
SEMI_FORMAL

```
            PARAMETER_MATCHING
            subnetwork: < ssccNotConnected, ROLE:involvedSubnetwork > AND
                        < ssccConnected, ROLE:involvedSubnetwork >;
            snpa : < ssccNotConnected, ROLE:potentialAEnd > AND
                    < ssccConnected , ROLE:connectedAEnd >;
            snpz : < ssccNotConnected , ROLE: potentialZEnd > AND
                    < ssccConnected , ROLE:connectedZEnd >;
            dir : < ssccConnected, ROLE: involvedSubnetwork ,ATTRIBUTE:
                    directionality >;
            newSNC : <ssccConnected, ROLE: involvedSubnetwork>;
            suppliedUserLabel : <ssccConnected, ROLE:involvedSubnetwork, ATTRIBUTE: userLabel >

            OR <> ;  -- The user does not have to supply a user label value
            agreedUserLabel : <ssccConnected, ROLE:involvedSubnetwork, ATTRIBUTE: userLabel >;
            serviceCharacteristics : < ssccConnected , ROLE:involvedServiceCharacteristics >;

            PRE_CONDITIONS   < ssccNotConnected> ;
        -- The ssccNotConnected schema defines a schema type with two non-connected
        -- networkTP information objects subtypes candidates to the point-to-point connection
        -- management service.
            POST_CONDITIONS < ssccConnected> ;
        -- The ssccConnected schema defines the schema type of two connected networkTP
        -- information objects candidates to the point-to-point connection management service.
            EXCEPTIONS

            IF PRE_CONDITION <inv_1> NOT_VERIFIED RAISE_EXCEPTION
                    incorrectSubnetworkTerminationPoints;
            IF PRE_CONDITION <inv_2> NOT_VERIFIED RAISE_EXCEPTION
                    subnetworkTerminationPointsConnected ;
            IF PRE_CONDITION <inv_3> NOT_VERIFIED RAISE_EXCEPTION
                    subnetworkTerminationPointsConnected ;
            IF POST_CONDITION <inv_1> NOT_VERIFIED RAISE_EXCEPTION
                    failure;
            IF POST_CONDITION <inv_2> NOT_VERIFIED RAISE_EXCEPTION
                    failure;
            IF POST_CONDITION <inv_3> NOT_VERIFIED RAISE_EXCEPTION
                    failure;
            IF POST_CONDITION <inv_4> NOT_VERIFIED RAISE_EXCEPTION
                    userLabelInUse;
            ;
}
```

ANNEX D

OSI management engineering viewpoint templates and guidelines

This Annex includes the templates used in the engineering viewpoint, and also provides a set of guidelines for transforming computational and information objects into engineering GDMO objects that are members of a specific scenario.

Completion of this Annex is for further study.

D.1 Templates

The ensemble template is used to define the scenario objectives that the engineering objects must meet.

The GDMO template is used for the definition of engineering objects.

D.2 Possible mappings to managed object definitions

A managed object in the engineering viewpoint represents a mapping of information and computational objects into GDMO, that can facilitate an implementation using OSI management and CMISE/CMIP.

The guidelines for creating managed object definitions are as follows:

1) Target managed objects are constructed from information and computational objects such that:
   a) attributes come from the information objects in the information viewpoint;
   b) computational server interfaces may be defined as:
      – separate packages in the engineering viewpoint;
      – separate GDMO object altogether (if the server interface is instantiated);
   c) operations within a server interface may be defined as:
      – GET, GET-REPLACE, REPLACE operations on attributes;
      – separately defined actions.
2) Some client interfaces could be realized as X.500 directory service queries (e.g. computational operations that are performing name/address resolution).
3) It is possible to define a managed object class derived from a common supertype of the list of object classes, with its behaviour class stating that its instances are compatible with the object classes in the list. This can be used in the actual GRM in the domain-specific computational specification and in the engineering viewpoint.

ANNEX E

Label syntax

E.0 Introduction

The label reference is provided as a string. A "," in the string indicates that the label entity to the left must be evaluated before the remainder of the string is processed. For example in the string:

<INFORMATION_RELATIONSHIP:compoundLinkHasLinks,ROLE:container,ATTRIBUTE: directionality> is a pointer to the ATTRIBUTE directionality in the INFORMATION_OBJECT that is playing the ROLE container in the compoundLinkHasLinks relationship.

E.1 BNF definition of label syntax

```
<label_reference>      ::= "<"<label_string>">"

<label_string> ::= <element_label>
                       | <label_list>
                       | <element_label>","<label_list>

<label_list>       ::= <label_entity>
                       | <label_entity> ","<label_list>

<label_entitiy>    ::= <clause_label>":"<element_label>

<element_label>    ::= <label_text> -- a quoted text string, or a text string
                           -- without spaces or colons or slashes
                           -- or open_angle_brackets.
                       |   -- null choice used when there is no label for the keyword
```

<clause_label> **::= <enterprise_clause_label>**
| <information_clause_label>
| <computation_clause_label>
| <engineering_clause_label>
-- extend as necessary

<enterprise_clause_label> ::=
| "COMMUNITY"
| "PURPOSE"
| "ROLE"
| "POLICY"
| "ACTION"
| "ACTION_POLICY"
| "ACTIVITY"
| "ACTIVITY_POLICY"
| "WITH_ACTION_GRAPH"
| "PERMISSION"
| "PROHIBITION"
| "OBLIGATION"
| "EXCEPTION"
| "CONTRACT"

-- See E.2 for the complete label tree for the enterprise viewpoint

<information_clause_label> ::=
| "INFORMATION_OBJECT"
| "INFORMATION_RELATIONSHIP"
| "STATIC_SCHEMA"
| "DYNAMIC_SCHEMA"
| "ATTRIBUTE"
| "STATE"
| "DEFINITION"
| "INVARIANT"
| "TRANSITION"
| "POTENTIAL_RELATIONSHIP"
| "ROLE"
| "PRE_CONDITION"
| "POST_CONDITION"
| "RELATIONSHIP"

-- See E.3 for the complete label tree for the information viewpoint

<computational_clause_label> ::=
| "COMPUTATIONAL_OBJECT_CLASS"
| "SERVER_INTERFACES"
| "CLIENT_INTERFACES"
| "BEHAVIOUR"
| "COMPUTATIONAL_INTERFACE"
| "DERIVED_FROM"
| "OPERATIONS"
| "OPERATION"
| "INPUT_PARAMETERS"
| "OUTPUT_PARAMETERS"
| "RAISED_EXCEPTIONS"
| "PARAMETER_MATCHING"
| "PRE_CONDITIONS"
| "POST_CONDITIONS"
| "TRIGGERING_CONDITIONS"
| "TRANSITION_TO"
| "EXCEPTIONS"
| "NOT_VERIFIED"

| "RAISE_EXCEPTION"
| "PRE_CONDITION"
| "POST_CONDITION"
| "AND"
| "OR"
| "NOT"
| "INFORMAL"

-- See E.4 for the complete label tree for the computational viewpoint

<engineering_clause_label>

-- For Further Study – Dependent on communications domain selected

E.2 Enterprise viewpoint label tree structure

"Rec. G.852.xx"
COMMUNITY
<label>
PURPOSE
ROLE
<label>
PERMISSION
<label>
OBLIGATION
<label>
PROHIBITION
<label>
EXCEPTION
<label>
ACTION
<label>
PERMISSION
<label>
OBLIGATION
<label>
PROHIBITION
<label>
EXCEPTION
<label>
ACTIVITY
<label>
PERMISSION
<label>
OBLIGATION
<label>
PROHIBITION
<label>
EXCEPTION
<label>
EXCEPTION
<label>
WITH_ACTION_GRAPH
<label>
CONTRACT
<label>

E.2.1 Examples of use

<"Rec. G.852.1",COMMUNITY:sscc,ROLE:caller>
<"Rec. G.852.1",COMMUNITY:sfm,ACTION:sfm3,OBLIGATION:OBLG_1>

E.3 Information viewpoint label tree structure

"Rec. G.853.xx"
- **INFORMATION_OBJECT**
  - **<label>**
    - **DEFINITION**
    - **ATTRIBUTE**
      - **<label>**
        - **STATE**
          - **<label>**
        - **INVARIANT**
          - **<label>**
        - **TRANSITION**
          - **<label>**
    - **INVARIANT**
      - **<label>**
    - **TRANSITION**
      - **<label>**
    - **RELATIONSHIP**
      - **<label>**
    - **POTENTIAL_RELATIONSHIP**
      - **<label>**
- **INFORMATION_RELATIONSHIP**
  - **<label>**
    - **DEFINITION**
      - **<label>**
    - **ROLE**
      - **<label>**
    - **INVARIANT**
      - **<label>**
    - **TRANSITION**
      - **<label>**
- **STATIC SCHEMA**
  - **<label>**
    - **DEFINITION**
      - **<label>**
    - **ROLE**
      - **<label>**
    - **INVARIANT**
      - **<label>**
- **DYNAMIC_SCHEMA**
  - **<label>**
    - **DEFINITION**
      - **<label>**
    - **PRE_CONDITION**
      - **<label>**
    - **POST_CONDITION**
      - **<label>**
- **ATTRIBUTE**
  - **<label>**
    - **DEFINITION**
      - **<label>**
    - **STATE**
      - **<label>**

INVARIANT
<label>
TRANSITION
<label>

E.3.1 Examples of use

<"Rec. G.853.1",INFORMATION_OBJECT:networkConnectivity,ATTRIBUTE:signalIdentification>

<"Rec. G.853.2",INFORMATION_OBJECT:monitoredEntity,ATTRIBUTE:operationalStateSTATE: enabled>

<"Rec. G.853.1",INFORMATION_RELATIONSHIP:clientServer,ROLE:client>

<"Rec. G.853.1",INFORMATION_RELATIONSHIP:clientServer,INVARIANT:inv_1>

<"Rec. G.853.2",STATIC_SCHEMA:enabledAndReportOn,INVARIANT:enabled>

<"Rec. G.853.2",DYNAMIC_SCHEMA:reportFailureOnEnabledToDisabled,
PRE_CONDITION:EnabledAndReportOn>

<"Rec. G.853.1",ATTRIBUTE:directionality,STATE:unidirectional>

E.4 Computational viewpoint label tree structure

"Rec. G.854.xx"
COMPUTATIONAL_OBJECT_CLASS
<label>
SERVER_INTERFACES
<label>
CLIENT_INTERFACES
<label>
COMPUTATIONAL_INTERFACE
<label>
OPERATIONS
<label>
OPERATION
INPUT_PARAMETERS
<label>
OUTPUT_PARAMETERS
<label>
RAISED_EXEPTIONS
<label>
PARAMETER_MATCHING
<label>
PRE_CONDITIONS
<label>
POST_CONDITIONS
<label>
TRIGGERING_CONDITIONS
<label>
EXCEPTIONS
<label>

E.4.1 Examples of use

<"Rec. G.854.1",COMPUTATIONAL_INTERFACE:simpleSncPerformerIfce,
OPERATION:releaseSNC,INPUT_PARAMETER:userLabel>

<"Rec. G.854.1",COMPUTATIONAL_INTERFACE:snQueryIfce,
OPERATION:querySnForSNCs,RAISED_EXCEPTIONS:unconnectedSubnetwork>

ANNEX F

Ensemble template

F.1 The ensemble technique

This Annex describes how the ensemble concept and format of the NM Forum may be used for the network level view within the RM-ODP framework.

The contents of the ensemble are, as far as possible, references to the appropriate viewpoint documents. In some cases, to aid readability, some of the text may be duplicated in the ensemble.

The contents description and template proforma can be found in Network Management Forum Document Forum 025, OMNIPOint 1, The "Ensemble" Concepts and Format, August 1992 [8].

F.2 Ensemble template

The ensemble template consists of the following parts:

– Introduction.

– Management context.

– Management information model.

– Ensemble conformance requirements.

The following subclauses describe these parts of the ensemble specification.

F.2.1 Introduction

The introduction consists of a textual summary of the application scenario by identifying the enterprise viewpoint communities that must be supported. It consists of the following items:

– global description of the management problem using the global text in the enterprise viewpoint;

– identification of the resources to be managed;

– global requirements and constraints to be met;

– identification of the interactions across the engineering interfaces.

F.2.2 Management context

This subclause consists of the following items:

– more detailed description of the management requirements and constraints using the policies stated in the enterprise viewpoint (e.g. by defining which of the permissions are obligations in this management context);

– description of the resources to be managed by reference to the information viewpoint;

– definition of the interface protocol to be supported;

– definition of the scenarios in which the message flows across the engineering interfaces are identified and specified, by referencing the computational interfaces.

F.2.3 Management information model

This subclause references the engineering viewpoint that has been developed for this application scenario.

F.2.4 Ensemble conformance requirements

The following items are to be identified:

– general conformance requirements;
– functional support;
– the engineering objects;
– conformance requirements for the engineering solutions, such as:
  • objects (e.g. MOCS proformas for GDMO based engineering solutions);
  • protocol (e.g. using PICS proforma);
  • use of directory service.

An example of the use of the ensemble concept can be found in Appendix V.

APPENDIX I

Examples of templates and specifications guidelines

I.1 Enterprise services versus contracts

The notion of service and contract are closely related. In fact, the service is the external perception of the contract application. The two concepts can, therefore, be used interchangeably.

A contract type can be private or standardized. In case of private contracts, the provider is responsible for the establishment and maintenance of them (alone or in association with its clients). If the contract is the result of the standardization process, it is established and maintained by standards organizations (e.g. ITU-T). Standards organizations cannot establish contract instances.

The terms in which contract features are structured and written are very important since they will be used to establish the responsibilities in case of contract transgressions. In this sense, a contract type defines enterprise conformance points.

There is a requirement to establish contract types starting from previous ones by adding new features or by developing compound features. It is useful in the following circumstances:

– to construct new services by reusing existing ones;
– to develop private contracts starting with standardized ones;
– to guarantee backward compatibility in case of service enhancements.

A composite service is the union of:

– all the features that are defined in the services that are composing it (called imported services); and
– the locally defined features.

Two rules have to be respected with regard to composition of services:

– the imported services have to be consistent with each other. That is, they do not contain contradictory features;
– the locally added features must be consistent with the imported ones.

APPENDIX II

Representation of combined states

Consider an application where the requirement is to express whether a resource can supply its normal service. This state could be viewed across interface A to a computational object. See Figure II.1.

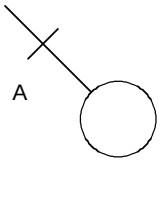


Figure II.1/G.851.1 – Computational interface

The state of the resource viewed across interface A can be defined by the in service or resource failed states (S1, S2). These can be mapped on to values of the operational state of enabled or disabled as shown in Figure II.2. S1 and S2 may also be considered as (simple) static schema.

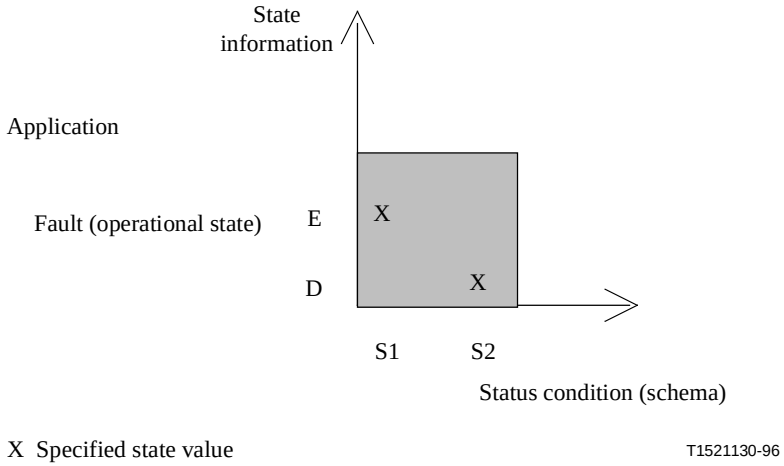


Figure II.2/G.851.1 –State description for fault application

Now consider the situation in Figure II.3 where the state of the resource may be viewed via multiple interfaces. Interface A remains as above but interface "B" is used for configuration and provisioning applications as well. The issue is to allow an application to interface to the object via interface "A" which only knows about states S1 and S2, and also to allow interactions via interface "B" which has a wider range of states.

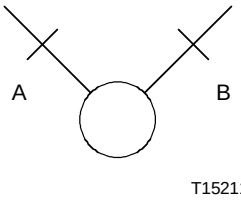


Figure II.3/G.851.1 – Computational interfaces

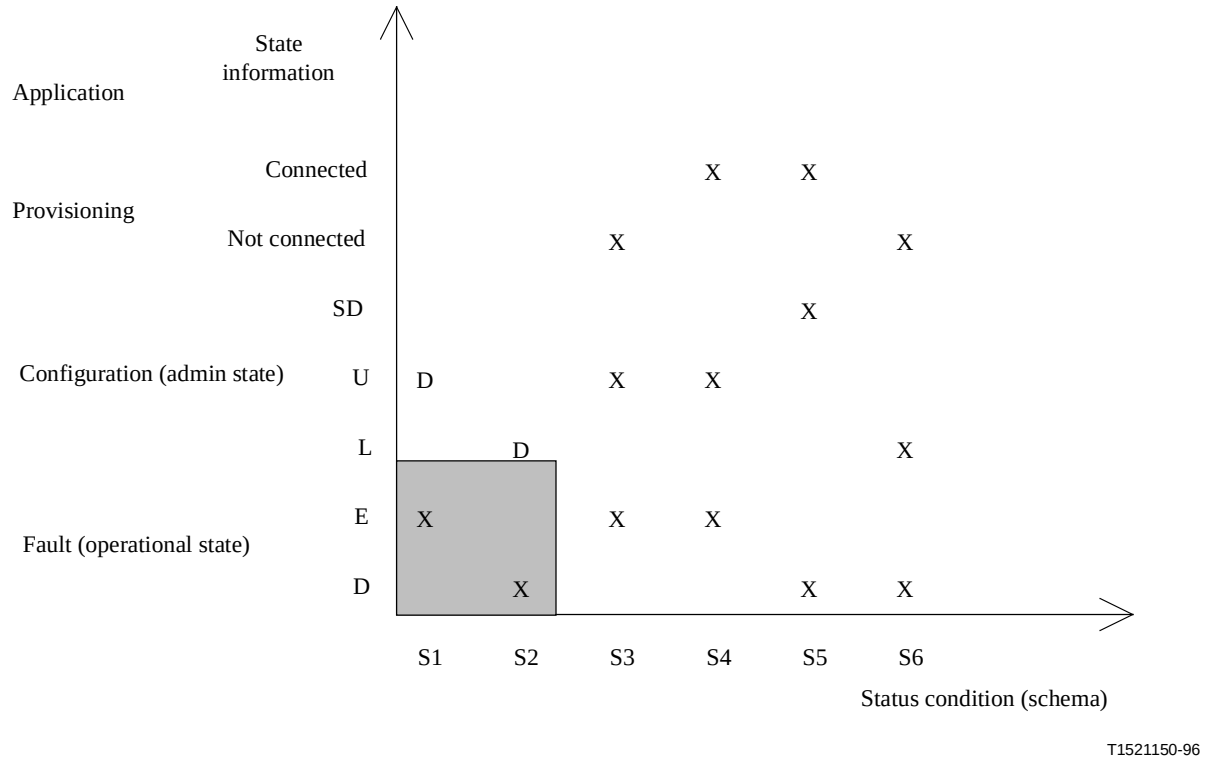
Figure II.4 shows the range of states visible across interface "B". States S1 to S6 comprise the state table. Each state is composed from the base states: administrative state, operational state, and the connected/not connected status condition. S3 to S6 may also be defined as static schema.

States S1 and S2 were defined without the knowledge of other states. Therefore in the interface "B" state table they are assigned default values for base states administration, and the connected/not connected status condition.

To define the state S5, three base state values are required, while to define state S1 only one base state value is required. If the other two base states of the resource are made visible via interface B, then the system must maintain consistency of states S1 and S2 within the context of wider set of states. It may be considered that the additional states are assigned default values to retain consistency in the wider state table. If S1 to S6 are considered as static schema then when schema S3 to S6 are defined S1 and S2 are unchanged. However, the system must ensure that the values of operational state (S1 and S2) reported over interface A are consistent with the values of operational state (of the same resource) reported as S3 to S6 over interface B. In this case for example if schema S1, S3 and S4 require that the operational state is enabled while schema S2, S5 and S6 require that the operational state be disabled then if S3 or S4 is reported via interface B then S1 must be reported over interface A. In this way the original states which were defined with a single base state are still valid when they become part of a much larger state table. A user of interface A will not notice the impact if the computational object is enhanced to support interface B.

The set of states in an object or system can be extended by adding new base states and defining the relationship between the new base states and the existing schema. The extension of state by adding a new value to an existing base state should be avoided. However if this cannot be avoided then a mapping from the new (extended) base state to the existing base state must be provided.

Similar considerations apply to produce two separate profiles of the information object for two separate applications (and different computational objects).



X Specified state value
D Assigned default state value

Figure II.4/G.851.1 – State description for combined application showing use of default values

APPENDIX III

Service realization description

III.1 Scope

The way by which the provider performs the service is not relevant to the client, and hence is not part of the contract specification. However, the provider has to document, from the enterprise perspective, the way a given service is realized. A service realization includes two characteristics:

– the internal policy that is applicable for each contract feature;

– the subsequent services that are invoked as a consequence of the policy application.

For these subsequent services, the former provider will act as a client. Obviously, the subsequent services may be standardized or private, depending on the policy that governs their use.

From the standards perspective, it could be useful to provide an informative behaviour that will govern the use of subsequent standardized services. The rationale is:

– to justify the introduction of new standardized services;

– to be able to express service policy constraints between related standardized services.

For several reasons, a provider may decide to handle several service realizations for a particular service specification (e.g. changes in the enterprise policy). If the service specification is not changed then the clients are not aware of a change in a service realization.

III.2 Concepts

While the previous subclause can be subject to standardization, this subclause provides a solution to realize the provision of the service; any solution is obviously not unique and can be changed according to the service provider policy. A policy will be developed for each action or activity defined in the contract. For each contract, the standards bodies may provide a service realization section in order to indicate how standardized services may be related to each other, for example.

In any case, it is useful to produce standardized templates for the service realization description as part of this Recommendation in order to enable service providers to describe their own realization.

The service realization section will be structured into subsections, one for each action or activity defined in the contract specification. Each subsection will start with a community item indicating the roles that are involved to perform the action or activity. The enterprise object taking the service provider role with regards to the service contract will always take service caller roles with regard to subsequent services.

The community item is followed by a policy item, where each policy rule is equivalent to an action.

As a consequence, the realization policy associated with a service contract action (respectively a service contract activity) will form an activity description as it will be expressed as an acyclic graph of actions.

The term action is used in the RM-ODP sense, i.e. it may be internal or it may be an interaction. This subsection selects whether actions are internal or interactions as part of a realization decision. When an action is an interaction, it will be expressed in terms of service invocations; otherwise, it will be internal. See Figure III.1.

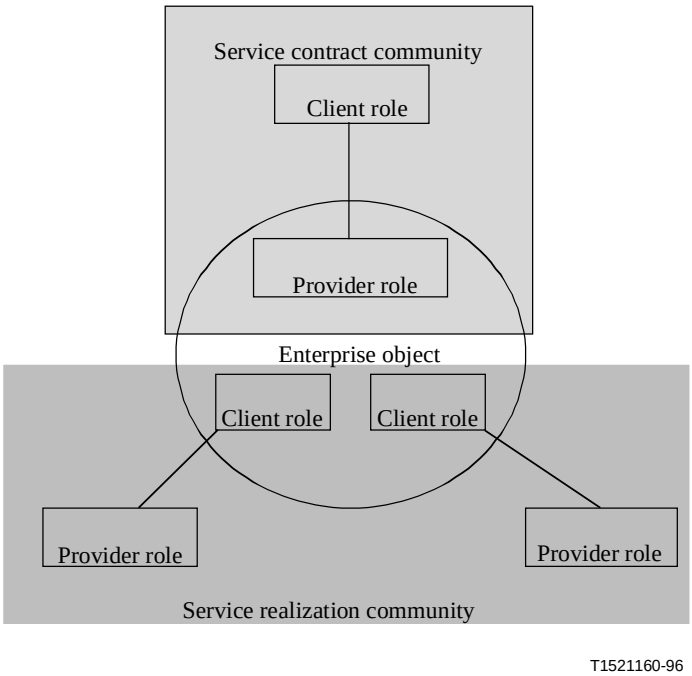


Figure III.1/G.851.1 – Example of role versus object

An action is not instantaneous and can be defined as:

– is occurring: the action has started. This action can be asynchronous, and other actions can be realized at the same time.

– has occurred: the action is finished. This is a synchronous action and no other action following it in the graph can start before its completion.

– may occur: the action is optional, and may be realized following some particular requests.

As described in RM-ODP Part 2 [2], actions may overlap in the time. However, RM-ODP does not precisely specify whether some temporal constraints exist between actions. For specification purposes, an action which is occurring may be started only if another preceding action has occurred.

The graph description is specified as a schema in this subsection with labelled actions. Then each action is informally described by stating the following points:

– whether this action may occur, is occurring or has occurred;

– whether this action is internal or is an interaction (with the invoked service);

– indicates the reasons why this action occurs at this level of the graph.

When several actions appear at the same level in the graph, they may occur independently at any moment. The effective order will be specified in other viewpoints.

An example of a graph is presented in the Figure III.2.

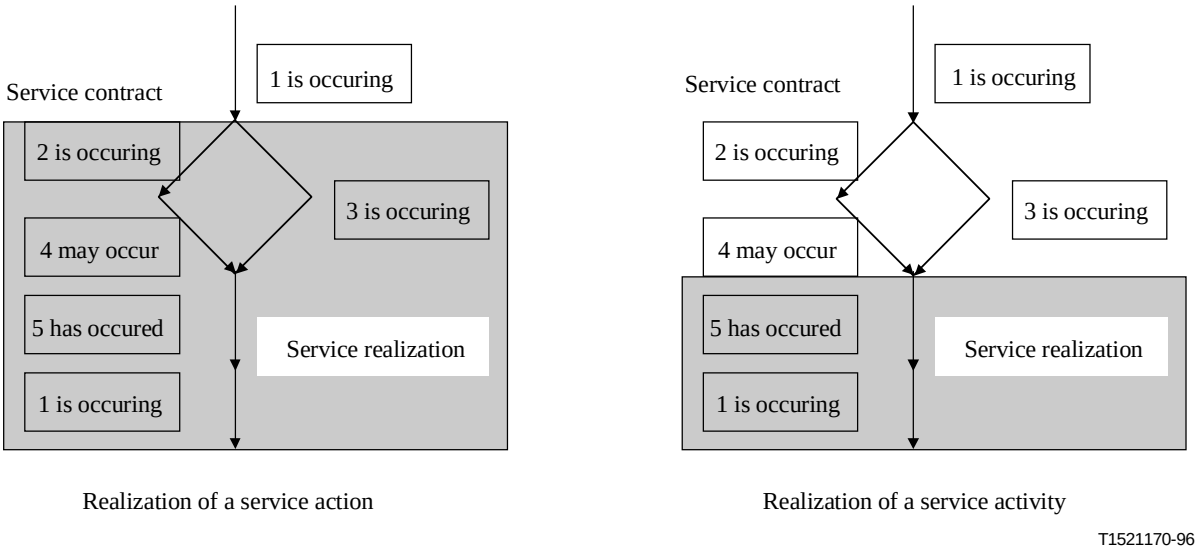


Figure III.2/G.851.1 – Example of activity description

Template

{ACTION |ACTIVITY} Name <action label>
Community
community_definition
Community Roles
<community label><Role Label>
role_definition
Community Policies
Action Name <action label>
action_definition
Action Policies
WITH ACTION GRAPH
Start
<action label>
<nth action label>
End

APPENDIX IV

Example of use of the ensemble concepts and format

Consider the example of the subNetworkConnection information object type defined in Recommendation G.853.1; an object of that type represents a G.805 subnetwork connection. Partial views of the underlying resource can then be described on a service-per-service basis. For instance, scmSubNetworkConnection and msSubNetworkConnectionManagement constitute respectively partial views of the subnetwork connection for the subnetwork connection configuration management (SCM) and the monitoring (MS) services. For each of these information object types, state variables may be defined in the information specification as attributes or relationships, for example.

In the computational specification of each service, a computational interface is then defined corresponding to the service provider capabilities. For instance, the sncConfiguration computational interface is defined as an enumeration of operations that can be invoked on that interface (e.g. setupSNCPointToPoint, setupSNCPointToMultiPoint, ReleaseSNC); it should be noted that each of these operations correspond to an enterprise action defined in the service contract and can be defined as:

```
INTERFACE_TEMPLATE sncConfiguration {
        OPERATION setupSNCPointToPoint
        OPERATION setupSNCPointToMultiPoint
        OPERATION ReleaseSNC
        BEHAVIOUR
                ...
}
```

Computational operations are described in terms of signature, pre-conditions and post-conditions. The two latter items refer to static schemas described in the information specification of the service. For instance, the operation setupSNCPointToPoint has the pre-condition scmNotConnected and the post-condition scmConnected. The transition between these two states constitute the behaviour of the operation. How the state transition is achieved is of no interest in the context of standardized behaviour specification; similarly, for how the invariants are respected. However, subsequent actions may be either considered as internal or as interactions with other objects on their open interfaces, if any.

To derive engineering interfaces, profiles must be developed. The description of a profile has four parts:

– **communication protocols**, e.g. CMISE/CMIP, SNMP, etc. The choice will have an influence on the engineering language;

– **managed resources**, e.g. list the resources that must be managed in the system (networkTPs, subNetworkConnections, subnetworks, etc.);

– **functions**, e.g. subnetwork connection configuration management and monitoring of networkCTPs and subnetworkConnections;

– **level of abstraction**, e.g. depending on whether the profile is described from the client perspective or from the provider perspective.

As an example, consider a Q3 interface to an SDH ring, where:

– CMISE/CMIP is supported at the interface;

– only subnetworks and subnetworkConnections are managed. This can be easily derived from the enterprise contract specifications;

– only subnetwork connection configuration and monitoring are expected. This can be easily derived from the enterprise contract specifications;

– the client view is only described.

Thus, an ensemble called "SNC" configuration and monitoring in a SDH ring can be described made up of:

1) Requirements which are captured in the enterprise contract specifications of services.

2) Scenarios which are captured in the enterprise contract specifications of services (basic services, enhanced services).

3) Resources, derived from the enterprise contract specifications of services and from the information specification of services.

4) Managed object specifications, which are part of the engineering specification.

5) Managed object conformance statements.

Modelling guidelines have to be defined and respected for managed object specifications. In particular, differences may appear in models depending on how to realize the interobject relationships mapping. An example is whether the relationship between a connection performer and the subnetwork is realized either through name binding between managed object classes or in conditional packages which may be imported into managed object classes (note that CMISE was selected, tightly coupled with full GDMO at the engineering viewpoint).

The resulting managed object class and package definitions would be as follows:

```
subnetwork MANAGED OBJECT CLASS
        DERIVED FROM "Recommendation X.721 | ISO/IEC 10165-2 : 1992":top;
        CONDITIONAL PACKAGES
                sncConfigurationPackage PACKAGE
                BEHAVIOUR
                        sncConfigurationPackageBehaviour BEHAVIOUR
                        DEFINED AS " ...";
                PRESENT IF "...";;;
REGISTERED AS {...};

subnetworkConnection MANAGED OBJECT CLASS
        DERIVED FROM "Recommendation X.721 | ISO/IEC 10165-2 : 1992":top;
        CONDITIONAL PACKAGES
                sncMonitoringPackage PACKAGE
                BEHAVIOUR
                        sncMonitoringPackageBehaviour BEHAVIOUR
                        DEFINED AS " ...";
                PRESENT IF "...";;;
REGISTERED AS {...};

sncConfigurationPackage PACKAGE
        BEHAVIOUR ...
        ACTIONS
                setupSNCPointToPoint,
                setupSNCPointToMultiPoint,
                releaseSNC;
REGISTERED AS {...};

sncMonitoringPackage PACKAGE
        BEHAVIOUR ...
        NOTIFICATIONS
                operationalStateValueChangeNotification;
REGISTERED AS {...};
```

setupSNCPointToPoint ACTION
BEHAVIOUR
setupSNCPointToPointBehaviour BEHAVIOUR
DEFINED AS " ***See the definition of the behaviour of the setupSNCPointToPoint operation of the computational interface scmConfiguration*** **";;**
MODE CONFIRMED;
WITH INFORMATION SYNTAX ***see the INPUT PARAMETERS construct of the operation setupSNCPointToPoint*****;**
WITH REPLY SYNTAX ***see the OUTPUT PARAMETERS construct of the operation setupSNCPointToPoint;***
REGISTERED AS {...};

As much as possible, convergence should be achieved between resulting GDMO specifications and existing managed object classes libraries in two ways: reutilization of generic managed object classes for specific purposes or enhanced managed object class libraries starting from specific models.

APPENDIX V

Example specification development process

This process may be used to develop a complete set of specifications based on this methodology. See Figure V.1.

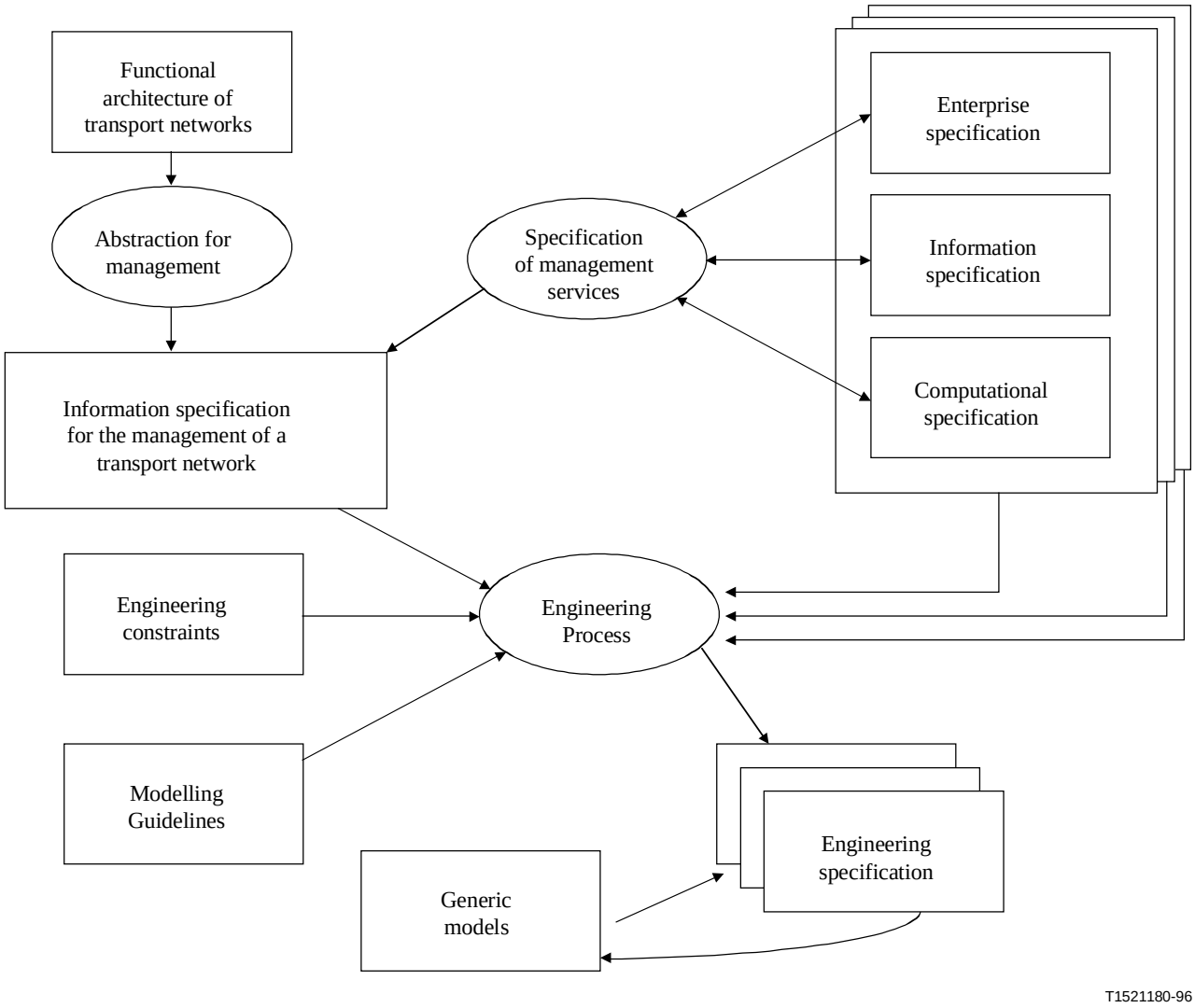


Figure V.1/G.851.1 – Specification development process

APPENDIX VI

Inter-viewpoint mapping

VI.1 Approach

The process used to derive the specifications for the network level model using the RM-ODP framework has resulted in a variety of relationships between elements (e.g. role, information object and static schema) associated with different viewpoints. It is of interest to characterize these inter-viewpoint relationships in terms of two principal types of relationships: *influence* and *reference*. The *influence* relationships are further differentiated as having either strong (or determining) influence or weaker (affecting) influence. A reference relationship borrows or maps

some or all of the information from one element to another. To avoid redundancy, the relationships are described only in the context of the related element within the viewpoint that is "closest" to the implementation specification (engineering). Thus, no inter-viewpoint relationships are discussed in the context of the enterprise viewpoint.

Figure VI.1 depicts the inter-viewpoint relationships considered to be *determining* and *reference*. A determining relationship is indicated by an arrow pointing from the influencing element to the influenced element. A reference relationship is shown as a line without arrowheads. Each of these relationships as well as other *affecting* relationships are described in the subclause below corresponding to the viewpoint of the element closest to the implementation specification. The engineering viewpoint is not addressed using this model since it is addressed in a separate context using an ensemble technique.

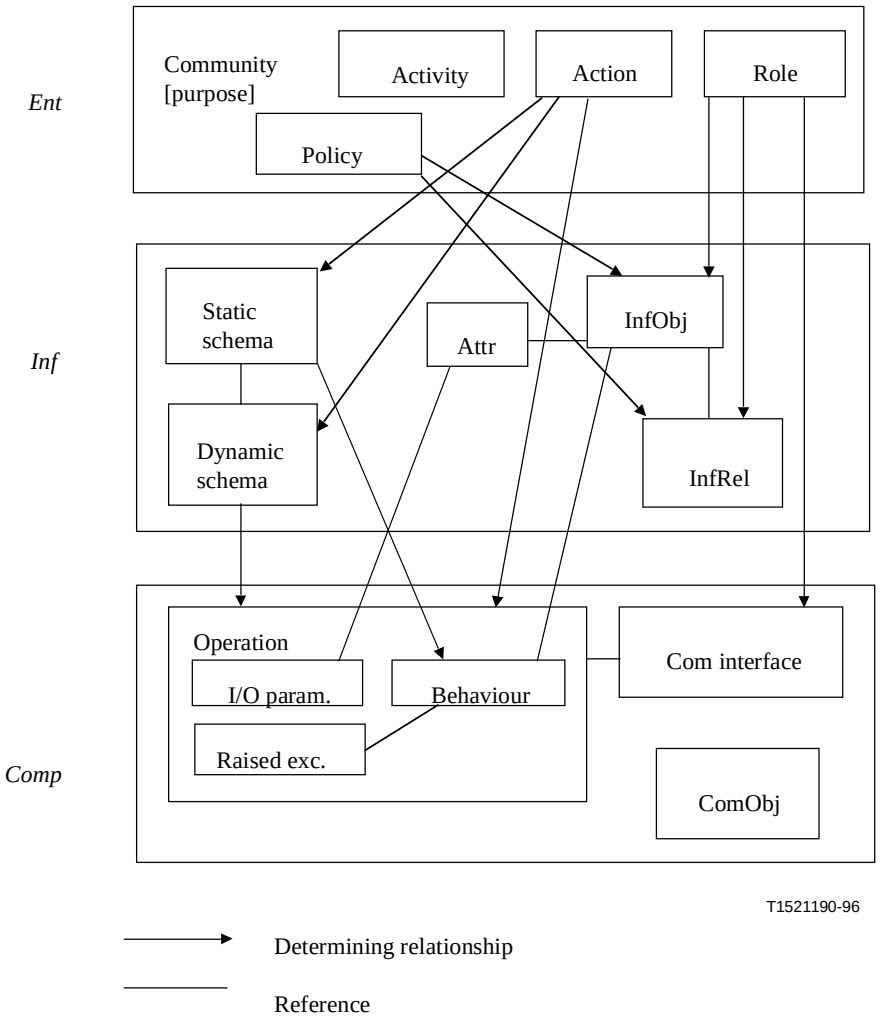


Figure VI.1/G.851.1 – Inter-viewpoint relationships

VI.2 Information viewpoint mappings

Key elements within the information viewpoint are information objects and attributes, information relationships, static schema and dynamic schema.

VI.2.1 Information objects and relationships

This selection of information objects and relationships for the information viewpoint is determined principally by the resource roles defined in the enterprise viewpoint. However, information objects and relationships may also be determined by community policies, action policies and activity policies. Specific attributes associated with information objects are determined by the community purpose and/or community actions. The need to include attributes in the information viewpoint may be affected by community policies and action policies.

VI.2.2 Static schema

The definition of a static schema in the information viewpoint is determined by one or more enterprise community action policies. The roles associated with a static schema reference information objects; invariants reference information relationships. Enterprise community policies affect static schema.

VI.2.3 Dynamic schema

Dynamic schema are defined to provide a way of referring to particular transitions between static schema. Dynamic schema are determined by actions.

VI.3 Computational viewpoint mappings

Key elements in the computational viewpoint are operations, computational interfaces and computational objects. Operations are characterized by input parameters, output parameters, raised exceptions, and behaviour. Computational interfaces represent assemblies of operations. Computational objects represent assemblies of computational interfaces with definitions of constraints on the interfaces. The need to allow for distribution is a key driver of the computational viewpoint, but is not included in the discussion of inter-viewpoint relationships.

VI.3.1 Computational operation

The general nature of an operation as well as each operation attribute is determined by community action (enterprise viewpoint) and dynamic schema (information viewpoint). A given operation may represent one or more community actions. Both input and output parameters reference information viewpoint attributes. Raised exceptions are generally determined by static schema invariants. Behaviour is determined principally on the basis of information viewpoint static schema. In particular, pre-conditions and post-conditions of an operation BEHAVIOUR are determined by static schema; PARAMETER MATCHING associates (references) particular roles in each static schema with particular information objects.

VI.3.2 Computational interface

The definition of a computational interface in terms of its comprised operations is determined by roles defined in the enterprise viewpoint. The nature of the interface (client or server) is also determined by the type of role.

VI.3.3 Computational object

The definition of a computational object as a collection of computational interfaces is determined principally by needs of the computational viewpoint, and thus is not strongly related to elements within other viewpoints.

APPENDIX VII

Guidelines for the use of Z in the information viewpoint

VII.1 Introduction

The formal parts of the information viewpoint specification have been specified using the Z notation. The schema notation of Z is not used directly to describe the object and relationship types that define the information viewpoint. Rather, the specification is based on two sets (OBJECT and RELATIONSHIP) that are opaque abstractions of all the possible object and relationship instances of an application. The system is described in terms of a set of mappings from the base sets to object and relationship characteristics. These mappings, taken together, describe the actual objects and relationships that comprise the system at a given time.

The attributes of an object and the role players in a relationship must be determined from the appropriate mappings. Z schemas are used to group these mappings in a way consistent with the concepts of object type declaration, as described in detail below. Each type (attribute, object or relationship type) is defined by two Z schemas; the first schema declares the components needed to define the type, and expresses its static invariants; the second defines the permitted state transitions for the type. The name of the first schema consists by convention of the name of the type followed by the suffix **_Static**; the name of the second schema similarly appends the suffix **_Dynamic**.

VII.2 Z notation review

VII.2.1 Schemas

Z is a formal notation based on set theory and first order predicate logic. The basic modelling concept in Z is the set, upon which more elaborate structure can be built. For example, a function is simply a (possibly infinite) set of ordered pairs. A set may be a given set (without internal structure of its members visible), may be defined by extension (by enumerating its elements) or may be defined by comprehension (by providing a base set and a predicate that all potential elements should verify). Encapsulation is provided in Z by the schema notation, which may be used in a number of ways. In the current specification, schemas are only used to encapsulate declarations and invariants for labelling and reuse. A named schema in Z has the following form:

_____ Schema-name _____

Declaration
Predicate

where:

- declaration is composed of a list of variables and their types (the sets to which their values belong); and
- predicate is a (possibly empty) list of conditions that the variable values must satisfy.

After this declaration, **schema-name** may be used in the declaration part of a subsequent schema, with the effect of adding **declaration** and **predicate** to the declaration and predicate parts of the including schema.

A standard Z convention typically used for describing state transitions is that the declaration Δ**schema-name** declares two copies of the variables in **schema-name**, one set of variables being decorated with a prime (**Ô**). Primed variables are interpreted as post-state variables, unprimed as pre-state.

VII.2.2 Symbols

Z provides a rich variety of notation applying to sets and to logic, which allows compact specification of data and functional aspects of a system. In the specification of information objects and relationships, only a small subset of these symbols is used, which are summarized below:

1) $\in$: set membership.

2) $\cup$: set union.

3) $\cap$: set intersection.

4) $\subseteq$: subset relation.

5) #: number of members of a set.

6) --+-->: a partial function. If X and Y are sets, X --+--> Y is the set of partial functions from X to Y. These are relations which relate each member x of X to at most one member of Y.

7) **dom**, **ran**: domain and range of a relation (in particular, a function). If R is a binary relation between X and Y, the domain of R (**dom** R) is the set of all members of X which are related to at least one member of Y by R. The range of R (**ran** R) is the set of all members of Y to which at least one member of X is related by R.

8) ~: relational inverse. If relation R maps a to b, then R~ maps b to a.

9) (| |): relational image. If R is a relation from X to Y, then for any subset S of X, R(|S|) is the set of values in Y related by R to a value in S.

10) \/: logical disjunction (or).

11) /\: logical conjunction (and).

12) $\Leftrightarrow$: logical implication.

13) $\forall$: universal quantifier. The predicate $\forall$ x: S | pre • cond, where pre and cond are logical formulas, may be read as "for each element x of set S satisfying the condition pre, the predicate cond holds".

14) $\exists$: existential quantifier. The predicate $\exists$ x : S | pre • cond may be read as "there exists an element x in set S satisfying the condition pre, for which the predicate cond holds".

15) **F**: finitepower set. If S is a set, **F** S is the set of all the subsets of S.

16) Δ: delta ("operation") schema naming convention (see VII.2.1).

NOTE – In this Appendix, many of the symbols above have a slightly different appearance from the actual specifications, where specialized fonts have been used.

VII.2.3 Example

The fragment of Z below introduces two given sets **X** and **Y**, then declares a schema **S** with two variables: **a**, of type **X** (i.e. a member of **X**) and **f**, a partial function from **X** to **Y**. The predicate of **S** asserts that **a** is one of the elements of **X** mapped by **f**, and that there are at least two elements of **Y** that **f** maps to.

[X, Y]

_____ S _____

a: X
f: X --+--> Y

---

a $\in$ **dom** f /\ #(**ran** f) > 1

VII.3 Specification conventions

The specification of a system consists of the attributes that objects may have, the object classes themselves, and the relationship classes. Z's schema notation is used to provide encapsulated declarations of all the mappings necessary to specify a given type. These mappings have domains within the given sets **OBJECT** (the set of all potential objects), and **RELATIONSHIP** (the set of all potential relationships).

[OBJECT, RELATIONSHIP]

VII.3.1 Attribute specification

An attribute type is modelled by a partial function (with the same name as the type) over the set of potential objects. The domain of the function is the set of instances of any object class that has that attribute. The values the function can take are the values the attribute can have. This range may be specified by the usual set enumeration and constructions of Z, and is declared in the _Static schema.

An attribute type may have dynamic behaviour constraints. These are declared in the _Dynamic schema, which refers to both pre- and post-state.

For example, the attribute **number** might be declared as:

_____ number _Static _____

number: OBJECT --+--> Integer

∀ n: **ran** number • n > 5

If the **number** attribute of an object never changed, the dynamic schema might be:

_____ number_Dynamic _____

Δnumber_Static

∀ obj: **dom** number ∪ **dom** number' • number'(obj) = number(obj)

indicating that an object that exists before and after a state transition of the system cannot change its **number** attribute.

VII.3.2 Object specification

In the specification style followed here, a schema describing the attributes of a single object instance in not defined. Instead, for each object type, a set of objects (the set having the name of the type) is defined. This object class is the set of all the actual objects of the system that satisfy the properties of the object type (in ODP terminology, an object class is the set of all the actual instances of an object type). The notion of instantiation (in ODP sense) is not modelled in the information viewpoint. Moreover, a Z schema defining an object class is not a template that allows the instantiation of an object of this class.

There are several structuring conventions that must be followed in order to describe object attributes, inheritance and invariants.

attributes

To specify that all the objects of a class have a given attribute, the _Static schema for that attribute must be included (explicitly or by inclusion of another object schema) in the declaration section of the object _Static schema. In addition, the predicate of the object schema must ensure that the attribute mapping applies to each member of the class (that is, the class members form a subset of the domain of the mapping).

subtyping

To specify a subtyping relationship (represented in GDMO by the mechanism of class inheritance), the _Static schema for each super-class (more than one for multiple inheritance) must be included in the declaration section of the object _Static schema. In addition, the predicate of the object schema must ensure that the members of the subclass are all members of the super-class(es) as well.

invariants

Constraints on the combinations of attribute values permitted for an object may be expressed in the predicate of the object _Static schema.

transitions

To specify the valid state transitions for an object type, a _Dynamic schema must be specified for the type. The signature for the schema must be the usual Z delta schema (describing primed and unprimed copies of the static variables), supplemented as follows: for each schema included in the _Static schema to describe inheritance and attributes of the type, a corresponding _Dynamic schema must be included in the object _Dynamic schema. In addition, further constraints may be added to the predicate of the object _Dynamic schema.

example

Suppose an object type **super** has already been defined. Then a subclass **sub** that adds an extra attribute could be defined by:

_____ sub_Static _____

sub: **F** OBJECT
super_Static
number_Static

---

sub subseteq super
sub subseteq **dom** number

_____ sub_Dynamic _____

Δsub_Static
super_Dynamic
number_Dynamic

VII.3.3 Relationship specification

Similarly, a relationship type is specified by describing the relationship class as a set of instances (in this case a subset of the set **RELATIONSHIP**, rather than **OBJECT**), together with mappings that in this case must describe roles as well as attributes.

roles

To specify that all the relationships of a class involve a particular role, the relationship _Static schema must declare (directly or by inheritance) an appropriately named partial function over the set of all possible relationship instances. In addition, the predicate of the relationship schema must ensure that the role mapping applies to each member of the relationship class.

role compatibility

The object classes compatible with a role are specified by including an appropriate set inclusion condition in the predicate of the relationship _Static schema. For a given relationship type, the range of the partial function modelling a role is constrained to be a subset of the union of the object classes compatible with this role.

role cardinality

The cardinality of a role may be specified in two ways. First, the declaration of the role may imply a certain cardinality: if the function modelling the role has the set OBJECT as its range set, the role has implied cardinality (1..1); if the function has as its range set **F** OBJECT (the set of subsets of OBJECT), the role has implied cardinality (0..N). In the latter case, an explicit predicate can restrict the cardinality further.

subtyping

Similarly to the object types, subtyping is represented by schema inclusion of the super-type(s) and predicate(s) that ensure that the new relationship class is a subset of the super-class(es). In addition, the object types that are permitted to play an inherited role may be restricted to a subset of the inherited object types by adding a predicate on the range of the appropriate role mapping.

invariants and attributes

Invariants for a relationship are specified in the predicate of the _Static schema. Since these invariants typically involve the attributes of the objects playing the roles, the _Static schema for any attribute referred to in an invariant must be included in the declarations for the relationship _Static schema.

transitions

In the same way as for the object transitions, the _Dynamic schema for a relationship is based on a delta of its _Static schema, and for any schema included in the _Static schema there must be a corresponding _Dynamic declaration in the relationship _Dynamic schema.

example

Suppose there was a relationship **rel** involving two roles: **firstRole**, played by a unique object of type **super** and **secondRole**, played by at least two objects of type **sub**, each of which must have a value of 10 as a **number** attribute. This might be specified by:

_____ rel_Static _____

```
rel: F RELATIONSHIP
    firstRole: RELATIONSHIP --+--> OBJECT
    secondRole: RELATIONSHIP --+--> F OBJECT
    super_Static
    sub_Static
    number_Static
```

```
rel subseteq dom firstRole
rel subseteq dom secondRole

∀ R: rel•
      firstRole(R) ∈ super /\
      secondRole(R) ∈ F sub

∀ R: rel•
      #(secondRole R) > 1

∀ R: rel•
      ∀ s: (secondRole R) • number(s) = 10
```

(Note that there are more compact ways of specifying the properties above. However, it is important to follow a systematic structure which allows easy identification of the invariants described in the informal specification.)

_____ rel_Dynamic _____

```
Δrel_Static
super_Dynamic
sub_Dynamic
number_Dynamic
```

ITU-T RECOMMENDATIONS SERIES

Series A	Organization of the work of the ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	**Transmission systems and media, digital systems and networks**
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communication
Series Z	Programming languages