INTERNATIONAL STANDARD

ISO/IEC 10746-1

First edition
1998-12-15

Information technology — Open Distributed Processing — Reference model: Overview

Technologies de l'information — Traitement réparti ouvert — Modèle de référence: Présentation

Reference number
ISO/IEC 10746-1:1998(E)

Contents

		Page
1	Scope and field of application	1
2	Normative references	1
2.1	Identical Recommendations | International Standards	1
2.2	Paired Recommendations | International Standards equivalent in technical content	2
2.3	International Standards	2
3	Definitions	2
3.1	Definitions in this Recommendation | International Standard	2
3.2	Definitions from other Recommendations | International Standards	2
4	Abbreviations	6
5	Conventions	7
6	ODP standardization	7
6.1	Objectives and motivation	7
6.2	Realization	8
6.2.1	Object modelling	8
6.2.2	Viewpoint specifications	9
6.2.3	Distribution transparency	9
6.2.4	Conformance	9
6.3	Standards	10
6.3.1	The Reference Model	10
6.3.2	Specific standards	10
7	Foundations	10
7.1	Basic modelling concepts	11
7.1.1	Objects	11
7.1.2	Interfaces and interaction points	11
7.1.3	Behaviour and state	12
7.2	Specification concepts	12
7.2.1	Composition/Decomposition	12
7.2.2	Behavioural compatibility	13
7.2.3	Type and class	13
7.2.4	Templates	13
7.2.5	Roles	13
7.2.6	Base classes and derived classes	14
7.3	Structuring concepts	14
7.3.1	Groups and domains	14
7.3.2	Naming	14
7.3.3	Contract	14
7.3.4	Liaison and binding	15
8	Architecture	15
8.1	Architectural framework	15
8.1.1	Viewpoints	15
8.1.2	Distribution transparencies	16
8.2	Enterprise language	17

8.3 Information language 19
8.4 Computational language 20
8.4.1 Computational interfaces 21
8.4.2 Binding model 21
8.4.3 Typing and subtyping for computational interfaces 23
8.4.4 Portability 24
8.5 Engineering language 24
8.5.1 Clusters, capsules and nodes 25
8.5.2 Channels 25
8.5.3 Interface references 28
8.5.4 Binding 29
8.5.5 Channel establishment 29
8.5.6 Management interfaces 30
8.5.7 Interceptors 30
8.5.8 Conformance points 32
8.6 Technology language 32
8.7 Consistency between viewpoints 32
8.7.1 Enterprise viewpoint consistency with other viewpoints 34
8.7.2 Correspondences between computational and engineering specifications 35
8.8 ODP functions 37
8.8.1 Management functions 38
8.8.2 Coordination functions 38
8.8.3 Repository functions 39
8.8.4 Security functions 39
8.9 ODP distribution transparencies 40
8.9.1 Access transparency 40
8.9.2 Failure transparency 40
8.9.3 Location transparency 40
8.9.4 Migration transparency 40
8.9.5 Persistence transparency 41
8.9.6 Relocation transparency 41
8.9.7 Replication transparency 41
8.9.8 Transaction transparency 41
9 Conformance assessment 41
9.1 Conformance assessment and the development process 41
9.2 Conformance assessment: Relevant relationships 42
9.3 Conformance points and related concepts 42
9.4 ODP conformance specifications 43
9.4.1 Level of abstraction 43
9.4.2 Use of multiple reference points 43
9.5 Conformance implications of viewpoint languages 44
9.6 Conformance assessment activities 44
10 Management of ODP systems 44
10.1 Management domains 45
10.2 Management policy 45
10.3 Modelling management structures 45
11 The use of standards in ODP systems 46
11.1 Enterprise viewpoint 46
11.1.1 Enterprise specification 46
11.1.2 The application of standards 47
11.2 Information viewpoint 47
11.2.1 Information specification 47
11.2.2 The application of standards 48
11.3 Computational viewpoint 48
11.3.1 Computational specification 48
11.3.2 The application of standards 49
11.4 Engineering viewpoint 49

11.4.1 Engineering specification 49
11.4.2 The application of standards 51
11.5 Technology viewpoint 51
11.5.1 Technology specification 51
11.5.2 The application of standards 52
12 Examples of ODP specifications 52
12.1 Multimedia Conferencing System 53
12.1.1 Introduction 53
12.1.2 Enterprise specification 54
12.1.3 Information specification 55
12.1.4 Computational specification 55
12.1.5 Engineering specification 58
12.1.6 Technology specification 60
12.2 Multiparty audio/video stream binding 60
12.2.1 General description 60
12.2.2 Enterprise specification 61
12.2.3 Information specification 62
12.2.4 Computational specification 64
12.2.5 Engineering specification 67
12.2.6 Technology specification 68
12.3 A management example – Metric Object 68
12.3.1 Enterprise specification 69
12.3.2 Information specification 70
12.3.3 Computational specification 71
12.4 Database example 72
12.4.1 Enterprise specification 72
12.4.2 Information specification 72
12.4.3 Computational specification 72
Annex A – Bibliography 76

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 10746-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 33, *Distributed application services*, in collaboration with ITU-T. The identical text is published as ITU-T Recommendation X.901.

ISO/IEC 10746 consists of the following parts, under the general title *Information technology — Open Distributed Processing — Reference Model*:

— *Part 1: Overview*

— *Part 2: Foundations*

— *Part 3: Architecture*

— *Part 4: Architectural semantics*

Annex A of this part of ISO/IEC 10746 is for information only.

Introduction

The rapid growth of distributed processing has led to a need for a coordinating framework for the standardization of Open Distributed Processing (ODP). This Reference Model provides such a framework. It creates an architecture within which support of distribution, interworking and portability can be integrated.

The Reference Model of Open Distributed Processing, ITU-T Rec. X.901 | ISO/IEC 10746-1 to ITU-T Rec. X.904 | ISO/IEC 10746-4, is based on precise concepts derived from current distributed processing developments and, as far as possible, on the use of formal description techniques for specification of the architecture.

The RM-ODP (ISO/IEC 10746) consists of:

- ITU-T Rec. X.901 | ISO/IEC 10746-1: **Overview:** contains a motivational overview of ODP giving scoping, justification and explanation of key concepts, and an outline of the ODP architecture. It contains explanatory material on how this Reference Model is to be interpreted and applied by its users, who may include standards writers and architects of ODP systems. It also contains a categorization of required areas of standardization expressed in terms of the reference points for conformance identified in ITU-T Rec. X.903 | ISO/IEC 10746-3. These common texts are not normative.
- ITU-T Rec. X.902 | ISO/IEC 10746-2: **Foundations:** contains the definition of the concepts and analytical framework for normalized description of (arbitrary) distributed processing systems. This is only to a level of detail sufficient to support ITU-T Rec. X.903 | ISO/IEC 10746-3 and to establish requirements for new specification techniques. These common texts are normative.
- ITU-T Rec. X.903 | ISO/IEC 10746-3: **Architecture:** contains the specification of the required characteristics that qualify distributed processing as open. These are the constraints to which ODP standards must conform. It uses the descriptive techniques from ITU-T Rec. X.902 | ISO/IEC 10746-2. These common texts are normative.
- ITU-T Rec. X.904 | ISO/IEC 10746-4: **Architectural semantics:** contains a normalization of the ODP modelling concepts defined in ITU-T Rec. X.902 | ISO/IEC 10746-2, clauses 8 and 9. The normalization is achieved by interpreting each concept in terms of the constructs of the different standardized formal description techniques. These common texts are normative.

This Recommendation | International Standard contains one annex.

Clause 6 explains the business benefits of open distributed systems, and how the RM-ODP and its associated ODP standards will enable corporations to realize these benefits. This clause states the “promises” of ODP – plug-and-play building blocks and system integration tools for distributed systems.

Clauses 7 to 10 explain what RM-ODP and its distributed functions are about. These clauses justify how RM-ODP supports the development of plug-and-play building blocks and system integration tools for distributed systems.

Clause 11 shows how ODP standards and specifications by other groups can be referenced in an ODP specification of a system. These relationships are key to ODP’s ability to enable integration of disparate technologies.

Clause 12 contains examples that demonstrate the use of RM-ODP and the use of underlying principles to solve business problems.

INTERNATIONAL STANDARD

ITU-T RECOMMENDATION

INFORMATION TECHNOLOGY – OPEN DISTRIBUTED PROCESSING – REFERENCE MODEL: OVERVIEW

1 Scope and field of application

This Recommendation | International Standard:

- gives an introduction and motivation for ODP;
- provides an overview of the Reference Model of Open Distributed Processing (RM-ODP) and an explanation of its key concepts;
- gives guidance on the application of the RM-ODP.

This Recommendation | International Standard covers both overview and detailed explanation, and can be consulted in various ways when reading the standards:

a) if you intend to read only this Recommendation | International Standard, to gain a general understanding of the importance of ODP to your organization, concentrate on clause 6;

b) if you intend to study the whole RM-ODP, you should also read clause 6 before moving on to ITU-T Rec. X.902 | ISO/IEC 10746-2 and ITU-T Rec. X.903 | ISO/IEC 10746-3;

c) as you read ITU-T Rec. X.902 | ISO/IEC 10746-2 and ITU-T Rec. X.903 | ISO/IEC 10746-3 you may wish to consult clauses 7 to 10, which give supporting explanation for the various concepts that these common texts define;

d) when you have completed a first reading of ITU-T Rec. X.902 | ISO/IEC 10746-2 and ITU-T Rec. X.903 | ISO/IEC 10746-3, read clauses 11 and 12 which discuss the use of standards in ODP system specifications, and provide some examples of applying the ODP concepts in the specification of systems.

2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendation and International Standard listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

2.1 Identical Recommendations | International Standards

- ITU-T Recommendation X.200 (1994) | ISO/IEC 7498-1:1994, *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*.
- ITU-T Recommendation X.207 (1993) | ISO/IEC 9545:1994, *Information technology – Open Systems Interconnection – Application Layer structure*.
- ITU-T Recommendation X.720 (1993) | ISO/IEC 10165-1:1993, *Information technology – Open Systems Interconnection – Structure of management information: Management Information Model*.

- ITU-T Recommendation X.902 (1995) | ISO/IEC 10746-2:1996, *Information technology – Open distributed processing – Reference Model: Foundations*.
- ITU-T Recommendation X.903 (1995) | ISO/IEC 10746-3:1996, *Information technology – Open distributed processing – Reference Model: Architecture.*
- ITU-T Recommendation X.904 (1997) | ISO/IEC 10746-4:1998, *Information technology – Open distributed processing – Reference Model: Architectural semantics.*

2.2 Paired Recommendations | International Standards equivalent in technical content

- ITU-T Recommendation X.290 (1995), *OSI conformance testing methodology and framework for protocol Recommendations for ITU-T applications – General concepts*.
- ISO/IEC 9646-1:1994, *Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 1: General concepts.*

2.3 International Standards

- ISO/IEC 11578-2[1)]: *Information technology – Open Systems Interconnection – Remote Procedure Call (RPC) – Part 2: Interface Definition Notation.*
- ISO/IEC TR 10000-1:1995, *Information technology – Framework and taxonomy of International Standardized Profiles – Part 1: General principles and documentation framework*.

3 Definitions

3.1 Definitions in this Recommendation | International Standard

There are no definitions in this Recommendation | International Standard.

3.2 Definitions from other Recommendations | International Standards

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.902 | ISO/IEC 10746-2:

- Abstraction;
- Action;
- Action template;
- Activity;
- Architecture;
- Atomicity;
- Base class;
- Behaviour (of an object);
- Behavioural compatibility;
- Binding;
- Binding behaviour;
- Chain (of actions);
- Class;
- Client object;
- Communication;
- Compliance;
- Composite object;
- Composition;
- Configuration (of objects);
- Conformance points;

1) To be published.

- Consumer object;
- Contract;
- Contractual context;
- Creation;
- Data;
- Decomposition;
- Deletion;
- Derived class;
- Distribution transparency;
- Entity;
- Environment (of an object);
- Environment contract;
- Error;
- Establishing behaviour;
- Failure;
- Fault;
- Identifier;
- Information;
- Initiating object;
- Instance;
- Instantiation;
- Interaction;
- Interface;
- Interface signature;
- Internal action;
- Interworking reference point;
- Introduction (of an <X>);
- Invariant;
- Liaison;
- Location in space;
- Management information;
- Name;
- Name resolution;
- Naming domain;
- Notification;
- Object;
- Obligation;
- ODP standards;
- ODP system;
- Perceptual reference point;
- Permission;
- Persistence;
- Policy;
- Portability;
- Producer object;
- Programmatic reference point;
- Prohibition;
- Quality of service;

- Reference point;
- Refinement;
- Responding object;
- Role;
- Server object;
- State;
- Subclass;
- Subtype;
- System;
- Template class;
- Template type;
- Terminating behaviour;
- Thread;
- Trading;
- Type;
- Unbinding;
- Viewpoint.

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.903 | ISO/IEC 10746-3:

- <Viewpoint> language;
- Access control information;
- Access transparency;
- Announcement;
- Basic engineering object;
- Binder;
- Binding object;
- Capsule;
- Capsule manager;
- Channel;
- Checkpoint;
- Checkpointing;
- Cluster manager;
- Cluster template;
- Communication interface;
- Community;
- Compound binding action;
- Computational viewpoint;
- Deactivation;
- Dynamic schema;
- Engineering viewpoint;
- Enterprise viewpoint;
- Explicit binding;
- Failure transparency;
- Federation;
- Flow;
- Hide;
- Implicit binding;
- Information viewpoint;

- Interceptor;
- Interrogation;
- Invariant schema;
- Invocation;
- Location transparency;
- Migration;
- Migration transparency;
- Node;
- Nucleus;
- ODP function;
- Operation interface;
- Operation interface signature;
- Persistence transparency;
- Primitive binding actions;
- Protocol object;
- Reactivation;
- Recovery;
- Relocation transparency;
- Relocator;
- Replication schema;
- Replication transparency;
- Security authority;
- Security domain;
- Security policy;
- Signal;
- Signal interface;
- Signal interface signature;
- Static schema;
- Stream interface;
- Stream interface signature;
- Stub;
- Target;
- Technology viewpoint;
- Termination;
- Transaction transparency;
- Validate.

This Recommendation | International Standard makes use of the following terms defined in ISO/IEC 9646:

- Implementation conformance statement;
- Implementation Extra Information for Testing;
- Point of Control and Observation.

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.200 | ISO/IEC 7498-1:

- Open System;
- Abstract syntax;
- Transfer syntax.

4 Abbreviations

For the purposes of this Recommendation | International Standard, the following abbreviations apply:

A-profile	Application profile
ACID	Atomicity Consistency Isolation Durability
AE(I)	Application Entity (Invocation)
ALS	Application Layer Structure
AP(I)	Application Process (Invocation)
API	Application Program Interface
ASO	Application Service Object
BEO	Basic Engineering Object
CAD	Computer Aided Design
CD	Compact Disk
CIM	Computer Integrated Manufacturing
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
DL	Definition Language
F-profile	Format and presentation profile
FDT	Formal Description Techniques
GUI	Graphical User Interface
HCI	Human Computer Interface
HDTV	High Definition TV
ICS	Implementation Conformance Statement
IDL	Interface Definition Language
IT	Information Technology
IXIT	Implementation Extra Information for Testing
MIM	Management Information Model
MMC(S)	Multimedia Conferencing (System)
ODP	Open Distributed Processing
OMG	Object Management Group
OMT	Object Modelling Technique
OSE	Open System Environment
OSF	Open Software Foundation
OSI	Open Systems Interconnection
PCO	Point of Control and Observation
QOS	Quality of Service
RDA	Remote Database Access
RM-ODP	Reference Model of Open Distributed Processing
RPC	Remote Procedure Call
T-profile	Transfer profile
TINA	Telecommunication Information Networking Architecture
ULA	Upper Layers Architecture

5 Conventions

The following conventions are specific to this Recommendation | International Standard:

1) The first use in clauses 7 and 8 of formal terms from ITU-T Rec. X.902 | ISO/IEC 10746-2 and ITU-T Rec. X.903 | ISO/IEC 10746-3 is italicized.

2) Examples in clause 12 use OMT drawing conventions defined in [Rumbaugh 91].

3) In diagrams:

   - Objects are represented as ovals or circles.

   - The symbol "⊥" protruding from an object represents an interface.

6 ODP standardization

6.1 Objectives and motivation

The objective of ODP standardization is the development of standards that allow the benefits of distributing information processing services to be realized in an environment of heterogeneous IT resources and multiple organizational domains. These standards address constraints on system specification and the provision of a system infrastructure that accommodate difficulties inherent in the design and programming of distributed systems.

Distributed systems are important because there is a growing need to interconnect information processing systems. This need arises because of organizational trends such as downsizing, which demand the exchange of information both between groups within an organization and between cooperating organizations. Advances in technology are making it possible to respond to these trends by giving increasing importance to information service networks and personal workstations, and by permitting the construction of applications distributed across large configurations of interconnected systems.

In order both to manage system distribution and to exploit it (e.g. use the potential for availability, performance, dependability and cost optimization), organizations must deal with a number of key characteristics of system distribution:

- **Remoteness**: Components of a distributed system may be spread across space; interactions may be either local or remote.

- **Concurrency**: Any component of a distributed system can execute in parallel with any other components.

- **Lack of global state**: The global state of a distributed system cannot be precisely determined.

- **Partial failures**: Any component of a distributed system may fail independently of any other components.

- **Asynchrony**: Communication and processing activities are not driven by a single global clock. Related changes in a distributed system cannot be assumed to take place at a single instant.

- **Heterogeneity**: There is no guarantee that components of a distributed system are built using the same technology and the set of various technologies will certainly change over time. Heterogeneity appears in many places: hardware, operating systems, communication networks and protocols, programming languages, applications, etc.

- **Autonomy**: A distributed system can be spread over a number of autonomous management or control authorities, with no single point of control. The degree of autonomy specifies the extent to which processing resources and associated devices (printers, storage devices, graphical displays, audio devices, etc.) are under the control of separate organizational entities.

- **Evolution**: During its working life, a distributed system generally has to face many changes which are motivated by technical progress enabling better performance at a better price, by strategic decisions about new goals, and by new types of applications.

- **Mobility**: The sources of information, processing nodes, and users may be physically mobile. Programs and data may also be moved between nodes, e.g. in order to cope with physical mobility or to optimize performance.

Building such systems is not easy. It requires an architecture and, because a single engineering solution will not meet all requirements, it must be a flexible architecture. Moreover, since a single vendor will not have all of the answers, it is essential that the architecture, and any functions necessary to implement the architecture, be defined in a set of standards, so that multiple vendors can collaborate in the provision of distributed systems. Such standards will enable systems to be built that:

- Are **open** – Providing both portability (execution of components on different processing nodes without modification) and interworking (meaningful interactions between components, possibly residing in different systems).
- Are **integrated** – Incorporating various systems and resources into a whole without costly ad-hoc developments. This may involve systems with different architectures, and different resources with different performance. Integration helps to deal with heterogeneity.
- Are **flexible** – Capable both of evolving and of accommodating the existence and continued operation of legacy systems. An open distributed system should be capable of facing run-time changes – for example, it should be capable of being dynamically reconfigured to accommodate changing circumstances. Flexibility helps to deal with mobility.
- Are **modular** – Allowing parts of a system to be autonomous, but interrelated. Modularity is the basis for flexibility.
- Can be **federated** – Allowing a system to be combined with systems from different administrative or technical domains to achieve a single objective.
- Are **manageable** – Allowing the resources of a system to be monitored, controlled and managed in order to support configuration, QOS and accounting policies.
- Meet **quality of service** needs – Covering, for example, provision of timeliness, availability and reliability in the context of remote resources and interactions, together with provision of fault tolerance that allows the remainder of a distributed system to continue to operate in the event of failure of some part. Provision of fault tolerance (and of dependability in general) is necessary within large distributed systems where it is unlikely that all parts of the system will ever be operational simultaneously.
- Are **secure** – Ensuring that system facilities and data are protected against unauthorized access. Security requirements are made more difficult to meet by remoteness of interactions, and mobility of parts of the system and of the system users.
- Offer **transparency** – Masking from applications the details and the differences in mechanisms used to overcome problems caused by distribution. This is a central requirement arising from the need to facilitate the construction of distributed applications. Aspects of distribution which should be masked (totally or partially) include: heterogeneity of supporting software and hardware, location and mobility of components, and mechanisms to achieve the required level for QOS in the face of failures (e.g. replication, migration, checkpointing, etc.).

6.2 Realization

ODP standardization has four fundamental elements:

- an object modelling approach to system specification;
- the specification of a system in terms of separate but interrelated viewpoint specifications;
- the definition of a system infrastructure providing distribution transparencies for system applications;
- a framework for assessing system conformance.

6.2.1 Object modelling

Object modelling provides a formalization of well-established design practices of abstraction and encapsulation. Abstraction allows the description of system functionality to be separated from details of system implementation. Encapsulation allows the hiding of heterogeneity, the localization of failure, the implementation of security and the hiding of the mechanisms of service provision from the service user.

The object modelling concepts cover:

- Basic modelling concepts – Providing rigorous definitions of a minimum set of concepts (action, object. interaction and interface) that form the basis for ODP system descriptions and are applicable in all viewpoints.

- Specification concepts – Addressing notions such as type and class that are necessary for reasoning about specifications and the relations between specifications, provide general tools for design, and establish requirements on specification languages.
- Structuring concepts – Building on the basic modelling concepts and the specification concepts to address recurrent structures in distributed systems, and cover such concerns as policy, naming, behaviour, dependability and communication.

6.2.2 Viewpoint specifications

A viewpoint (on a system) is an abstraction that yields a specification of the whole system related to a particular set of concerns. Five viewpoints have been chosen to be both simple and complete, covering all the domains of architectural design. These five viewpoints are:

- the enterprise viewpoint, which is concerned with the purpose, scope and policies governing the activities of the specified system within the organization of which it is a part;
- the information viewpoint, which is concerned with the kinds of information handled by the system and constraints on the use and interpretation of that information;
- the computational viewpoint, which is concerned with the functional decomposition of the system into a set of objects that interact at interfaces – enabling system distribution;
- the engineering viewpoint, which is concerned with the infrastructure required to support system distribution;
- the technology viewpoint, which is concerned with the choice of technology to support system distribution.

For each viewpoint there is an associated viewpoint language which can be used to express a specification of the system from that viewpoint. The object modelling concepts give a common basis for the viewpoint languages and make it possible to identify relationships between the different viewpoint specifications and to assert correspondences between the representations of the system in different viewpoints.

6.2.3 Distribution transparency

Distribution transparencies enable complexities associated with system distribution to be hidden from applications where they are irrelevant to their purpose. For example:

- access transparency masks differences of data representation and invocation mechanisms for services between systems;
- location transparency masks the need for an application to have information about location in order to invoke a service;
- relocation transparency masks the relocation of a service from applications using it;
- replication transparency masks the fact that multiple copies of a service may be provided in order to provide reliability and availability.

ODP standards define functions and structures to realize distribution transparencies. However, there are performance and cost tradeoffs associated with each transparency and only selected transparencies will be relevant in many cases. Thus, a conforming ODP system must implement those transparencies that it supports in accordance with the relevant standards, but it is not required to support all transparencies.

6.2.4 Conformance

The basic characteristics of heterogeneity and evolution imply that different parts of a distributed system can be purchased separately, from different vendors. It is therefore very important that the behaviours of the different parts of a system be clearly defined, and that it be possible to assign responsibility for any failure to meet the system's specifications.

The framework defined to govern the assessment of conformance addresses these issues. It covers:

- identification of the conformance points within the set of viewpoint specifications at which observations of conformance can be made;
- definition of classes of conformance point;
- specification of the nature of conformance statements to be made in each viewpoint and the relation between them.

6.3 Standards

6.3.1 The Reference Model

The RM-ODP provides the overall framework for ODP standardization. It comprises two main parts:

- ITU-T Rec. X.902 | ISO/IEC 10746-2: **Foundations**, which defines the concepts and analytical framework for the description of distributed processing systems, including a general framework for the assessment of conformance;
- ITU-T Rec. X.903 | ISO/IEC 10746-3: **Architecture,** which defines how ODP systems are specified and the infrastructure providing distribution transparencies.

ITU-T Rec. X.904 | ISO 10746-4: **Architectural semantics** complements these two main parts by providing a formal interpretation of the modelling concepts and viewpoint languages in terms of existing formal description techniques.

The RM-ODP is generic, that is, independent of, and equally applicable to, arbitrary application domains making use of or requiring distributed systems technology. For some specific application domains it will be necessary to refine and specialize the RM-ODP to suit particular needs, resulting in:

- specific reference models which cover individual types of enterprise, use concepts and common functions given in the RM-ODP, and define additional conceptual detail and specific functions, e.g. Telecommunication Information Networking Architecture (TINA);
- standards for the realization of specific functions needed for particular applications and possibly identified in a specific reference model, e.g. interfaces for telephone call connection.

Because it is generic, the RM-ODP also enables disparate distributed system technologies to be integrated into cost effective technical system solutions to business requirements. In particular, in the case of the architectures published by the Open Software Foundation (OSF) and the Object Management Group (OMG) to explain how the functions they specify to support distributed systems fit together, the ODP approach adds value by addressing such issues as federation, transparency and system management, and by defining a fine grained framework of reference points to support the integration of functions from different sources.

6.3.2 Specific standards

Four categories of standards are identified within the overall framework provided by the RM-ODP:

- additional architectural frameworks, which complement the RM-ODP in specific areas such as naming, security and conformance assessment;
- notation standards, which define notations for expressing specifications of different aspects of system integration and distribution, and rules for relating different specifications;
- component standards, which define a single ODP function or closely interrelated set of ODP functions, possibly capable of implementation as a single hardware or software platform;
- component composition standards, which define the coordinated use of a number of components to achieve some objective of the system as a whole, such as provision of a specific transparency.

NOTE – Some standards may specify both components and their composition (so that a useful facility can be implemented directly). Other standards may form the basis for a number of component composition standards, for example, an ODP Relocator standard would be referenced in component composition standards for the provision of location or migration transparencies.

The RM-ODP provides a framework for component standards and component composition standards for ODP functions which permits a number of different approaches to their realization. This flexibility is necessary if the framework is to have a reasonable lifetime, incorporating new developments as they mature. Thus, a specific standard or set of standards, specifies one particular solution to the provision of some ODP requirement, making all the specific choices needed for implementation of open products to be possible, and there may be a number of such standards, corresponding to different design choices. In time, new technologies will be incorporated, leading to new generations of standards within the one ODP framework.

7 Foundations

ITU-T Rec. X.902 | ISO/IEC 10746-2 defines a set of modelling concepts which provide the foundation for expressing the *architecture* of *ODP systems* defined in ITU-T Rec. X.903 | ISO/IEC 10746-3. These concepts fall into three categories:

- basic modelling concepts which introduce a general object-based model. In general, an ODP system can be described as a collection of related, interacting *objects*;

- specification concepts which are not intrinsic to distributed systems but which allow their user to describe and reason about ODP system specifications. They place requirements on any specification language that is used for the specification of an ODP system and, because they are essentially language independent, can be applied in any given specification language or programming language;

- structuring concepts, covering organization, the properties of systems and objects, *policy*, *naming*, *behaviour* and management, that correspond to notions and structures that are generally applicable in the design and description of distributed systems.

ITU-T Rec. X.902 | ISO/IEC 10746-2 also provides a general framework for understanding *conformance* in ODP systems, expressed in terms of the general object model. This framework is discussed in clause 9.

7.1 Basic modelling concepts

7.1.1 Objects

ODP system specifications are expressed in terms of objects. An object is a representation of an *entity* in the real world. It contains *information* and offers services. A system is composed of interacting objects. An object is characterized by that which makes it distinct from other objects and by *encapsulation*, *abstraction* and behaviour.

Encapsulation is the property that the information contained in an object is accessible only through *interaction*s at the *interfaces* supported by the object. Because objects are encapsulated, there are no hidden side effects of interactions. That is, an interaction with one object cannot affect the *state* of another object without some secondary interaction with that object taking place. Thus, any change in the state of an object can only occur as a result of an *internal action* of the object or as a result of an interaction of the object with its *environment*.

Abstraction implies that the internal details of an object are hidden from other objects and is crucial for dealing with heterogeneity, permitting different services to be implemented in different ways, using different mechanisms and technologies, enabling *portability* and interoperability.

Abstraction also builds a strong separation between objects, enabling them to be replaced or modified without changing their environment, provided they continue to support the services their environment expects (i.e. they are backward compatible). This approach to extensibility is essential in large, heterogeneous, distributed environments, which by their very nature are continuously evolving. An object model provides modularity and the ability to compose new modules from existing modules: these are capabilities important for building flexible systems and encourage reuse to enhance productivity.

The ODP object model is general and makes a minimum number of assumptions. For instance:

- objects can be of an arbitrary granularity (e.g. they can be as large as the telephone network, or as small as an integer);

- objects can exhibit arbitrary (encapsulated) behaviours, and have an arbitrary level of internal parallelism;

- interactions between objects are not constrained and can include, for example, asynchronous and multiway-synchronous interactions.

7.1.2 Interfaces and interaction points

Objects can only interact at interfaces, where an interface represents a part of the object's behaviour related to a particular subset of its possible interactions. Each interface is identified with a set of interactions in which the object can participate. Note that these interactions do not necessarily occur with other objects: an object can interact with itself. An important characteristic of the concept of object in the RM-ODP is that an object can have a number of interfaces. The motivations for considering multiple interfaces are functional separation and distribution. Functional separation can be understood, for example, in the context of systems management, where management interactions and non-management interactions are normally separated into different interfaces. When dealing with distributed objects, separation is also necessary where interfaces constitute points of access to the object that are situated at different locations in space.

As a consequence of defining a number of interfaces for an object, the interactions at any one of the interfaces may be affected by the interactions at other interfaces, and are not necessarily determined in isolation.

An interface exists at an *interaction point* which, at any point in time, is associated with some point in space. A number of interfaces may exist at a given interaction point and the interaction point may be mobile. The significance of points in space and time, and the way in which they are expressed, depends on the language in which a specification is expressed.

7.1.3 Behaviour and state

A behaviour of an object is a collection of *actions* that the object may take part in, together with the set of constraints on when those actions can occur. The object model does not constrain the form or nature of object behaviour. The actions can be interactions of the object with its environment or internal actions of the object.

State and behaviour are interrelated concepts. The state of an object is the condition of the object at a given instant that determines the potential future sequences of actions that object may be involved in. At the same time, actions bring about state changes and, hence, the current state of an object is partly determined by its past behaviour. Of course, the actions the object will actually undertake are not entirely determined by its present state; they will also depend on which actions the environment is prepared to participate in.

7.2 Specification concepts

7.2.1 Composition/Decomposition

Composition and *decomposition* can be used to organize the specification of a distributed system as a set of specifications, each one dealing with a different level of abstraction. It permits the specification of a complex distributed system to be decomposed into specifications of a number of simpler objects which may also be decomposed at a lower level of abstraction.

The processes of composition and decomposition provide for a hierarchical specification of a distributed application. In this composition hierarchy, object *classes* at higher levels are assembled from *configurations* of component object classes at lower levels. Thus, composition is a powerful modelling concept in that it permits a subsystem to be treated as a single higher-level object.

As an example, the diagram in Figure 1 shows the composition of two objects, C and D. In order to compose C with D, the behaviour of C must be defined so that it interacts appropriately with D. The interface between C and D may be as simple as a single interaction (for example, to pass information from C to D), or it may be a more complicated behaviour (such as a sequence of interactions where C invokes an operation which returns a result). If C and D are composed into an object, then interactions between C and D are hidden and become internal actions of the *composite object*. The left interface represents an interface of the composite object.

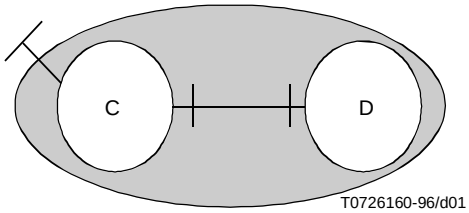


Figure 1 – Object composition

Object composition yields composition of states and behaviours and it is therefore possible to speak of a composite behaviour and of a composite state.

Composition hierarchy is orthogonal to the *subclass* hierarchy discussed in 7.2.3, and should not be confused with it. In general there is no subclass hierarchy relationship between classes of the component objects and the classes of the composite object. In fact, examples can be found where the composite object belongs to a subclass of the class to which

a component object belongs (e.g. where the objects represent communications services, with the composite object adding value to one of its components).

7.2.2 Behavioural compatibility

One object is said to be behaviourally compatible with another object in some environment if the first object can replace the second, without the environment being able to detect any difference. Any particular interpretation of *behavioural compatibility* will impose constraints on the allowed behaviour of the environment. A common approach is to assume that the environment behaves as a tester for the original object. That is, the environment should be capable of fully exercising the original behaviour but should be able to do no more. Such assumptions are essential if behavioural compatibility in the context of some unknown environment is to be considered.

7.2.3 Type and class

A *type* is a predicate (that is, it is a property or set of properties) of a collection of things (objects, interfaces, etc.). For example “is red” is a type. We say that something satisfies a type, or is of the type, if the predicate holds for the thing. Things can be quite dissimilar and still satisfy the same type; they only need to possess the properties prescribed by the type. For instance, a particular flag, a particular brick house and a particular sports car might all be red.

Types implicitly classify things into sets known as classes, where a class is the collection of things with the properties prescribed by a type.

The notion of type is very general and can be specialized in various ways. It is useful in any context where it is necessary to talk about, reason about and verify properties of things (e. g. for *trading*, for *binding*).

The concepts of type and class yield natural class/subclass and type/subtype hierarchies. The class/subclass distinction corresponds to the intuitive distinction in set theory between sets and subsets. One class is a subclass of another if, and only if, the former is a subset of the latter. One type is a subtype of another if the predicates of the first type imply the predicates of the second type.

Subclassing and subtyping go hand in hand. For every type there is an associated class (which may, of course, be empty). So if we have two types T1 and T2, then there must be associated classes C1 and C2. T1 is a subtype of T2 exactly when C1 is a subclass of C2.

7.2.4 Templates

A *template* describes a collection of things (objects, interfaces etc.) in sufficient detail for a new thing to be instantiated from it.

Where a template describes a set of objects, it describes features such as state parameters, operations and behaviour. Typically, *instantiation* of an object entails establishing the initial state, for example, a buffer object might be created with empty contents.

The concept of behavioural compatibility also applies to object templates in the sense that there is behavioural compatibility between two object templates if the objects instantiated from those templates are behaviourally compatible.

A *template type* is a predicate defined in a template. A template type is satisfied by all instantiations from the template, and, in general, can be satisfied by other things, where they fulfill the same requirements as the instantiations. For example, a template type may be defined so that objects instantiated from different templates, but satisfying that template type, show behavioural compatibility.

Each template type gives rise to a *template class*: the set of *instances* of the template type. The template classes can be organized into subclass hierarchies, in accordance with the type/subtype relationships between template types.

7.2.5 Roles

A *role* identifies, in a template for a composite object, a behaviour to be associated with one of the component objects.

A role may correspond to a subset of the total behaviour of a component object. When an object is viewed in terms of a role, only a named subset of its actions is of interest, and other actions are abstracted away, possibly to other roles. A component object may have several roles at a given time depending upon its interactions, and may take different roles at different times. These roles may be associated with interfaces.

For example, an object may have its normal functional or mission role (i.e. its purpose), and, for management purposes, have a management role (i.e. the behaviour needed to monitor and control the mission role behaviour). Each role has its

own interface, where the mission role is associated with a mission interface and the management role is associated with a management interface.

7.2.6 Base classes and derived classes

The concepts of *base class* and *derived class* are based on a general notion of modification of templates known as *incremental modification*. Incremental modification is the derivation of a new template through the modification of an existing template. The new template is called the derived template and the original template is called the base template. Instances of the original template and the derived template are called the base class and the derived class, respectively.

In general, incremental modification which allows replacement may yield a different hierarchy than the class/subclass one.

For instance, consider a template class C1 of red cars defined by a template Temp 1 containing the following line:

COLOUR=RED.

Suppose, to give template class C2, this line is replaced by:

COLOUR=BLUE.

The template class C2 is derived from C1 and is not a subclass of C1.

In some cases, the implementation of a derived class may be based on the implementation of the base class. This concept is known as implementation inheritance, and enables sharing of code in executable programs. However, this can cause problems in a distributed environment, since changes to the base class code must be propagated to update all the derived class implementations, and therefore ODP systems are not required to support implementation inheritance.

7.3 Structuring concepts

7.3.1 Groups and domains

A *group* is a set of objects grouped together for structural reasons or because the behaviours of the objects have common features (e.g. in a replication group they can replace each other, in a communicating group they participate in the same interaction). The group concept is generic and allows the specification of different kinds of group that can be used in distributed systems for many different purposes, such as *fault* tolerance, availability and application support (e.g. in conferencing applications).

A *domain* is a particular form of group in which a particular aspect of the behaviour of objects in the group is controlled by the same authority. For example, in a *security domain* the security policies that apply to the behaviour of the objects in the domain are set by the same *security authority*. The domain concept allows the notions of autonomy, authority and control to be introduced into distributed systems. The domain concept covers many different needs since distributed systems use many kinds of domains (e.g. security domains, management domains, *naming domains*).

7.3.2 Naming

Naming is necessary to distinguish and access components of a distributed system and is, therefore, a fundamental element in distributed system construction.

Context dependent naming and *name* management is necessary in order to deal with heterogeneity, autonomy and *federation*. It brings flexibility and evolution by allowing independent development of name systems and arbitrary combinations of name systems, including existing, independent ones. It also allows heterogeneity in name systems at several levels, e.g. forms of names, name assignment, naming policies and strategies for *name resolution* can be different in different systems.

Names are used to refer to entities in a given context. There may be circumstances when a name refers to more than one entity. A name that refers unambiguously to one entity is called an *identifier*.

The naming concepts in ITU-T Rec. X.902 | ISO/IEC 10746-2 do not specify a full naming framework for ODP. Such a framework is a subject for separate standardization.

7.3.3 Contract

A *contract* is an agreement that governs cooperation among a number of objects, and embodies the ideas of *obligation*, *permission*, *prohibition* and *expectation* associated with cooperating objects. Thus, it is a general concept for characterizing and regulating the cooperation of objects.

Whenever objects cooperate (interact), there is some contract between them. In cases where the contract may be agreed at some time and later terminated, it is a dynamic specification of the configuration of the objects. Though the potential cooperation between the objects is constant, the rules provided by the contract constrain the potential cooperation to some current, transitory behaviour.

However, contracts are often derivable from the *viewpoint language* rules and do not need to be stated explicitly. For example, in the case of the computational language, a client is obliged not to invoke an operation which is not defined in a server's interface. Only contracts which specify additional constraints need to be expressed explicitly.

As an example, a contract can specify:

- the roles of objects and the obligations applying to roles, i.e. the expected cooperative behaviour;
- the *Quality of Service* aspects of object cooperation (issues of dependability, correctness, etc.);
- the kind of behaviour that *invalidates* the contract.

An *environment contract* is a particular kind of contract that applies between an object and its environment. This contract describes the requirements placed by the object on its environment and vice versa. In particular, it is concerned with Quality of Service (QOS) constraints.

7.3.4 Liaison and binding

Binding behaviour establishes a *contractual context* (a binding) between interfaces and enables object cooperation. A binding can exist at several levels of abstraction.

A *liaison* is the relationship that exists between the objects cooperating under the auspices of a binding. When the liaison is in place, an object knows that the other objects in the liaison obey the contract. An object can be involved in several simultaneous liaisons: for each of these liaisons there is a corresponding contract.

NOTE – The following terms, used in the example in the next paragraph, belong to the OSI terminology: Bind, Bind Response, Unbind, and Application Context.

An example of liaison establishment is provided by the OSI Association binding behaviour. The *establishing behaviour* is provided by a Bind operation which includes the sending of contract parameters by the initiating application object to the responding application object. The *responding object* responds with a Bind Response operation (with possibly different contract parameters), thereby establishing the liaison. The contractual context for the liaison is given by the Application Context agreed through the Bind and the Bind Response exchange. When either party wishes to terminate the liaison, it initiates the *terminating behaviour* by invoking the Unbind operation.

8 Architecture

ITU-T Rec. X.903 | ISO/IEC 10746-3 makes the prescriptive statements which must hold for a system to be characterized as an ODP system. Using the concepts and terminology defined by ITU-T Rec. X.902 | ISO/IEC 10746-2, it defines:

- an architectural framework for structuring the specification of ODP systems in terms of the concepts of *viewpoints* and *viewpoint specifications*, and *distribution transparencies*;
- a set of *languages* in terms of which the different viewpoint specifications can be expressed;
- a system infrastructure providing *distribution transparencies* for system applications.

8.1 Architectural framework

Distributed systems can be very large and complex, and the many different considerations which influence their design can result in a substantial body of specification, which needs to be given structure if it is to be managed successfully. A good framework should allow different parts of the design to be worked on separately if they are independent, but should identify clearly those places where different aspects of the design constrain one another. In order to achieve this, two main structuring approaches are used in the ODP architecture: the definition of viewpoints and the definition of transparencies.

8.1.1 Viewpoints

A viewpoint is a subdivision of the specification of a complete system, established to bring together those particular pieces of information relevant to some particular area of concern during the design of the system. An ODP system is anything of interest, so that, for example, it may equally well be an information processing system of an organization, or

a particular component (hardware or software) of such a system. The viewpoints are not completely independent: key items in each are identified as related to items in the other viewpoints. However, the viewpoints are sufficiently independent to simplify reasoning about the complete specification.

Each of the viewpoints in the set can be related to all the others. They do not form a fixed sequence like a set of protocol layers, nor are they created in a fixed order according to some design methodology. The architecture is expressed in terms of the complete set of related viewpoints, without laying down how a complete specification is to be constructed for any given system.

The RM-ODP defines five viewpoints. These are:

a) The *enterprise viewpoint*: A viewpoint on the system and its environment that focuses on the purpose, scope and policies for the system.

b) The *information viewpoint*: A viewpoint on the system and its environment that focuses on the semantics of the information and information processing performed.

c) The *computational viewpoint*: A viewpoint on the system and its environment that enables distribution through functional decomposition of the system into objects which interact at interfaces.

d) The *engineering viewpoint*: A viewpoint on the system and its environment that focuses on the mechanisms and functions required to support distributed interaction between objects in the system.

e) The *technology viewpoint*: A viewpoint on the system and its environment that focuses on the choice of technology in that system.

In order to represent an ODP system from a particular viewpoint, it is necessary to define a structured set of concepts in terms of which that representation (or specification) can be expressed. This set of concepts provides a language for writing specifications of systems from that viewpoint, and such a specification constitutes a model of a system in terms of the concepts. The terms of each *viewpoint language*, and the rules applying to the use of those terms, are defined using the object modelling concepts defined in ITU-T Rec. X.902 | ISO/IEC 10746-2. Each language has sufficient expressive power to specify an *ODP function*, application or policy from the corresponding viewpoint. Where system specifications conform to these languages, the systems designed are ODP systems, at least from an architectural point of view.

In the RM-ODP, the various viewpoint languages differ in the strengths of the constraints their use implies. Those viewpoint languages concerned with organizing distribution and providing common solutions to its problems (the computational and engineering viewpoints) place a significant number of constraints that must be observed, and in so doing, give guarantees of interworking between, and portability of, components. On the other hand, since the RM-ODP is generic, there are few rules to be stated in the enterprise and information languages, and these are limited to a set of basic concepts and guidance about the scope of enterprise and information modelling for distributed systems.

More extensive constraints will be defined for the enterprise and information languages when these are used in the specification of systems within particular fields of application and specific enterprises. For example, for any Trader system the enterprise and information specifications are constrained by the provisions of the Trader standard.

8.1.2 Distribution transparencies

When designing a distributed system, a number of concerns become apparent which are a direct result of the distribution: the system components are heterogeneous, they can fail independently, they are at different and, possibly, varying locations, and so on. These concerns can either be solved directly as part of the application design, or standard solutions can be selected, based on best practice.

If standard mechanisms are chosen, the application designer works in a world which is transparent to that particular concern; the standard mechanism is said to provide a *distribution transparency*. Application designers simply select which distribution transparencies they wish to assume and where in the design they are to apply.

The distribution transparency approach can lead directly to software reuse. Selection of distribution transparencies in the system specification can lead to the automatic incorporation of well established implementations of the standard solutions by the system building tools in use, such as compilers, linkers and configuration managers. The designer expresses system requirements in the form of a simplified statement of the system required and the distribution transparency properties that it should possess.

The distribution transparencies defined in the RM-ODP are:

a) *access transparency*, which masks differences in *data* representation and *invocation* mechanisms to enable interworking between objects. This distribution transparency solves many of the problems of interworking between heterogeneous systems, and will generally be provided by default.

b) *failure transparency*, which masks from an object the failure and possible *recovery* of other objects (or itself) to enable fault tolerance. When this distribution transparency is provided, the designer can work in an idealized world in which the corresponding class of failures does not occur.

c) *location transparency*, which masks the use of information about *location in space* when identifying and binding to interfaces. This distribution transparency provides a logical view of naming, independent of actual physical location.

d) *migration transparency*, which masks from an object the ability of a system to change the location of that object. Migration is often used to achieve load balancing and reduce latency.

e) *relocation transparency*, which masks relocation of an interface from other interfaces bound to it. Relocation allows system operation to continue even when migration or replacement of some objects creates temporary inconsistencies in the view seen by their users.

f) *replication transparency*, which masks the use of a group of mutually behaviourally compatible objects to support an interface. Replication is often used to enhance performance and availability.

g) *persistence transparency*, which masks from an object the *deactivation* and *reactivation* of other objects (or itself). Deactivation and reactivation are often used to maintain the *persistence* of an object when the system is unable to provide it with processing, storage and communication functions continuously.

h) *transaction transparency*, which masks coordination of activities amongst a configuration of objects to achieve consistency.

In any system specification, the definition of the transparency involves both a set of requirements and a distribution transparency that satisfies it. The set of requirements states where the distribution transparency is needed (i.e. which interactions it affects). This may simply be a statement that it applies throughout a system, or may be a more selective statement involving specific interfaces and defining, for example, the interactions which make up a transaction or selecting the objects and interfaces to be supported by replication. The solution takes the form of a set of rules for transforming the specification of the distribution transparency requested into a specification in which selected interactions or objects are expanded to include mechanisms which provide that transparency.

8.2 Enterprise language

The enterprise language introduces basic concepts necessary to represent an ODP system in the context of the enterprise in which it operates. The aim of an enterprise specification is to express the objectives and policy constraints on the system of interest. In order to do this, the system is represented by one or more enterprise objects within a *community* of enterprise objects that represents the enterprise, and by the roles in which these objects are involved. These roles represent, for example, the users, owners and providers of information processed by the system. Creating a separate viewpoint to convey this information decouples the specification of the objectives set for a system from the way in which that system is to be realized.

One of the key ideas in the enterprise language is that of a contract, linking the performers of the various roles in a community and expressing their mutual obligations. A contract can express the common goals and responsibilities which distinguish roles in a community, such as a business and its customers or a government organization and its clients, as being related in particular ways in a single enterprise.

Where appropriate, an enterprise specification will also express aspects of ownership of resources and responsibility for payment for goods and services in order to identify, for example, constraints on accounting and security mechanisms within the infrastructure which supports the system.

One particular kind of community is a federation, which is a coming together of a number of groups answering to different authorities (and thus representable as distinct domains) in order that they may jointly cooperate to achieve some objective. Since the evolution of distributed systems will repeatedly result in the merging of existing, separately managed sub-systems to share information or support commercial interests, specification of the *creation* of federations, and expression of the rules which are to govern them, form an important part of system specification in the enterprise viewpoint.

The domains concerned in a federation may be administrative domains (each subject, for example, to particular security or management controls) or technology domains (each subject, for example, to common choices of system hardware or

software). The specification of federation involves specification of the objectives for interworking between different domains and of the policies governing that interworking.

Federation of administrative domains relates to interworking between domains in the same or different enterprises in order to provide sharing, integration or partitioning of resources and applications across different systems and locations in response to user needs. Federation of technology domains is concerned with integration of different system architectures, and of systems with different resources and different performance; it provides modularity that allows incremental growth without impacting existing applications. The two kinds of federation often coincide, since differences in administration can lead to differences in choice of technology.

Between administrative domains, either or both administrations may wish to impose their own access controls for such purposes as security, accounting, and monitoring, in addition to controls imposed by the objects themselves. Administrative boundaries are also the points where changes of management responsibility take place for such things as resource allocation and dependability guarantees.

Policies governing the operation of federation involve policies governing interworking. Thus, specification of federation can relate to the need for specification of *interceptor* facilities in the engineering description, and the objectives and policy for federation establish constraints on the provision of interceptor facilities.

An enterprise specification defines the policies governing the behaviour of the communities that it specifies. These policies determine the actions of the enterprise objects that comprise those communities, and are concerned with the placing and fulfilling of obligations (e.g. requesting delivery; making a delivery), and the permitting or forbidding of actions (e.g. authorizing or rejecting access to system facilities). Policies may relate to:

a) The structuring of the community in terms of roles and the assignment of roles to enterprise objects. For example, **community rules** may state:

   - assignment of roles and responsibilities to enterprise objects within the community;
   - how enterprise objects are related in the community structure (e.g. hierarchy or isocracy).

   An enterprise specification may also include **business rules** that express:

   - the enterprise as a business entity;
   - accounting requirements;
   - evolution of the business in order to fulfil its objectives.

b) Permitted interactions between enterprise objects holding different roles (i.e. access control). For example, **security rules** may define:

   - the role-*activity*-object relationships, and their integrity and confidentiality requirements for activities and objects;
   - the rules for detection of security threats;
   - the rules for protection against security threats;
   - the rules for limiting any damage caused by any security breaches.

c) The responsibility delegated to enterprise objects. For example, **delineation of authority rules** are used to assign:

   - privileges to enterprise objects (trust);
   - permission or prohibition of actions of enterprise objects.

d) Accounting for resource usage. For example, **resource usage rules** define the constraints that may be imposed by external:

   - regulatory bodies;
   - market demands;
   - environment,

   depending on whether the resource usage is:

   - public;
   - private;
   - third-party.

e) The ownership of resources. For example, **transfer rules** may state the exchange of ownership and/or responsibilities for resources between enterprise objects.

f) The membership of federations. For example, an enterprise specification can include **domain rules** that specify:

- the membership rules of a domain;
- the interaction rules between domains of the same type;
- the domain naming rules.

It is expected that there will be different notations for expressing enterprise specifications for specific organizational structures and business practices. The RM-ODP requires that an appropriate specification be generated, but places few constraints on the form that organizations should take.

Assessment of conformance to the enterprise specification of a system involves relating the requirements (e.g. a response time for fulfilling an obligation) expressed in the specification to sets of observations of the behaviour of the system at *conformance points* identified in the engineering and technology specification, and assessing the degree of consistency between the requirements and the observations.

8.3 Information language

The individual components of a distributed system must share a common understanding of the information they communicate when they interact, or the system will not behave as expected. Some of these items of information are handled, in one way or another, by many of the objects in the system. To ensure that the interpretation of these items is consistent, the information language defines concepts for the specification of the meaning of information stored within, and manipulated by, an ODP system, independently of the way the information processing functions themselves are to be implemented.

Information held by the ODP system about entities in the real world, including the ODP system itself, is represented in an information specification in terms of information objects, and their relationships and behaviour. Basic information elements are represented by atomic information objects. More complex information is represented as composite information objects each expressing relationships over a set of constituent information objects.

Just as in familiar data modelling, the information specification comprises a set of related schemata, namely, the *invariant*, *static* and *dynamic* schemata.

An *invariant schema* expresses relationships between information objects which must always be true, for all valid behaviour of the system. Thus, an invariant schema for a bank account might specify that the balance must always be non-negative as the bank does not offer an overdraft facility.

A *static schema* expresses assertions which must be true at a single point in time. A common use of static schemata is to specify the initial state of an information object. For example, the initial state of a bank account object consists of an account balance of $0 and the amount withdrawn on that day which is also $0. Another static schema might be used to describe how the amount withdrawn that day is $0 at midnight every night; note that this static schema makes no restriction on the account balance at that point.

A *dynamic schema* specifies how the information can evolve as the system operates. For example, a bank account would require a dynamic schema for depositing money, withdrawing money, paying interest, and charging account fees. A dynamic schema might be applicable only in certain circumstances (which could be specified by the use of a static schema). For example, the dynamic schema for withdrawing $N might specify that the account balance is decremented by $N provided that the total amount withdrawn that day does not exceed $500. No dynamic schema can specify a resultant state that violates the invariant constraint, i.e. only money in the account can be withdrawn.

In addition to describing state changes, dynamic schemata can also create and delete component objects. This allows an entire information specification of an ODP system to be modelled as a single (composite) information object.

These schemata may apply to the whole system, or they may apply to particular domains within it. Particularly in large and rapidly evolving systems, the reconciliation and federation of separate information domains will be one of the major tasks to be undertaken in order to manage information.

Schemata for composite information objects do not need to reference all components of the information object. Schemata for composite information objects can be composed from schemata for their component objects, provided such composition is meaningful. Encapsulation of information objects is related to the level of abstraction of the description concerned. Thus, at an appropriate level of abstraction, schemata for composite information objects can reference the

internals of their component objects, although at a higher level of abstraction, this may not be possible. This permits the specification of such complex noun phrases as “the phone numbers of the customers with accounts that withdrew over $400 today”.

Some elements visible from the enterprise viewpoint will be visible from the information viewpoint and vice versa. For example, an activity seen from the enterprise viewpoint may appear in the information viewpoint as the specification of some processing which causes a state transition of an information entity.

Different notations for information specifications model the properties of information in different ways. Emphasis may be placed on classification and reclassification of information types, or on the states and behaviour of information objects. In some specification languages, atomic information objects are represented as values. The approach to be taken will depend on the modelling technique and notation being used.

Assessment of conformance to the information specification of a system involves relating the requirements expressed in the specification (e.g. in an invariant schema) to sets of observations of the behaviour of the system at conformance points identified in the engineering and technology specification, and assessing the degree of consistency between the requirements and the observations.

8.4 Computational language

The computational viewpoint is directly concerned with the distribution of processing but not with the interaction mechanisms that enable distribution to occur. The computational specification decomposes the system into objects performing individual functions and interacting at well defined interfaces. It thus provides the basis for decisions on how to distribute the jobs to be done, because interfaces can be located independently, assuming communications mechanisms can be defined in the engineering specification to support the behaviour at those interfaces.

The heart of the computational language is the object model which defines:

- the form of interface an object can have;
- the way that interfaces can be bound and the forms of interaction which can take place at them;
- the actions an object can perform, in particular the creation of new objects and interfaces, and the establishment of bindings.

The computational object model provides the basis for ensuring that specification languages, programming languages and communication mechanisms all perform in a consistent way, thus allowing open interworking and portability of components.

The computational language enables the specifier to express constraints on the distribution of an application (in terms of environment contracts associated with individual interfaces and interface bindings) without specifying the actual degree of distribution. This ensures that applications contain no unstated assumptions affecting the distribution of their components. Because of this, the configuration and degree of distribution of the hardware on which ODP applications are run can easily be altered, subject to the stated environment constraints, without having a major impact on application software.

The computational language does not preclude the use in a distributed environment of software designed for centralized systems. It allows for encapsulation of existing applications as (non distributed) components of a larger, distributed application. This permits an evolutionary approach to the provision of distribution, thereby protecting investments in existing software.

Interactions between computational objects are essentially asynchronous and can take three forms:

- *operations*, that are similar to procedures, and are invoked on designated interfaces;
- *flows*, that are abstractions of continuous sequences of data between interfaces;
- *signals*, that are elementary atomic interactions.

Operations reflect the client/server paradigm. An operation is an interaction between a *client object* and a *server object* which requests (an invocation) the performance of some function by the server. There are two types of operation:

- an *interrogation*, in which the server returns a response (a *termination*) to the client request;
- an *announcement*, in which there is no response to the client request.

The notion of termination generalizes results and exceptions as found in many object-based and non-object-based programming languages.

The performance of operations is extended in space and time. Consequently, when an operation fails, the failure need not occur for all the participants and may be observed by them at different times. The ability of the client to observe and take action on failures is different for interrogations and announcements.

In the case of an interrogation, the two-way handshake ensures both that the client has confirmation that the requested function has been performed, and that, if a client *thread* of activity invokes a *chain* of interrogations, the requests are responded to by the server in the order that they were issued by the client.

In the case of an announcement, any guarantees of the performance of requests, and the order of performance, are determined by the environment contracts that apply to the operations.

Flows can be used to model, for example, the flow of audio or video information in a multimedia application or in voice-based telecommunication services, or the continuous flow of periodic sensor readings in a process control application. A flow is characterized by its name and its type, which specifies the nature and format of data exchanged. The exact semantics of flows is left undefined in the computational model. In fact there can be many different semantics for flows, depending on the application domain.

Signals are the lowest level of description of interactions between computational objects. A signal is a pairwise, atomic shared action resulting in one-way communication from an *initiating* computational object to a *responding* computational object (in this context "responding" means "accepting the communication"). This means:

- that the signal occurs at a defined point in time and, hence, is a point of reference for measurement purposes (e.g. in QOS observations);
- that a failure is identical for, and visible to, all participants.

In many cases a signal will correspond, in implementation terms, to an observable event at some physical location, however the definition of the concept does not preclude the implementation of signals through transaction mechanisms which give the necessary behaviour guarantees.

An operation or a flow can be explained in terms of a combination of several signals. An interrogation, for instance, can be understood as a sequence of signals: invocation emission (by the client object), invocation receipt (by the server object), termination emission (by the server), termination receipt (by the client). In contrast, since the exact semantics of flows is not given in the computational model, their mapping on signals is not defined. Modelling operations or flows in terms of signals becomes necessary in order to define end-to-end QOS characteristics, and the operation of multiparty binding and bindings between different kinds of interface (e.g. stream to *operation interface* bindings).

8.4.1 Computational interfaces

A computational interface is characterized by a signature, a behaviour; and an environmental contract.

The signature depends on the interface type which can be operation, stream or signal:

- An *operation interface* has a signature that defines the set of operations supported at the interface and whether the interface has the role of client or server for that set of operations.
- A *stream interface* has a signature that defines the set of flows supported at the interface and, for each flow, whether the interface has the role of producer or consumer.
- A *signal interface* has a signature that defines the set of signals supported at the interface and, for each signal, whether the interface has the role of initiating or responding.

The behaviour is described by the allowed sequences of actions of the computational object that are associated with the interface. The behaviour can include internal actions of the object and will be constrained by the environment of the object, in particular by interactions at other interfaces.

Importantly, each interface specification also contains an environment contract which specifies a set of Quality of Service (QOS) constraints placed on a computational object and its environment. If the environment (other computational objects and the supporting infrastructure) delivers the required level of QOS, then the object itself is guaranteed (by design) to provide a certain level of QOS. The QOS specified for an interface expresses both its requirements on its environment, and the QOS exhibited by the object in an environment that meets the requirements. The particular notation for specifying quality of service is not prescribed by the computational language.

8.4.2 Binding model

Interactions between given computational interfaces are only possible if a binding (i.e. some communication path) has been established between them. The computational language specifies *explicit binding* actions for both operation and stream interfaces. In the case of operation interfaces, it also specifies that binding can be *implicit*, in order to allow the

use of notations that do not provide for the expression of bind actions. *Implicit binding* can only occur for operation interfaces since in other cases it is not self evident where the initiative in the binding is placed relative to subsequent interactions.

Where binding is implicit, an invocation by the client object results in the binding of an appropriate client interface to the server interface and the occurrence of the operation (interrogation or announcement). The computational language leaves undefined whether or not the client interface is deleted at the end of the process. It should be noted that implicit binding has no provision for reference to an environment contract for the binding.

Where binding is explicit, it is defined in terms of two kinds of binding actions: *primitive binding actions* and *compound binding actions*. These binding actions are only applicable in the context of explicit binding.

A primitive binding action allows the binding of two interfaces of the same or of different computational objects. The interfaces must be of the same type, but can be operation, stream or signal interfaces. A primitive binding action is carried out by one of the objects concerned and has the effect of establishing at each interface the information necessary for interaction to take place, i.e. the identity of the other interface concerned. No requirement is seen for defining an explicit *unbinding* action, but deleting either interface, obviously, deletes the binding as well. A primitive binding action requires that the interfaces concerned are of the same kind and have complementary signature type and roles (e.g. one is a client and the other a server).

A compound binding action allows the binding of two or more interfaces of the same or different type by means of a *binding object* (see Figure 2).

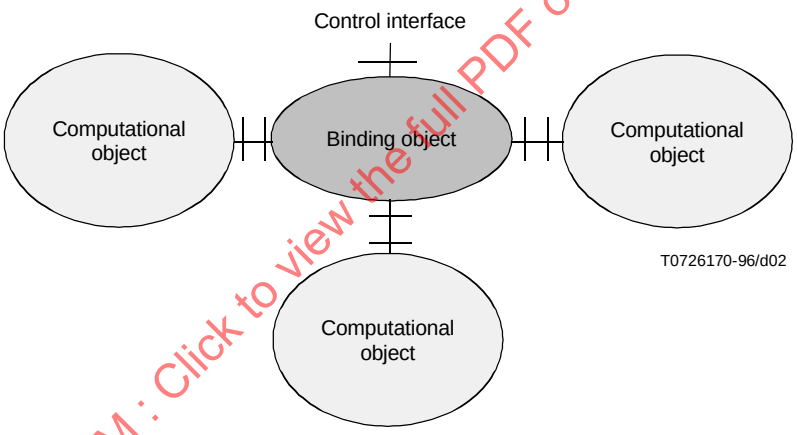


Figure 2 – Compound binding

The action can be carried out by one of the computational objects involved in the binding or by a computational object separate from the binding. It has the effect of instantiating a computational object to support the binding (the binding object). The binding object instantiates an appropriate set of interfaces and uses primitive binding actions to bind them to the interfaces to be bound. It also instantiates a set of control interfaces through which its operations can be controlled and returns the interface identifiers for the interfaces to the initiating computational object.

Behaviours of binding objects reflect the communication semantics they support and the computational model does not restrict the types of binding object, reflecting the fact that there is multiplicity of possible communication structures between objects. Nevertheless, useful classes of binding object may be standardized depending upon classes of applications. In particular, binding objects can specify the operation of multiway bindings and of complex bindings (e.g. between operation and stream interfaces of different types, and between operation interfaces and stream interfaces).

As with any other object, binding objects can be qualified by QOS assertions that further constrain their correct behaviour (e.g. to bound end-to-end communication delay or end-to-end delay jitter at a recipient interface). Where such

QOS assertions are made, the interfaces involved in the primitive bindings of a binding object must be signal interfaces, since the atomic nature of signals makes it possible to specify of the points in space and time at which QOS observations can be made.

Control interfaces for a binding object allow for *deletion* of the binding and can also allow control of its operation and of the QOS that it offers. Examples of the facilities that could be provided are:

a) Control of *notification* of *errors* that disrupt the binding object: This would allow specification of an interface at which the object invokes a notification operation if failures disrupt the binding.

b) Control of a dynamic multicast binding, allowing the addition of new consumers and removal of existing consumers.

c) Group invocation, making atomic multicast invocations available in the computational language and allowing members to be added to, or subtracted from, the group.

d) Control of the QOS associated with the binding, allowing manipulation of specific QOS characteristics: This form of control would be particularly useful for stream bindings for multimedia applications.

e) Notification of events of interest to the application, for example, an event might be signalled at the start or end of a period of silence in an audio flow.

In the case of binding of stream interfaces, the binding may abstract from application specific stream composition rules. In the simplest case, the binding will represent a single flow from a producer interface to a consumer interface (e.g. from an audio filing system to a speaker). However, the composition rules may be more complex:

a) A full duplex path may be created and managed as a single binding; the resultant flows link the producer aspects of the interface on each computational object with the consumer aspects of the interface on the other.

b) A number of full duplex interfaces may be linked by a binding object which encapsulates the rules of a conference system for allowing the flow from a selected *producer* (the current talker) to be delivered to all consumers. Varying degrees of application control might be exercised via the binding control interface to provide explicit flow control.

c) Flows from a number of producers may be combined to provide a composite flow to a single consumer. For example, a video flow from one source and an audio flow from another, might be combined into a single television flow as image and associated commentary. Here the control interface might allow manipulation of engineering flow synchronization mechanisms as part of the provision of lip synch.

An interface can be multiply bound. In the case of implicit binding, multiple binding of an interface requires that each binding be identified by the server interface involved. In the case of explicit binding, multiple binding requires that each binding be identified by the binding object involved.

8.4.3 Typing and subtyping for computational interfaces

Interfaces in the computational language are strongly typed to maximize early consistency checking of distributed programs conforming to the ODP computational language. Interface types are related by a subtyping relation that defines the minimal conditions to impose in order to provide for meaningful object interaction.

The signature type of an interface defines the form and kind of interactions available at the interface, and signature subtyping specifies minimum requirements for one interface to substitute for another. The rules are based on the interaction semantics of computational interfaces, and are sufficient to ensure that a substituted interface can consistently interpret the structure of any interactions that occur. It is, of course, also necessary for interfaces to match in terms of the semantics of data transferred but general rules cannot be defined for carrying out such matching.

A *signal interface signature* defines, for each signal in the interface, its name, the parameters and whether the interface concerned is the *initiator* or responder. An *operation interface signature* defines, for each kind of operation in the interface, the name of the operation, the number and types of its arguments, as well as, for interrogations, the set of possible outcomes for the operation, (terminations). For each termination, the name of the termination, together with the number and types of its arguments are defined.

Subtyping rules for signal and operation interfaces signatures are defined in Annex A of ITU-T Rec. X.903 | ISO/IEC 10746-3. Note that the interaction semantics for other interfaces types in addition to operation and stream can be expressed in terms of signals, since signals provide the basic building blocks from which to model any higher level interaction types.

A stream interface is typed in terms of a set of component flows, each of which has a basic type capable of being supported by the available underlying mechanisms. Examples of basic types are individual audio or video flows. Each component flow has a unique direction, either into or out of the binding. The component flows are organized by the type description into a stream signature (just as arguments are organized in a signature in an operation interface). A stream interface may consist of a number of related flows, either in one single direction or in opposite directions. In the case of streams, however, the interface type can itself be used to represent a more complex flow, so that in types constructed in a series of stages the flows may form a multi-level hierarchy.

Examples of stream interfaces types are:

a) a single audio flow from an audio source;

b) a single audio flow into an audio sink;

c) a full duplex speech type in which there is one inward and one outward flow, to represent a user view of the audio aspects of a telephony service;

d) a composite television signal consisting of both audio and video components;

e) a more complex application-oriented type in which several audio and video flows are combined to represent flows in a virtual reality system.

The fact that the flows in a stream interface each have a direction implies that interface types will, in general, exist in pairs which are related by reversal of all the flows involved. However, if the interface type has an equivalent set of inward and outward flows, the two related types themselves become equivalent. A stream interface definition notation may allow shorthand for asserting this, or for definition of the two related forms simultaneously.

Any stream interface type system will have an associated set of subtyping rules. These differ from the subtyping rules for operational interfaces in that they are constructed to allow communication between computational objects whose stream interfaces offer different capabilities. For example, an audio interface might be considered as a subtype of a composite audio and video interface, so as to allow a remote telephone user to communicate with a user of a videophone system. The optimum form of subtyping will depend on the application, so that selection of a suitable variant of subtyping is part of the design process.

In general, the stream subtyping rules can be divided into two steps:

a) identification of correspondences between primitive flows in the two types, and decision on whether the correspondences found are sufficient for a subtyping relationship to exist;

b) comparison of the types of each of the primitive flows, including comparison of QOS aspects, to determine whether a subtype relationship exists.

It is not possible to define completely general subtyping rules for stream interfaces since these depend upon details of the interactions that are abstracted in the definitions of the streams concerned.

8.4.4 Portability

The computational language defines the actions an object can perform, and enumerates the possible failure modes of these actions. Thus, the computational language defines an object based programming model for a generic virtual machine that is realized by the engineering and technology rules.

Different sets of portability rules can be defined, each of which specifies a particular subset of the actions defined by the computational programming model. A set of portability rules identifies the requirements on a computational notation to support the portability of objects between different environments that provide implementations of those rules. The RM-ODP itself defines a basic portability environment and a complete portability environment, depending on the sets of actions supported.

8.5 Engineering language

The engineering language focuses on the way object interaction is achieved and on the resources needed to do so. It defines concepts for describing the infrastructure required to support selective distribution transparent interactions between objects, and rules for structuring communication *channels* between objects and for structuring systems for the purposes of resource management.

Thus the computational viewpoint is concerned with when and why objects interact, while the engineering viewpoint is concerned with how they interact. In the engineering language, the main concern is the support of interactions between computational objects. As a consequence, there are very direct links between the viewpoint descriptions: computational

objects are visible in the engineering viewpoint as *basic engineering objects* and computational bindings, whether implicit or explicit, are visible as either channels or local bindings.

The concepts and rules are sufficient to enable specification of internal interfaces within the infrastructure, enabling the definition of distinct conformance points for different transparencies, and the possibility of standardization of a generic infrastructure into which standardized transparency modules can be placed.

8.5.1 Clusters, capsules and nodes

The engineering language deals with the basic engineering objects and with various other engineering objects which support them. It relates these objects to the available system resources by identifying a nested series of groupings.

At the outer level, engineering objects are physically located and associated with processing resources by grouping them into *nodes*, which can be thought of as representing independently managed computing systems. A node can be anything which has a strongly integrated view of resources, as long as the system designer can consider it as a whole. Thus a tightly coupled parallel processing system can be considered a node, so long as it has one scheduling and allocation policy – one operating system.

The node is under the control of a *nucleus* which is responsible for initialization, for creating groups of engineering objects, for making communications facilities available, and for providing basic services like timing and the source of unique identifiers.

Within a node, there may be a number of *capsules*. A capsule owns storage and a share of the node's processing resources. It can be thought of in terms of a traditional protected process, with its own address space. A capsule is thus the unit of protection and is generally the smallest unit of independent failure supported by the operating system. There is a special engineering object, called the *capsule manager*, associated with each capsule, and for descriptive purposes, a capsule is controlled by interactions with this manager.

A capsule will typically contain many engineering objects; the grouping of objects into capsules is done to reduce the cost of object interaction. This is because communication between traditional processes is slow and expensive, because of the checks which need to be performed; however, the compiling tools that build capsules can be trusted to *validate* and structure the interactions between closely related engineering objects to a sufficient extent to let them share resources. Resources within a capsule will be controlled by some kind of language specific runtime system.

The smallest grouping of engineering objects is into a set of clusters within a capsule. The objects in a cluster are grouped together in order to reduce the cost of manipulating them. Clusters, capsules and nodes are shown in Figure 3. The engineering objects in a cluster can be *checkpointed* together, transferred to persistent storage, reactivated or moved to another node altogether. This manipulation of complete clusters as a single operation opens the way to the management of very fine grain object-based systems at reasonable cost. For example, a geographical information system might consider data about individual points on a map to be engineering objects, but could not sustain the cost of giving each of these objects a completely separate existence. Communication between engineering objects in a cluster can be highly optimized, since the objects are created together, in the same language, and are expected to stay together.

Interaction within a cluster might therefore be supported by a simple local method invocation or equivalent.

Clusters are controlled and actions on them are initiated by interaction with an associated *cluster manager* object.

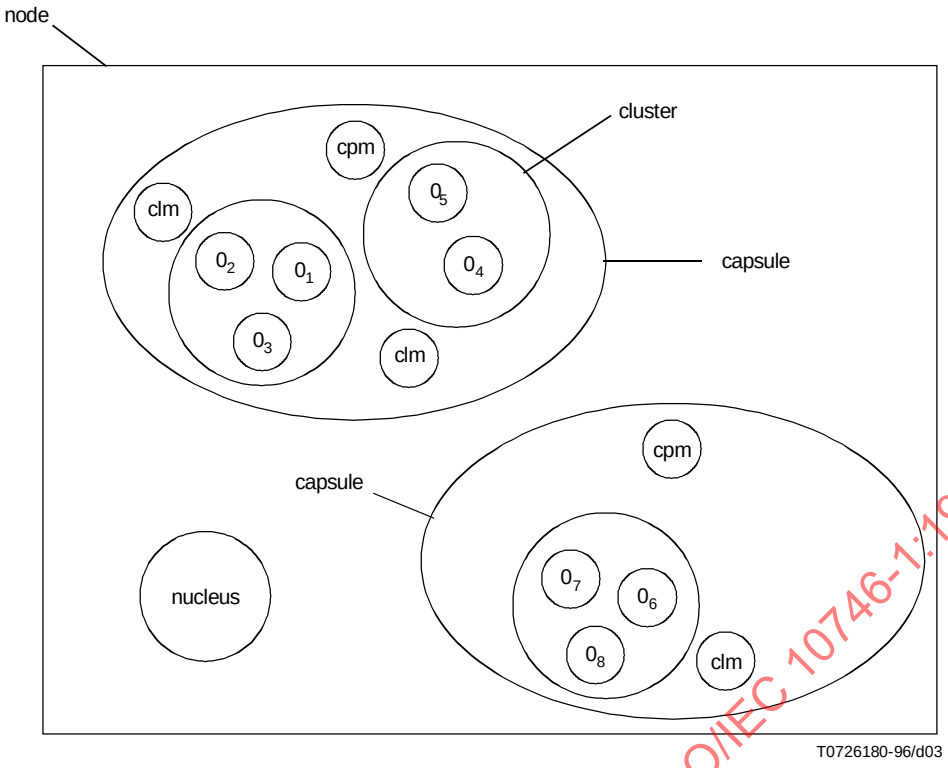
8.5.2 Channels

When engineering objects in different clusters interact, there is a need for a good deal of supporting mechanism. Even if the objects are currently within the same capsule or node, mechanisms are needed to cope with the possibility of one or other of them, terminating, failing or moving elsewhere. The set of mechanisms needed to do this constitute a channel, which is made up of a number of interacting engineering objects (see Figure 4).

The engineering objects within a channel are divided into three types, based on the job that they do. *Stubs* are concerned with the information conveyed in an interaction, *binders* are concerned with maintaining the association between the set of basic engineering objects linked by the channel, and *protocol objects* manage the actual communication.

Stubs interact directly with the basic engineering objects they support, and perform functions such as the marshalling and unmarshalling of parameters, or the logging of information about the interaction being performed. Thus the stubs need access to information about the type of the interaction, or, more generally, the type of the interface that is being supported. This distinguishes them from binders and protocol objects, which transfer complete messages without concern for their internal structure.

Depending on the design of the system, a stub may be directly associated with a particular basic engineering object, or it may be shared between a number of such objects. Sharing generally implies the need to transfer some additional information to identify, and thus distinguish between, the objects being supported.



cpm Capsule manager object
clm Cluster manager object

Figure 3 – Capsules, clusters and nodes

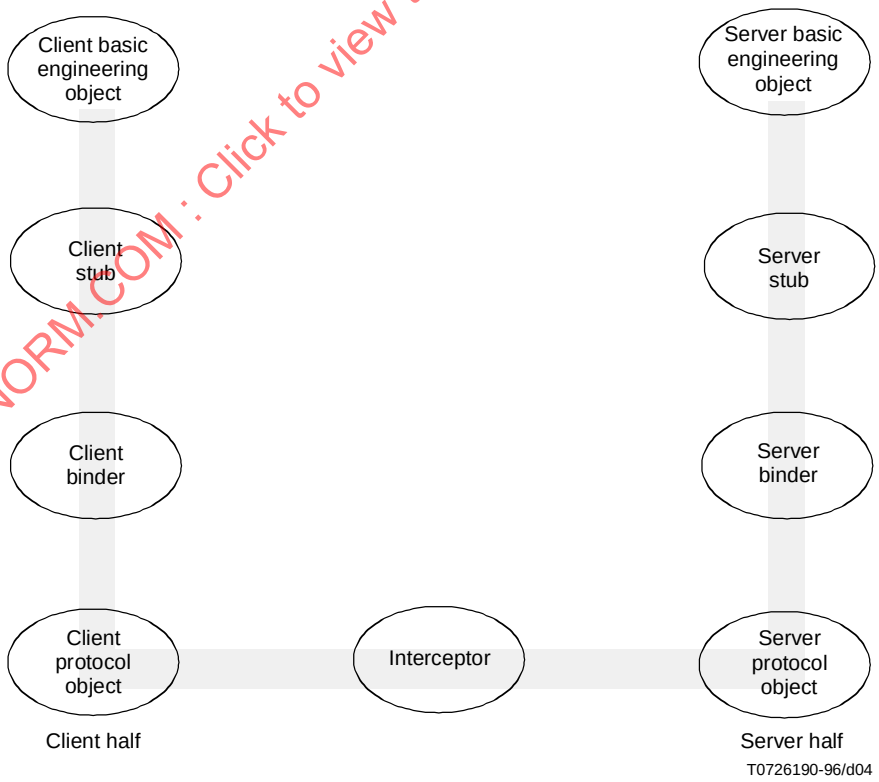


Figure 4 – An example of a client-server channel

Binders are responsible for solving many of the problems of distribution. The binder establishes the binding when the channel is created and subsequently maintains the end-to-end integrity of the channel. This means that it handles changes of configuration and communication or object failures, and keeps track of the other endpoints if objects move or fail and are replaced (the process of object relocation). Binders are thus involved in the provision of many of the distribution transparencies.

The protocol objects provide for communication of sufficient quality and reliability between the binders they serve. In addition to handling whatever peer protocols are in use, they provide access to supporting services, such as directory services for translating addresses, where necessary.

Any of these three kinds of engineering object may itself need to communicate with other parts of the system, in order to obtain the information it needs to do its job, or to supply *management information* to other engineering objects. Such communication may itself need the various distribution transparencies, and so the communication from these objects to elsewhere is by means of a channel; from this point of view, the engineering objects within one channel play the role of basic engineering objects in another. Similarly, any of these objects can support control interfaces, via which they can be managed. For example, a protocol object may provide a control interface through which the target quality of service for the channel can be adjusted.

In cases where the channel crosses some technical or organizational boundary, there may be a need for additional checks or transformations to match the requirements on the two sides. These functions are performed by interceptors (described later in 8.5.7), which form part of the channel. They may need to perform format or protocol conversion, or may provide accounting or access control checks. An interceptor may be built up from protocol objects, binders and stubs, depending on the nature of the job it has to do.

For simplicity, Figure 4 has been drawn showing a configuration of stubs, binders, protocol objects and interceptor supporting a single channel between two basic engineering objects. However, in general, such a configuration could support channels between multiple pairs of basic engineering objects (see Figure 5), or channels with many endpoints, supporting various forms of group communication (see Figure 6) or multicast. In this latter case, the binders are responsible for coordinating communication, but the multicast mechanisms may be provided by either binders or protocol objects, depending on the technology available. Multi-endpoint channels are used to support replication transparency.

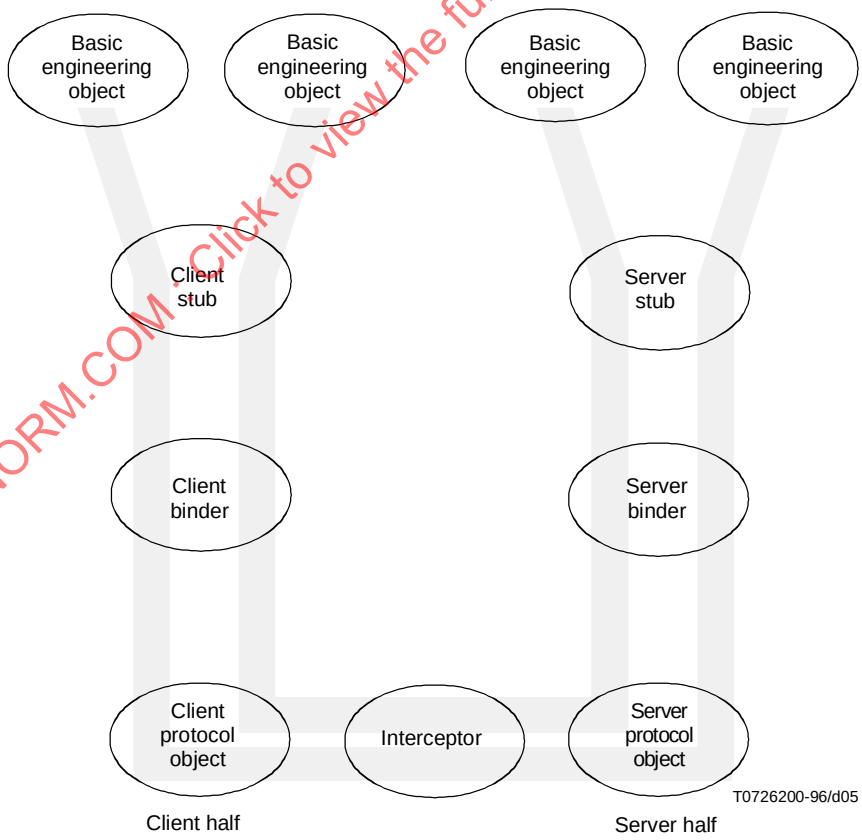


Figure 5 – An example of a multiple channel configuration

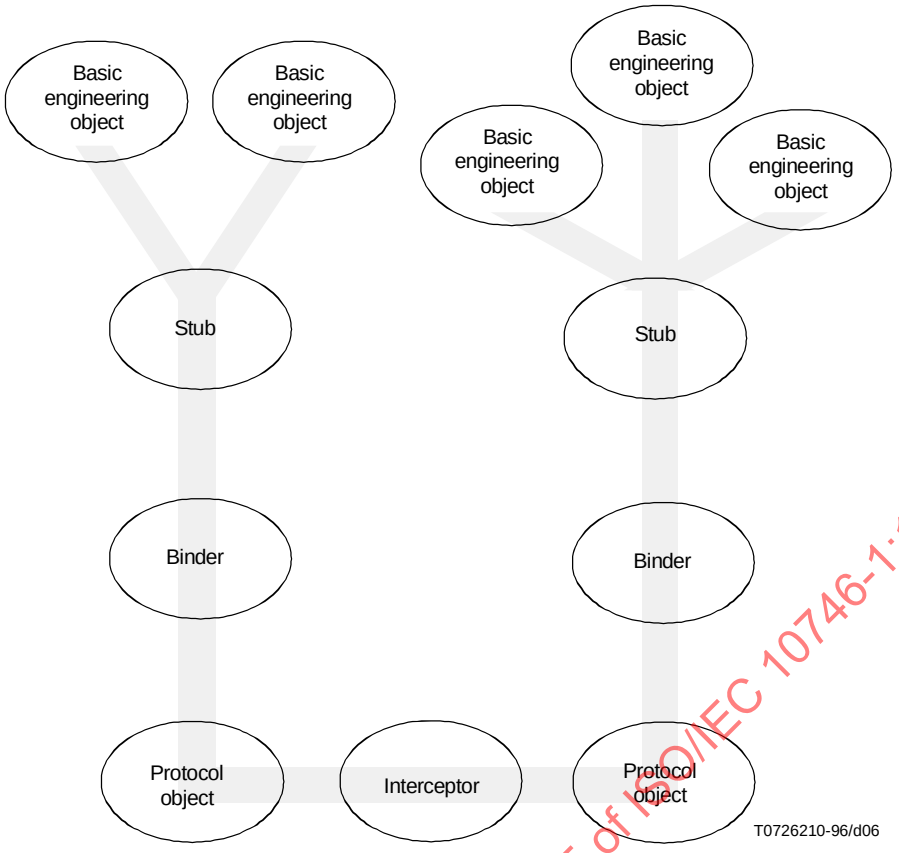


Figure 6 – An example of a group channel configuration

8.5.3 Interface references

When an interface is created, an interface reference for it is generated. The nucleus is involved in this process, so as to make the reference unambiguous, and sufficient resources are allocated and initialized for the engineering objects in that node to participate in bindings if asked to do so.

The interface reference is the key for access to a large amount of information. Given such a reference, it is possible to discover the type of the interface, a communications address at which binding to it can be initiated, and other information about the expected behaviour of stubs, binders and protocol objects within the channel, which is needed for a subsequent binding to succeed. It is also the starting point for calling upon the functions needed to handle errors; knowledge of an interface reference makes it possible to contact an appropriate *relocator*.

This does not imply, however, that the information is all encoded as part of the interface reference; to do so might make it a very big item to manipulate. The architectural requirement is that there should be some prescription for obtaining the necessary information, starting from the interface reference, but the exact prescription, in terms of decoding and enquiry from other engineering objects, can be chosen differently in different system designs.

In addition to these design variations, there can also be variations arising from the existence of multiple naming domains and the allocation of references with respect to these domains. For both these reasons, it can be necessary for interceptors, or other engineering objects in the channel, to transform interface references when they are passed across domain boundaries.

8.5.4 Binding

There are two kinds of engineering binding. Within a cluster, or between the engineering objects which cooperate within a node to provide a channel, there are local bindings, which are provided by system-specific mechanisms. Such bindings are regarded as primitive in the architecture. On the other hand, the bindings supported by channels provide appropriate distribution transparencies; these are called distributed bindings, and creating them will generally involve some interaction between a number of nodes to establish the channel.

8.5.5 Channel establishment

In order to establish a binding between engineering objects, it is necessary to be able to identify and describe the interfaces of those objects. An interface reference is used to identify and describe an engineering interface in sufficient detail to enable that interface to be bound. An interface reference can be passed through one or more interactions between engineering objects. It enables any object which receives that interface reference to initiate a binding to that interface without any additional information.

An engineering object which wishes to pass an interface reference to one of its interfaces, asks its nucleus to create an interface reference (via an operation on its node management interface). An interface reference identifies the interface and provides the information necessary to create a binding to it.

The type of an engineering interface conveys the computational interface type that it implements in addition to the channel configuration of stubs, binders, protocol objects, and interceptors (described later in 8.5.7) needed to support it. When engineering interfaces with compatible computational interface types are bound together, the binding establishment negotiates the exact configuration of the channel needed to support all the interfaces it encompasses. It might be possible to optimize this configuration. For example, if the interfaces support the same representation of data, then there is no need for the channel to be configured with engineering objects to convert between the data representations, although it will still be necessary to marshal a copy of the arguments or results into a message.

Bindings between interfaces are created through the interaction of the nucleus objects (usually, those nodes at which the interfaces are located). Therefore, when a nucleus creates an interface reference, it supplies sufficient instructions on how to be contacted in order to establish a binding to the referenced interface. In engineering language terms, the nucleus identifies the *communication interface*(s) at which the nucleus-nucleus interaction must occur. The identification of a communication interface might require information about the communication protocols supported at that communication interface.

Typically, the nucleus nominates one of its own communication interfaces for binding-establishment interactions, but a more general approach is also permitted. The communication interface (and its communication protocol) used in the binding-establishment interaction do not imply anything about the communication interface (and communication protocol) used to support the binding once it is created. An analogy is that a telephone conversation can be used to initiate the sending of paper documents through the postal system.

The interface reference conveys highly complex information which is interpretable throughout the ODP system. The detailed structure of an interface reference is, therefore, the subject of separate standardization. The RM-ODP does not prescribe whether the interface reference physically contains the information described here, or is simply a key used for accessing this information through interaction with other engineering objects (e.g. a name to be resolved by a name server).

As an example of channel establishment, consider the establishment of a stream channel between two engineering objects. This takes place in several steps.

*** Step 1**

One of the two engineering objects initiates the configuration of a channel by interaction with its nucleus. The interaction syntax may have the following format:

InitChannel (*StreamChannel*, *producer/consumer*, *IFPC1*, **result** *IFrefStreamchannel*)

where *StreamChannel* is of type "*StreamChannel*" and *Streamchannel* is the type of channel to be created; *producer/consumer* indicates that the concerned engineering object will have both a producer and consumer role for the stream channel; *IFPC1* is the interface of the engineering object to be bound to the stream channel.

When this interaction occurs, the nucleus creates a stub object, a binding object and a protocol object corresponding to the channel type and the role. These engineering objects are bound to create a first part of a stream channel. The presentation interface of the stub object is bound to the *IFPC1* interface. The stub object is then bound to the binding object that is bound to the protocol object. The result of this interaction is an interface reference (*IFrefStreamchannel*). The interface reference will be communicated to the engineering objects that want to bind to the channel.

*** Step 2**

The interface reference of the channel is communicated to the second engineering object. This object interacts with its nucleus to bind to the channel by means of the following interaction:

BindChannel (*StreamChannel*, *producer/consumer*, *IFPC2*, *IFrefStreamchannel*)

where *StreamChannel* is the type of channel; *producer/consumer* indicates that the second engineering object will have both a producer and consumer role for the stream channel; *IFPC2* is the interface of the second object to be bound to the stream channel.

The nucleus determines from the interface reference *IFrefStreamchannel*, the channel type and location of the protocol objects for the other participants in the stream channel. The nucleus creates a stub object, a binding object and a protocol object corresponding to the channel type of the other participants and the role. These objects are bound to the part of the stream channel already established and to the second engineering object. Then the binders in the channel interact with each other to enable communication across the channel.

*** Step 3**

Other objects may bind to the existing channel using the same **BindChannel** () interaction.

8.5.6 Management interfaces

Only an object can modify its own behaviour. An object may respond to requests from a management application to modify its behaviour and may, in consequence, delegate responsibility for some part of its management to the management application.

Resource management requires that it should be possible to invoke management operations on individual services, the engineering objects that contain them, the cluster that contains the engineering object, the capsule that contains the cluster and the node that supports the capsule.

8.5.7 Interceptors

Interceptors are specified to meet requirements, identified in an enterprise specification, for the federation of technology or administrative domains. Interceptors correspond to the notions of “gateway”, “agent”, or “monitor” objects which stand between two domains and enable or permit interactions on the basis of a contract between the Administrations that specifies the basis for their federation.

Within a technology domain, the nucleus objects have identical data representations and functionality of protocols, naming and addressing. When two technology domains meet, there is an opportunity to merge the domains – i.e. computational objects in each domain are extended to use both their own and the foreign domain’s technology (or both switch to a more generic common technology). Where this occurs, the technology boundary dissolves and access transparency is sufficient. Interceptors cover the case in which it is not possible to modify the technology of one or both of the domains and, in consequence, interception must occur at the boundary, providing protocol conversion and name translation.

Protocol and data translation over technology boundaries are carried out by in-line interceptors and these are involved in all object interactions over the boundary. For efficiency, only one in-line interceptor would be used for each technology boundary (see Figure 7).

Administrative boundary interceptors exist entirely within an Administration and fulfill protection responsibilities for an Administration. For instance, an administrative boundary interceptor could be used in the translation of security information containing permissions. Such an interceptor could be used before an interaction by the invoking computational object infrastructure. An administrative interceptor may communicate with the similar interceptor in the other domain to exchange information, such as cryptographic keys, and to check administrative information before translating it (see Figure 8). The translations carried out by administrative interceptors may be used by several successive interactions between computational objects without further use of the interceptor.

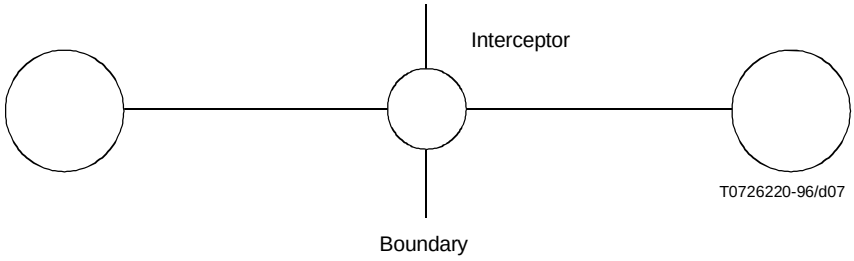


Figure 7 – In-line Interceptor – Technology boundary

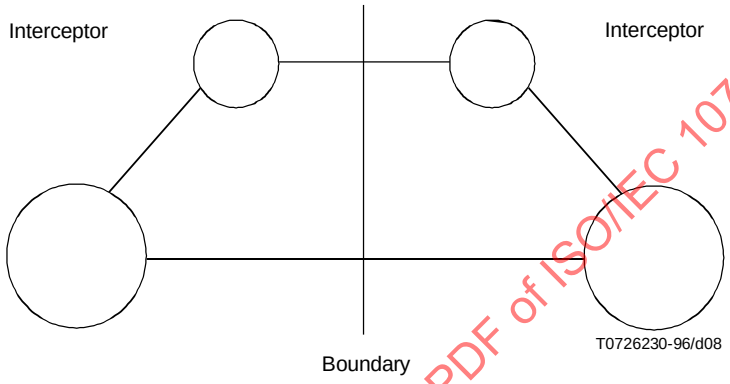


Figure 8 – Split Interceptor – Administrative boundary

Where an administrative and a technology boundary coexist, then both in-line and off-line interceptors would be used (combination of Figures 7 and 8). Alternatively, the in-line interceptor may be split, so that there are two in-line interceptors, one in each administrative domain to support the trust relationships (see Figure 9).

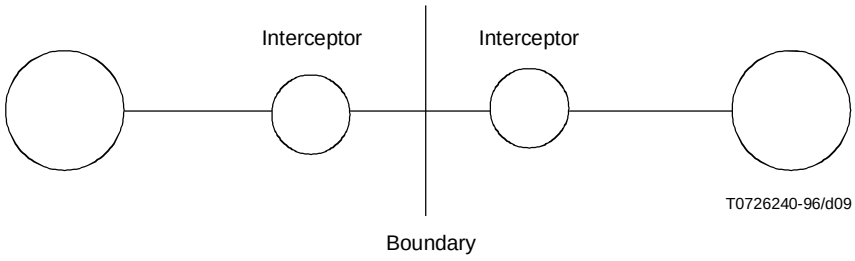


Figure 9 – Split Interceptor – Combined technology and administrative boundary

In general, boundaries are N-way because more than two subsystems may meet at the same location, logically however an N-way boundary can always be modelled as N 2-way boundaries and thus, architecturally, only the 2-way interceptor case needs to be considered.

Special conditions arise where federation requires that the traders in each domain be linked. If a trader is accessible through an in-line interceptor, then that trader can be used to access subsequent traders, but the interceptor must provide special initialization functions to allow access to at least an initial trader across itself in both directions. This initialization can be done by having the interceptor federate traders on either side of its boundary, by having the interceptor become a proxy trader, or by building a trader into the interceptor itself.

8.5.8 Conformance points

The structuring of the engineering specification into clusters, capsules and nodes, and the support of interaction by structured channels gives rise to a large number of interfaces, any of which can be selected as a conformance point, allowing for observation and conformance testing.

The various interfaces involve different kinds of conformance. The interface between protocol objects is an interworking conformance point, providing for familiar methods, like OSI testing, based on observation of the communication behaviour. Most of the other interfaces are internal to a node, and represent boundaries between software modules; they are *programmatic reference points* and allow testing for software compatibility and portability. Some of the interfaces to basic engineering objects may allow other forms of conformance testing, for interchange or perceptual conformance (correct interaction with the real world), they may, also, be conformance points at which behaviour is assessed for consistency with requirements in the enterprise and information specifications.

8.6 Technology language

The technology specification describes the implementation of the ODP system in terms of a configuration of technology objects representing the hardware and software components of the implementation. It is constrained by cost and availability of technology objects (hardware and software products) that would satisfy this specification. These may conform to implementable standards which are effectively templates for technology objects. Thus, the technology viewpoint provides a link between the set of viewpoint specifications and the real implementation, by listing the standards used to provide the necessary basic operations in the other viewpoint specifications, and the aim of the technology specification is to provide the extra information needed for implementation and testing by selecting standard solutions for basic components and communication mechanisms. Such a selection is necessary to complete the system specification, but is largely divorced from the rest of the design process.

There are consequences of the technology selection, however. One area in which the selections in the technology specification feed back to other aspects of the system design is in the provision of a specific quality of service. The selections in the technology viewpoint determine the performance costs of interactions and thus, indirectly, the quality of service which can be achieved by the behaviour defined in other viewpoint specifications.

The technology specification plays a major role in the conformance testing process. It identifies the conformance points in the real system at which a tester can make observations of its behaviour and it supplies the information needed to interpret the observations a tester can make in terms of the vocabulary and concepts used in the other viewpoints of the system specifications. For example, it allows valid interactions to be recognized, so that their appropriateness can be checked against some specified technology object behaviour. The information required for this purpose is called Implementation Extra Information for Testing (IXIT).

8.7 Consistency between viewpoints

The five viewpoint specifications for a system are linked by statements defining the relations between key terms in them and establish that:

- the specifications relate to a single system and are not independent;
- the specifications are self consistent;
- observable behaviour at conformance points in the technology specification can be related to requirements in the other viewpoint specifications.

Many of the links needed will be provided implicitly by the notations used, resulting from correspondences between names. However, some of the key constraints need to be stated explicitly. In the architecture, constraints are placed on the relations between terms in the viewpoint languages themselves, establishing some limits on the mappings which can

be established. Most of the constraints placed are between terms in the computational and engineering languages, and are defined so as to create consistent interpretations when system components, such as those supporting the ODP functions, are specified separately.

Clear mappings between viewpoints are necessary if the processes of identifying interfaces and of providing transparencies are to be supported automatically by development tools. For example, a computational object may be realized as a set of linked engineering objects, but a single engineering object cannot represent multiple computational objects; a computational interface cannot be divided into separate engineering interfaces except where they are related by replication functions; computational interfaces are identified unambiguously by engineering identifiers. These kinds of constraint help to ensure that common engineering mechanisms will be able to support the full range of possible computational behaviours.

In order to illustrate the nature of the correspondences that may apply, suppose, for example, that a certain specification of a given system may be represented as a set of interacting Viewpoint 1 (V1) objects, as represented in Figure 10.

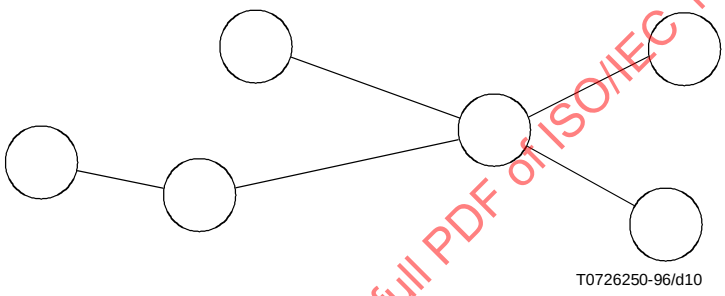


Figure 10 – A V1 view of a system

The same system might be described from another Viewpoint (V2) as a different set of different interacting V2 objects, as represented in Figure 11.

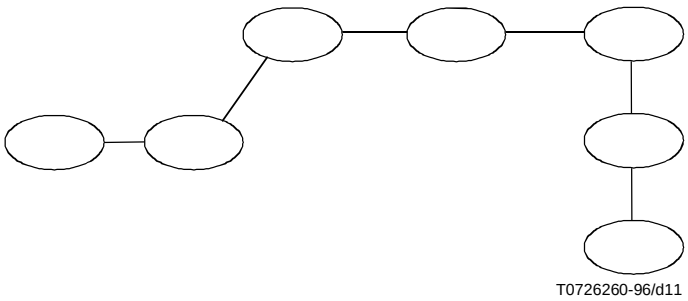


Figure 11 – A V2 view of a system

Since these are two descriptions of the same system, it is possible to group objects in the two preceding pictures, in order to verify that the correspondence rules are satisfied between groups. This process is depicted in Figure 12.

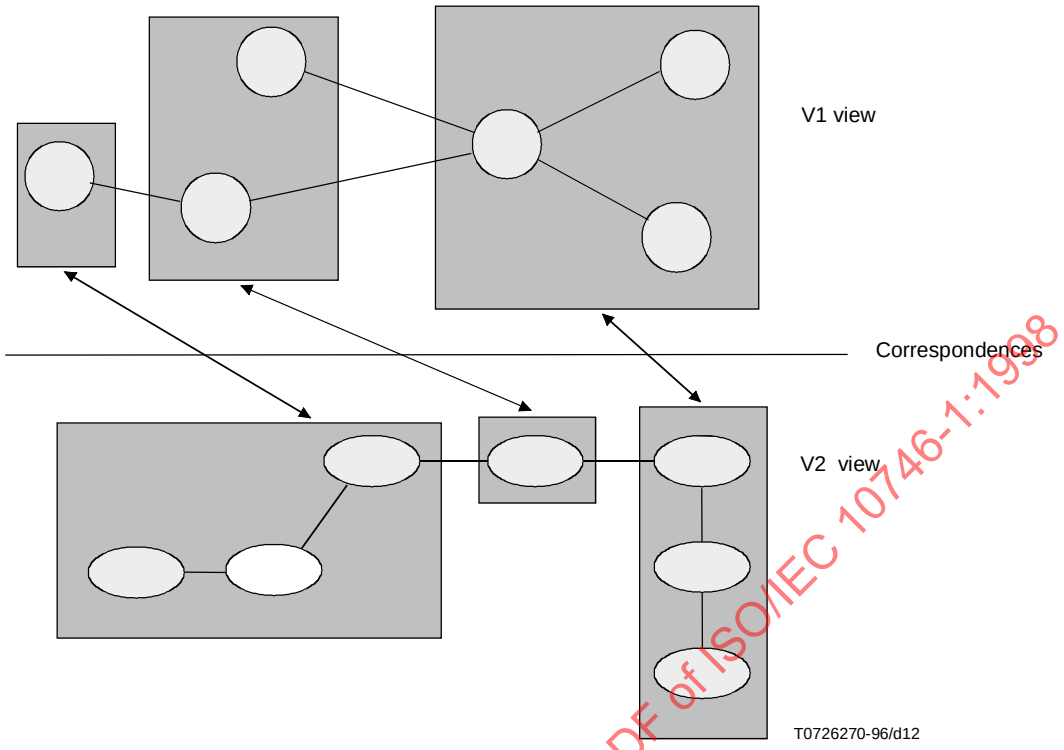


Figure 12 – Correspondences between different viewpoints of a system

If such a correspondence cannot be established, then the two different descriptions are not consistent, and should be refined until a correspondence can be demonstrated.

Configurations of objects that are compared (the configurations of boxes in Figure 12) are, in general, defined for the sole purpose of finding a correspondence between two specifications.

In other words, in order to compare a specification SpecA written in a given viewpoint language L1, and another specification SpecC of the same system, written in another viewpoint language L2, in the general case it is necessary to:

a) Transform SpecA into another specification in L2. Call this specification SpecB. Note that the RM-ODP **does not define** any transformation algorithms.

NOTE – It may sometimes be convenient, in order to carry out this transformation, to derive from SpecA another specification of the same system, still written in L1, that is equivalent to SpecA, in order to better verify correspondences. This corresponds, for example, to grouping the objects defined in the system of Figure 12 into other objects (the boxes).

b) Verify that there are no conflicts between SpecB and SpecC.

Correspondences apply between specifications expressed in different viewpoint languages, not between terms of those languages. In other words, there is no direct translation from one viewpoint language into another one.

8.7.1 Enterprise viewpoint consistency with other viewpoints

The enterprise language should serve as the basis for specifying enterprise goals which must be reflected directly or indirectly in all other viewpoint specifications. The enterprise viewpoint describes, explicitly, the objectives of the system in the context of the organization in terms of members, roles, actions, purposes, usage and policies.

Therefore, an information, computational, engineering or technology viewpoint specification is consistent with an enterprise specification if all the roles, activities, and policies described in the enterprise specification, are correctly reflected. For instance, dynamic schemata defined in an information specification must obey the policies described in the enterprise specification. Different roles identified in the enterprise specification may be supported by different computational objects, having different transparency requirements. Thus, transparency needs for each role in the enterprise specification should be reflected by the use of the corresponding transparency mechanism in the engineering specification. A flexibility requirement or policy in the enterprise specification can lead to the choice of specific technologies for implementation of a distributed system.

8.7.2 Correspondences between computational and engineering specifications

A correspondence exists between the computational specification and an engineering specification to be executed. This engineering specification exhibits the behaviour described in the computational specification. Figure 13 is an example of this correspondence.

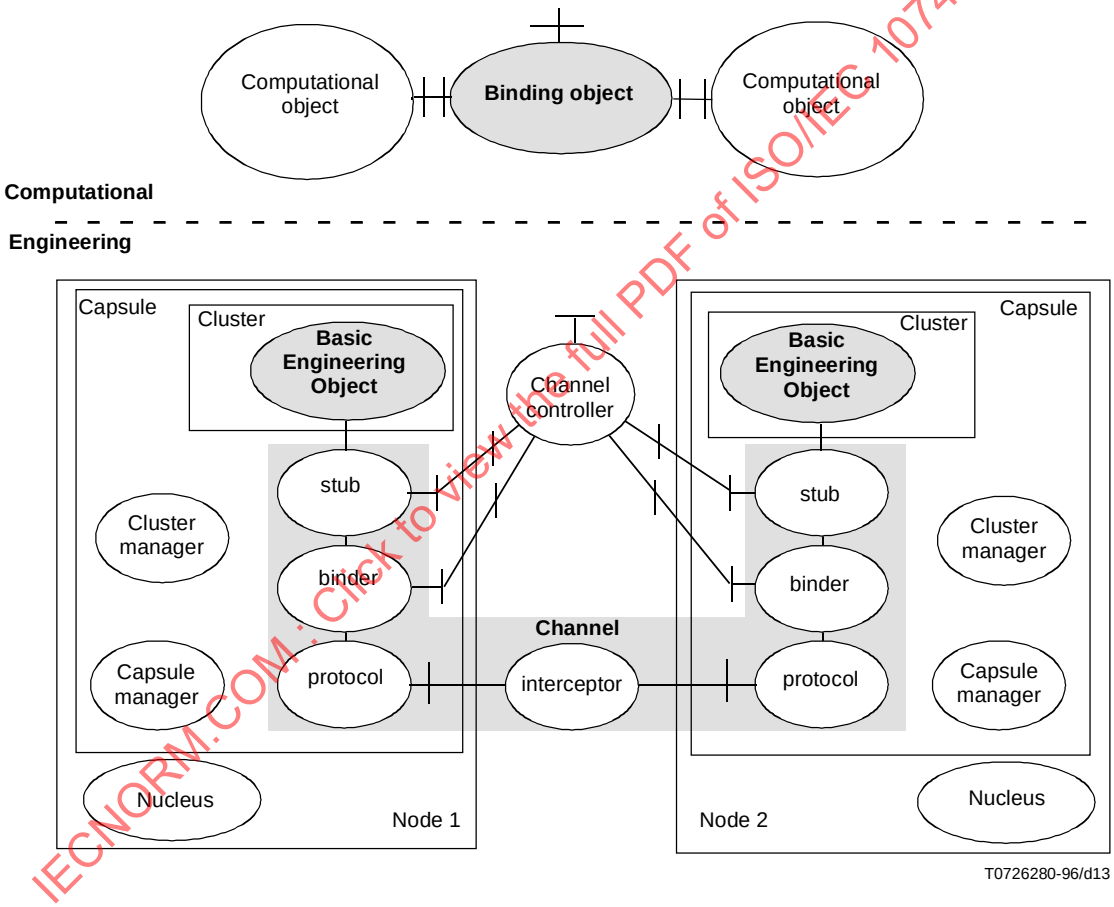


Figure 13 – Example of correspondence between computational and engineering viewpoints

The basic engineering objects correspond to computational objects. Basic engineering objects are grouped into clusters that could represent, for instance, executable pieces of code in C or C++. Clusters are organized into a capsule that could represent, for instance, a UNIX process. The capsule is bound to a nucleus object (representing, for instance, a particular operating system) that belongs to a node (representing, for instance, a work station). Basic engineering objects are supported by additional engineering supporting objects as shown in Figure 13.

The *refinement* of computational templates into engineering templates corresponds to the notion of *compiling* programs to produce object code. The refinement of engineering templates into *cluster templates* corresponds to the notion of *linking* modules to form an executable program image. The concept of capsule corresponds to the notion of *address space* or *process* in most operating systems.

The binding object in the computational specification represented in Figure 13 corresponds to a channel configuration in the engineering specification. The environments constraints specific to the interfaces that are being bound (e.g. security, QOS) are taken into account while establishing a channel between the basic engineering objects concerned.

The control interface of the binding object in the computational specification corresponds to interfaces to stubs and binders in different nodes in the engineering representation. A channel controller object could be introduced that is in charge of the dispatching of the control operations. The communication between the channel controller object and the stub and binders takes place through channels established for this purpose (not shown in Figure 13). Supporting engineering objects may be created (e.g. synchronization objects) to manage and control a set of interrelated channels.

More generally, a computational specification describes the functionality of a system as a set of interacting computational objects. The engineering specification is constrained by the computational specification in that it has to respect the computational objects and their interfaces. Each computational object must be represented as one basic engineering object or as a group of basic engineering objects. In the simplest case, a single engineering object is equated to a computational object. For example, an application program, when compiled, corresponds to an engineering object to be loaded as a load module in a cluster, after link-editing with other engineering objects like stub, binders, etc. This configuration is depicted in Figure 14.

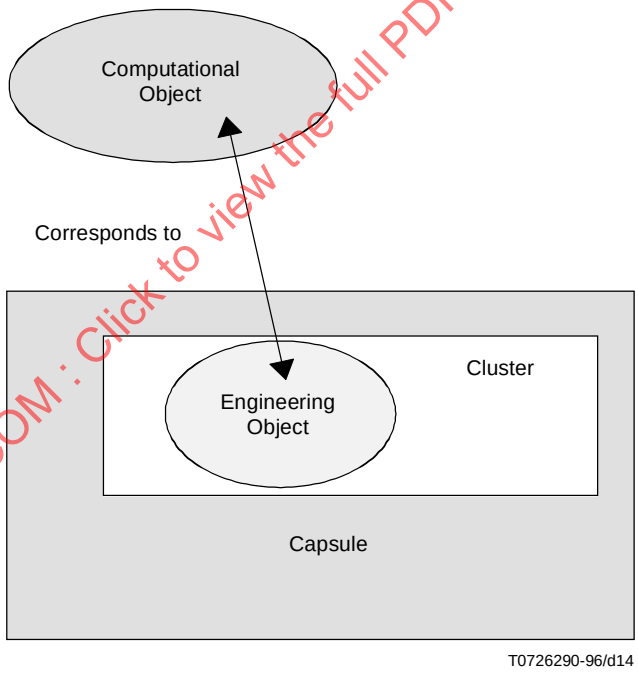


Figure 14 – One-to-one correspondence

For each computational interface there must be one corresponding engineering interface, except where transparencies which replicate objects are involved. In this case, the same computational interface can be associated with different engineering interface identifiers, so allowing replication, e.g. for performance reasons. The computational interface is associated with a set of *engineering interface references*, corresponding to different engineering objects. The activities of these engineering objects must be coordinated by replication objects in order to ensure that the system maintains a

consistent global state. An example of this case is in Figure 15, which shows two basic engineering objects, located in two different nodes, that replicate the functionalities offered at a computational interface. Since the engineering objects are in different nodes, replication coordination is ensured by two replication objects, one on each node, that communicate through a channel.

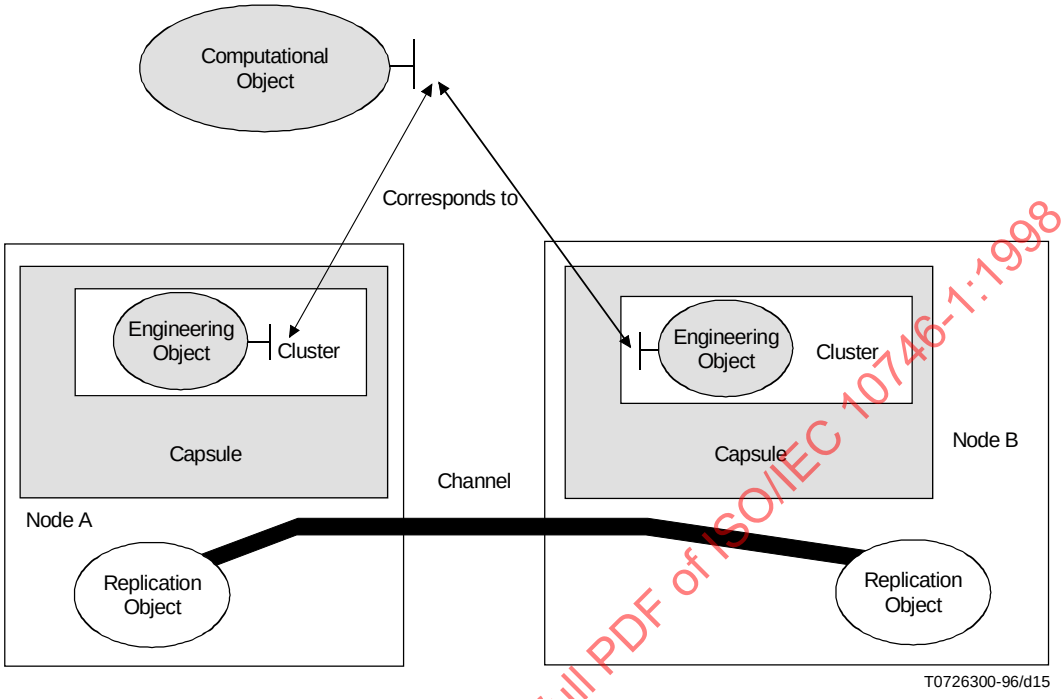


Figure 15 – Many-to-one correspondence

The refinement process between the computational specification and the engineering specification may simply consist of the identification of suitable supporting objects to populate channels that represent binding objects in the computational specification. In other cases, there may be significant transformation of the templates for the computational objects themselves, replacing declarative statements about behaviour (e.g. synchronization constraints) with explicit use of appropriate ODP functions (e.g. the transaction function).

Given sufficient knowledge of the system's configuration, particular optimizations can be engineered back in cases where full generality of the engineering structure is not required. Thus, an implementor can short-circuit interactions between basic engineering objects at the same node (for example, by using local procedure calls for increased performance) provided that such short-circuits do not affect the interworking through interfaces visible to objects at other nuclei. In this way, the architecture copes with distribution without requiring a multiplicity of mechanisms to cater for local and remote interactions, and yet without sacrificing execution efficiency.

8.8 ODP functions

ITU-T Rec. X.903 | ISO/IEC 10746-3 provides outline descriptions of a set of ODP functions. These functions are either fundamental or widely applicable to the construction of ODP systems. Detailed specifications for these functions will be the subject to specific standardization activities and the resultant standards may combine specifications of ODP functions to provide specifications for components of ODP systems.

The complete set of ODP functions is divided into four groups:

a) management functions;

b) coordination functions;

c) repository functions;

d) security functions.

8.8.1 Management functions

The management functions comprise:

- the node management function;
- the object management function;
- the cluster management function; and
- the capsule management function.

The node management function is provided by the nucleus of a node, and is concerned with control of processing, storage and communications functions within a node. It provides for:

- management of processing threads;
- clock access and timer management;
- channel creation and the handling of engineering interface references;
- capsule template instantiation and capsule deletion.

The object management function is provided, where required, by any object and allows for the checkpointing and deletion of the object.

The cluster management function is provided by a cluster manager and allows for the checkpointing, recovery, migration, deactivation or deletion of the cluster.

The capsule management function is provided by a capsule manager and allows for the instantiation (including recovery and reactivation), checkpointing, deactivating or deleting of all clusters in a capsule, and deletion of the capsule itself.

8.8.2 Coordination functions

The coordination functions comprise:

- the event notification function;
- the checkpoint and recovery function;
- the deactivation and reactivation function;
- the group function;
- the replication function;
- the migration function;
- the transaction function; and
- the engineering interface reference tracking function.

The event notification function records and makes available event histories. Event producers interact with the function to create event histories and the function notifies registered event consumers of the availability of event histories.

The checkpoint and recovery function coordinates the checkpointing of clusters and the recovery of failed clusters from *checkpoints*. It is governed by policies covering when clusters should be checkpointed and where the associated checkpoints should be stored, when and where clusters should be recovered, and which checkpoint should be recovered.

The deactivation and reactivation function coordinates the deactivation and reactivation of clusters. It is governed by policies covering when clusters should be deactivated and where the associated checkpoints should be stored, when and where clusters should be reactivated, and which checkpoint should be used for reactivation.

The group function provides the necessary mechanisms to coordinate the interactions of objects in a multiparty binding.

The replication function is concerned with the special case of a group in which the objects in the group are behaviourally compatible. It provides the necessary mechanisms to ensure that the group appears to other objects as if it were a single

object and also allows the membership of the group to be increased or decreased. The function can be used at the level of a cluster, in conjunction with the group function, to form a coordinated set of replica groups where the objects in each cluster form a replica group.

The migration function coordinates the migration of a cluster from one capsule to another. It can operate either by replicating the cluster, making use of the replication function, or by deactivating the cluster and reactivating it in another cluster, using the deactivation and reactivation function.

The transaction function coordinates and controls a set of transactions to achieve a specified level of visibility and permanence, subject to policies that determine the actions of interest for the transaction. The ACID transaction is a special case of the transaction function for which the transactions have the properties of being atomic, consistent, isolated and durable.

The engineering interface tracking function monitors the transfer of engineering interface references between engineering objects in different clusters in order to determine when the supporting infrastructure for the reference is no longer required because no object in any other cluster can bind to the referenced interface.

8.8.3 Repository functions

The repository functions comprise:

- the storage function;
- an information organization function;
- the relocation function;
- the type repository function; and
- the trading function.

The storage function stores data.

The information organization function manages a repository of information described by information schema and allows modification and updating of both the schema and the repository, and querying the repository.

The relocation function manages a repository of locations for interfaces and management functions for clusters supporting those interfaces.

The type repository manages a repository of type specifications and type relationships.

The trading function supports the export of *service offers* by service providers in the form of information about the interface at which the service is provided, and the import by service users of service offers matching specific requirements.

8.8.4 Security functions

Security functions address requirements for confidentiality, integrity, availability and accountability. They comprise:

- the access control function;
- the security audit function;
- the authentication function;
- the integrity function;
- the confidentiality function;
- the non-repudiation function; and
- the key management function.

The access control function prevents unauthorized interactions with an object.

The security audit function monitors and collects information about security related actions, and allows the analysis of the information to review policies, controls and procedures.

The authentication function provides assurance of the claimed identity of an object.

The integrity function detects and/or prevents the unauthorized creation, alteration or deletion of data.

The confidentiality function prevents the unauthorized disclosure of information.

The non-repudiation function prevents one object in an interaction from denying its involvement in the interaction.

The key management function provides facilities for the management of cryptographic keys.

The functions provide services that can be applied both to objects themselves and to the interactions between objects. The mechanisms for providing security services themselves require protection, since an intelligent and malicious threat is a characteristic of environments in which security is required. Engineering encapsulation can help to provide such protection. In many cases security services can be provided without requiring reference in computational specifications.

8.9 ODP distribution transparencies

As described in 8.5, engineering objects interact with one another via stubs, binders, protocol objects, interceptors and nuclei. The engineering objects cooperate to provide a transparency by bringing uniformity to some aspect of the distribution of the basic engineering objects they support. For example, they may translate invocations into exchanges of messages using a common data format to mask differences in data encoding.

Some forms of transparency require supporting functions. For example, if engineering objects can move from one location to another, a means of recording and discovering the current location of a component is required (the relocation function).

Supporting functions may themselves have transparency requirements. For example, a relocation function may be replicated to increase its availability.

Supporting functions are modelled as engineering objects so that the architecture provides the maximum degree of configuration flexibility and reuse of architectural concepts in defining the distribution of these functions. In an implementation, supporting functions may be, for example, collocated with one another for efficient interaction or replicated for reliability.

Distribution transparency is the property of hiding the properties of distribution from end users and specifiers in the enterprise, information, and computational languages. “Component composition” standards will contain precise recipes for using functions and other base components to provide transparencies.

8.9.1 Access transparency

Access transparency enables interworking across heterogeneous computer architectures and programming languages.

Access transparency is critical in building distributed systems using heterogeneous computer architectures, programming languages, etc.

8.9.2 Failure transparency

Failure transparency *hides* from a computational object the failure and possible recovery of other computational objects or the object itself, to enable fault tolerance. It can be provided by an appropriate infrastructure. Otherwise, it is supported by the checkpoint and recovery function or by the replication function, together with the relocation function.

A service which relies on checkpointing alone for failure transparency must provide, as part of that service, means for clients to detect that there has been a restart from a checkpoint and that state information held by the client about the service may be out-of-date. If there is a requirement for consistency between multiple computational objects, then transaction transparency should be specified.

8.9.3 Location transparency

Location transparency hides from a computational object the locations in space at which the computational objects with which it interacts reside. This implies that interfaces can be identified and accessed without specifying their location in space.

8.9.4 Migration transparency

Migration transparency hides from a computational object the fact that it has moved. It is supported by the migration function.

Migration transparency may be combined with persistence transparency or with failure transparency, so that a cluster is not reactivated at its original site but the cluster template is transferred directly to the new location and the cluster is reactivated there.

8.9.5 Persistence transparency

Persistence transparency hides from computational objects the allocation and deallocation of resources to clusters or their templates, and provides for the sharing of resources. It is supported by the reactivation and deactivation functions.

A basic engineering object in a cluster can interact with other objects. When the cluster is deactivated, the basic engineering object is saved in a cluster template, with its activities frozen, and cannot interact with other objects. Persistence transparency hides the deactivation and reactivation of cluster templates, so that basic engineering objects always appear to be available for interaction. This implies that an object has a lifetime of its own, independent of its supporting environment.

8.9.6 Relocation transparency

Relocation transparency hides from a computational object the fact that interfaces to which that object is bound have changed their location. This implies the ability to re-establish binding if necessary. The relocation transparency is supported by the relocation function.

8.9.7 Replication transparency

Replication transparency masks the use of a group of computational objects to support a single computational interface. It is supported by the replication function and the relocation function.

8.9.8 Transaction transparency

The coordination of transactions involves the scheduling, monitoring, and recovery of the actions of interest within those transactions. To achieve this control requires interaction between the computational objects involved in the execution of the actions of interest and the computational objects that realize the transaction function. It is not generally possible to coordinate the actions of interest simply by configuring engineering objects in the channels to intercept the actions of interest. In particular, internal actions such as the start and completion of a transaction are not detectable by monitoring the interactions between computational objects.

Therefore, transaction monitoring and control will usually require computationally visible interactions between computational objects representing the information objects or dynamic schemata (i.e. application objects) on the one hand, and computational objects providing the transaction function on the other hand. However, transaction processing is very complex and it is undesirable to complicate the specification of application functionality by the addition of the complex interactions needed for transaction control.

Transaction transparency is the provision of an automatic process that refines a computational specification without transaction control into a computational specification with transaction control.

The nature of the refinement, and the extent of involvement of the specifier in the refinement process, will depend on the particular transaction mechanism to be used. Typically, there will be the need for additional computational interfaces to be bound to the computational objects involved in the transaction function with interactions on these computational interfaces to coordinate the scheduling, monitoring and recovery of actions of interest to the transaction function. The computational object's behaviour will need to be extended to add recovery actions and the interfaces giving access to its normal functionality might be extended or replaced.

9 Conformance assessment

9.1 Conformance assessment and the development process

Product development extends from the initial realization of one or more requirements for an ODP system to the final provision of an example of an ODP system that fulfils those requirements. The development process potentially involves the production of a number of specifications. One specification may be responsible for the generation of a number of subsequent specifications using one of a number of types of step ("transformations") including:

- translation; and
- refinement.

Specifications are expressed in some natural or formal language. Translation produces a specification with the same meaning (usually in a different language). Refinement, on the other hand, produces a specification with new details that serve to define the product more closely. If each specification is characterized by the set of potential products that it could specify, translation leaves the set unchanged but refinement results in a subset.

After installation and commissioning, an instance of the ODP system then enters a phase of operational use, during which it should meet the needs expressed in the document specifying its requirements. The product's ability to do so depends upon a number of practices during each of these development phases. Standards for providing "quality" describe consistent sets of such practices along with the organization of people, documentation, and life cycle stages to which they apply. Typically, some measurement of quality is made at each phase and changes are made if the quality is found to be lacking. Conformance assessment provides such a measure of quality, usually in the phase during which the implementation specification is realized. However, it may also be used in or have implication for other phases.

9.2 Conformance assessment: Relevant relationships

The relationships between specifications and real implementations that are relevant to conformance are divided into two groups:

i) relationships between specifications and real implementations (conformance); and

ii) relationships between specifications alone (compliance, refinement, consistency and internal validity).

Conformance is a relation between a specification and a real implementation, such as an example of a product. It holds when specific requirements in the specification (the conformance requirements) are met by the implementation. Conformance assessment is the process through which this relation is determined.

Compliance is a relation between two specifications, A and B, that holds when specification A makes requirements which are all fulfilled by specification B (when B complies with A).

Conformance of a real implementation is not always assessed against the "lowest level" (i.e. implementation) specification in a product's development process. It is possible for a "higher level" specification to be used (for example, the one whose refinement resulted in the implementation specification). In either case, the correspondence between consecutive specifications can be important. One such correspondence is determined by the rules of refinement.

Two specifications are related by the refinement relation when one is a refinement of the other and all the products that could conform to the refinement also conform to the specification from which it was refined. This ensures that all the constraints of the more generic specification are present in the refined one.

Specifications are not always related by being derived from the same set of requirements. Sometimes they have been developed from two or more quite separate points of view (e.g. separate viewpoint language specifications). In this case, consistency between the specifications can be an issue – it is important that the requirements of one specification do not contradict those of another. Consistency is a relation between two specifications that holds when it is possible for at least one example of a product to exist that can conform to both of the specifications.

A specification is valid when there are no conflicts between its properties and those implicit properties required of the specification (e.g. a protocol specification may be expected to be free from deadlock), and when there is at least one example of a product that could conform to it (i.e. it is not self-contradictory).

Conformance assessment is the determination of these relationships either by testing (conformance) or by specification checking (compliance, refinement verification, consistency checking and internal consistency checking).

9.3 Conformance points and related concepts

When the conformance of a realization of an ODP specification is assessed using conformance testing, its behaviour is evaluated (by delivering stimuli and monitoring any resulting events) at specific (interaction) points. The points used are called "conformance points", and they are usually chosen from a number of such points whose location is specified in the RM-ODP Architecture. These potential conformance points are termed reference points.

In order to comply with the RM-ODP, standard ODP specifications are obliged to contain a conformance statement which must (amongst other things) state which reference points should be used during conformance testing. It is intended that every conformance point given in a specification will be one of the reference points defined in the Architecture.

In addition to the concepts of conformance point, reference point and interaction point outlined above, the OSI conformance testing methodology and framework standard [ISO/IEC 9646] also defines the notion of a Point of Control and Observation (PCO), which is not the point at which conforming behaviour is defined to exist (i.e. a conformance point), but the point at which the behaviour at a conformance point is controlled and observed.

A conformance point is specified in ODP specifications to which conformance is likely to be claimed. A point of control and observation is specified when documenting a specific means of testing the implementation. Different methods may involve the identification of different sets of PCOs.

The processing system used to perform testing can be described using ODP nomenclature. It should be noted that PCOs of the tested application are essentially the conformance points of the system that tests it. The reference points of the testing system are potential PCOs, that may previously have been specified for consideration when defining a test method.

The Foundations and Architecture in ITU-T Rec. X.902 | ISO/IEC 10746-2 and ITU-T Rec. X.903 | ISO/IEC 10746-3 do not define PCOs or potential PCOs separately – the notions of the conformance points and reference points of the testing system can be used instead. Positioning potential PCOs only at reference points would ensure that testing systems are themselves ODP systems and would limit the range of test methods that might require specification. As the use of ODP becomes more widespread, since there will be no practical distinction between potential PCOs and reference points, the term PCO may then become redundant.

9.4 ODP conformance specifications

9.4.1 Level of abstraction

Conformance testers may require additional information when testing an implementation of some ODP specification. Such information is called Implementation Extra Information for Testing (IXIT) and it includes information that is required to relate the concepts in ODP Implementation Conformance Statements (ICS) to their realization in an implementation.

The level of abstraction at which a conformance point is specified has implications on the work required to provide an IXIT mapping, the constraints imposed on the implementation process, and the amount of implementation independence:

- There is a requirement for additional extra information about the specification to implementation mapping at higher levels of abstraction. Since the provision of this information and the implementation process both involve the linking of specification terms to implementation artefacts, the provision of the IXIT mapping can require a level of effort proportional to that required in the whole of the implementation process. For conformance points defined using a particularly high level of abstraction, this can represent an unreasonable cost and may make conformance testing unattractive.

- It is also noted that all PCOs must have some explicit and accessible realization in a real implementation, a feature not necessarily shared with other aspects of a specification.

  The specification of conformance points (for which PCOs must be allocated) therefore represents a constraint on the implementation process that will be greater than the larger the number of points defined. The less related to an implementation a conformance point's specification is (e.g. the higher its level of abstraction), the greater the inconvenience of providing an explicit representation in the real implementation.

- The use of a lower level of abstraction in the specification of a conformance point implies that fewer details have been "abstracted away" from an implementation and therefore that the specification is less implementation-independent. The greater detail also implies that a greater effort may be required in the conformance assessment process to assess each detail (although less interpretation is necessary in providing IXIT).

9.4.2 Use of multiple reference points

The definition of many, as opposed to a single, conformance points in a specification can be required for purposes other than simply increasing the number of prescriptive aspects of a specification.

An object's specification may define a number of different types of interface to its environment. For example, it may have interfaces to people, to other objects, to communication mechanisms or to storage mechanisms. Each of these can be characterized by reference points of different classes (perceptual, programmatic, interworking and interchange classes), each of which has different conformance testing consequences. This implies that different conformance points will need to be identified when interfaces of these different types are present.

Even when only one type of interface is considered, the implementation of an object may be intended to be accessible at physically or logically separated points. Associating separate conformance points with each of these recognizes these details of the implementation and so lowers the level of abstraction at which the conformance points are specified

(because more details of an implementation are prescribed), which has the implications outlined in 9.4.1 above. Separate conformance points also make the "intention" to have separate points of access a prescriptive, rather than a descriptive detail (and thus ensure that the intention is realized).

When an object is specified in terms of a number of components with different interfaces between each other, it is not always clear whether the internal detail exists to prescribe the external behaviour of the object by implication or whether it is intended to be prescriptive (mandating the way in which the object should be built). However, when the components are considered to be a prescriptive detail, additional conformance points need to be defined (e.g. at each of the inter-component interfaces).

9.5 Conformance implications of viewpoint languages

The use of ODP viewpoint languages in ODP specifications has a number of implications:

- conformance testers are expected to evaluate the effect of specifications in terms of the engineering specification of an implementation under test, which, together with other factors requires extra information for testing to be provided by a testing laboratory client, including:
  - i) IXIT+ICS relating the implementation of the concepts and structures of an implementation's enterprise specification to the implementation of its engineering specification;
  - ii) IXIT+ICS relating the implementation of the concepts and structures of an implementation's information specification to the implementation of its engineering specification;
  - iii) IXIT+ICS relating the implementation of the concepts and structures of an implementation's computational specification to the implementation of its engineering specification;
  - iv) IXIT+ICS relating the implementation of the concepts and structures of an implementation's engineering specification to the implementation of the choices made in its technology specification (this is to be provided as part of the technology specification);
- each viewpoint language enables the separate specification of different types of requirement so that, for example, business goals, system design and use of technology can be addressed separately – this implies that parallel development of separate specifications is possible;
- checking parallel development of specifications requires the consistency of specifications to be evaluated against each other from time to time during a development process.

Because specifications provided from different viewpoints may be independent and may therefore not be related by refinement, refinement checking does not supply a complete check on the parallel progress of the different specifications during their development. Some such check is nonetheless clearly desirable. This requirement is met by consistency checking (see 8.7). The objects described by viewpoint specifications are often from the same universe of discourse: the information object in an information specification may appear as an argument to a function in the interface to a computational object, for example; the conformance points of the enterprise specification may reappear as conformance points in the computational and engineering specifications; and so on.

9.6 Conformance assessment activities

The activities for use during the conformance assessment process include:

- refinement checking, between specifications;
- internal validity checking of a single specification;
- consistency checking of a number of specifications;
- testing of a realization or animation, between a real implementation and a specification.

10 Management of ODP systems

An ODP system comprises a number of ODP applications together with supporting services. The supporting services include services like processing, file storage, user access and communications as provided by a traditional operating system in a centralized system, but they also include services necessary for system distribution such as directory and name management, trading and the services supporting distribution transparencies. All these services need to be managed, together with the ODP applications: the nature of the management functions needed to do this will depend on the services or applications concerned.

Management of an ODP system will itself be carried out by one or more management applications interacting with the system services and ODP applications through interfaces offering a level of management granularity determined by the specification of the management application of concern. Where necessary, the management application will interact with the ODP functions via the management functions defined for them.

Management applications in an ODP system will reflect interrelationships between management functions, for example, the fact that it may be necessary to perform reconfiguration in order to maintain QOS demonstrates that management functions are inter-related. Accounting is also normally included in management applications, reflecting the fact that most services need internal accounting for management purposes even if users are not billed for the service.

Security functions are not included in the set of management functions even though management interactions are expected to be subject to policy control, for example, authentication and access control are generally essential to prevent unauthorized users or managers from performing management actions on components of a service or application. Thus, all services in an ODP environment need the use of a security service just as they need the use of a communications service, nevertheless security management is not a generic management service required for managing other services.

10.1 Management domains

In order to cope with the complexity of management and the issues of scale, especially within large ODP systems, it is essential to provide a common framework for partitioning overall management. Within the ODP environment, there is a multiplicity of coexisting management views and boundaries of responsibility, each based on different structuring criteria.

Management domains provide a flexible and pragmatic means of specifying boundaries of management responsibility and authority that reflect these different views. A domain identifies a set of objects, each of which is related by a characterizing relationship to a domain controlling object. The member objects may be resources, workstations, modems, processes, etc., depending on the purpose for which a particular management domain is defined. The controlling object has an attribute which identifies member objects. An object is referred to as a member object if its identity is known to the management domain controlling object.

Management domains permit a set of managed objects to be controlled under a common policy, providing a basis for coping with the complexity of large scale ODP systems. They simplify management activity because policy and the set membership can be modified through interactions with a single object – the management domain controlling object – rather than by forcing managers to interact individually with the multiplicity of managed objects within the environment.

Management domains do not encapsulate the member objects – external objects may interact directly with an object in a domain. Management domains are persistent even if, at some points in time, they do not contain any object since it must be possible to create an empty management domain and later include objects in it.

10.2 Management policy

In enterprise terms, the characterizing relation for a domain embodies the policy associated with the domain. Thus, management domains provide the means for specifying management policy for a group of managed objects rather than having to do this for each individual object. The overall management objective and external constraints relating to laws (e.g. a national Data Protection Act), regulations, or higher level policies are two aspects of the policy for a management domain. These examples show that it may be difficult to specify some policies formally.

Internal constraints place restrictions on the operations which can be performed on objects in a management domain. These can be expressed declaratively in terms of obligations on potential members of the domain.

An important aspect of management policy is to specify what management operations managers may perform on the objects they manage. An access rule is an authority relationship which specifies the set of permitted interactions between a domain of managers and a domain of managed objects. For example, all members of the domain “SysProgrammers” are allowed to start and stop the objects in the “DepartmentServices” domain. The permitted interactions may be a subset of the management interactions defined by the interfaces to the objects in the domain.

10.3 Modelling management structures

Domain relationships can be used to model management structures. Two management domains are defined to be disjoint if they have no member objects in common. Two management domains overlap if there are objects which are members of both domains. An example is the shared management of a gateway interconnecting two networks by the management centres of each network. This can be accomplished by referencing the object from both management domains.

Implicit overlap may occur between two management domains containing managed objects of different type but referring to the same real-world entity. An example is scheduling and maintenance domains in which putting a workstation out of service in the maintenance domain makes it unavailable in the scheduling domain. Implicit overlap is likely to occur where there is a functional partitioning of management into different management domains.

11 The use of standards in ODP systems

The objective of this clause is to illustrate, by means of a simple example, the use of standards, other than specifically *ODP standards*, in an ODP system and where such use is defined in an ODP system specification. In order to do this, the example describes, in outline, the content of each viewpoint specification of an example system and, for each viewpoint specification, discusses the types of standard that are relevant to meeting the requirements identified by it.

The example system is illustrated in Figure 16. This represents a server system with a local operator, linked by a telecommunications facility to a workstation client system with two local operators. The system description considers one workstation application that provides access for its local operators to a picture provider application on the server system. It is assumed that both the server system and the workstation provide other services and that organizations to which they belong are distinct, i.e. subject to separate administrative policies.

The viewpoint descriptions identify requirements for common specifications. These common specifications will, in general, correspond to standards at different levels.

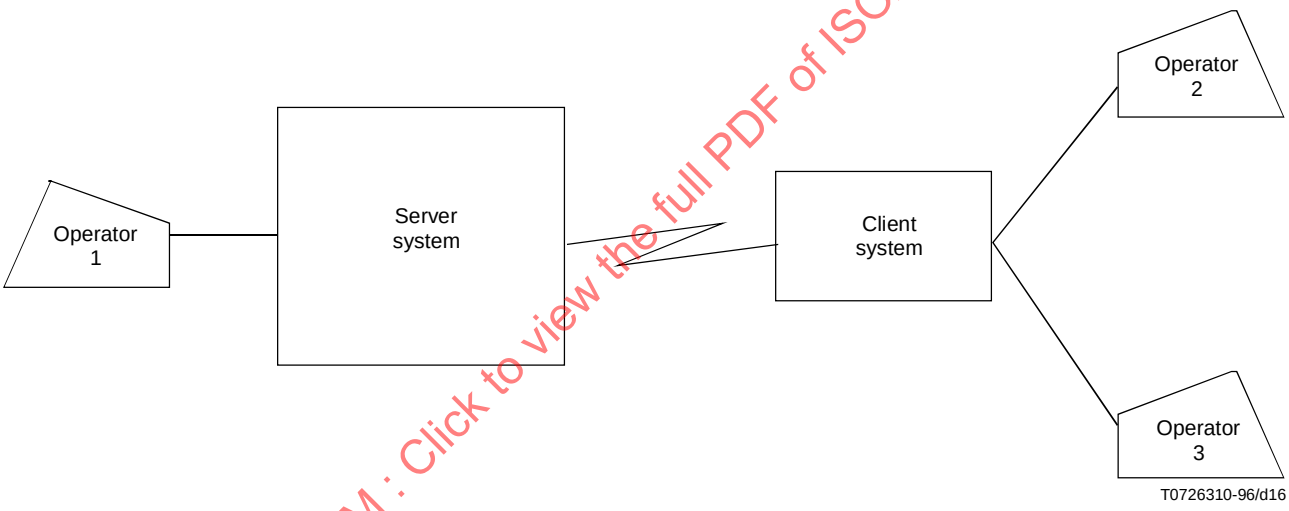


Figure 16 – Example system configuration

11.1 Enterprise viewpoint

11.1.1 Enterprise specification

Figure 17 illustrates the object model for the enterprise specification of the system represented in Figure 16, where:

- Picture user n is an enterprise object representing a person and is fulfilling the role “Picture user n”;
- Picture provider is an enterprise object representing an IT system and is fulfilling the role “Picture provider”;
- Picture accessor is an enterprise object representing an IT system and is fulfilling the role “Picture accessor”.

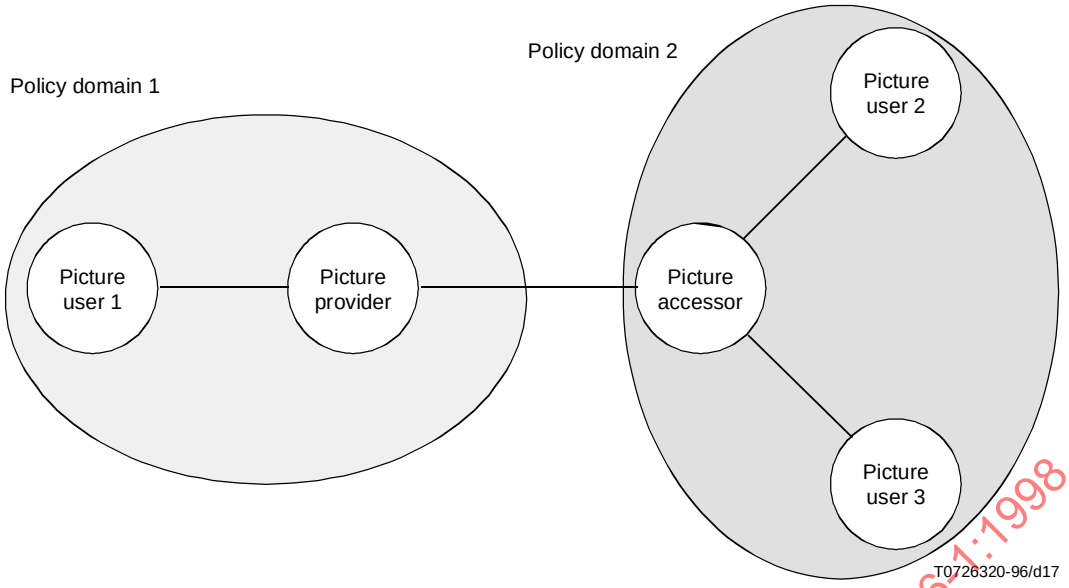


Figure 17 – Enterprise viewpoint

The objective of the enterprise represented by the configuration of enterprise objects governs the nature of the behaviour associated with the roles fulfilled by those objects. For example, meeting the objective might only require that there is a fixed picture display for Picture users 2 and 3, or it might require that the Picture accessor supports manipulation of the picture information, addition of picture information and feedback to Picture provider.

Thus, the enterprise specification defines the objective of the configuration of enterprise objects, and hence:

- the picture information needs of Picture users 1, 2 and 3;
- the interactions of Picture user 1 with Picture provider;
- the interactions of Picture user 2 and Picture user 3 with Picture accessor;
- the interactions between Picture accessor and Picture provider;
- the policies (including security) governing the interactions between enterprise objects;
- the QOS requirements for Picture users 1, 2 and 3.

11.1.2 The application of standards

In the enterprise description, common specifications could be developed for industry or business function specifics, such as a *security policy*, that must be upheld in terms of access controls and strength of protection. Such standards would relate to the business and application area of concern.

11.2 Information viewpoint

11.2.1 Information specification

The system is visible from the information viewpoint in terms of:

- the information object classes involved in the application;
- the information activities (changes of state of information objects) that constitute the application;
- the constraints on the changes of state that can take place in the information objects.

The information specification includes:

- Specification of the information object classes themselves, for example:
  - picture information object templates that determine the picture information object classes that are available;
  - picture display object templates that determine the picture display object classes that can be available. A picture display object template defines a composition of picture information objects;
  - a request information object template, where a request information object comprises the information necessary to request a display, including control information (e.g. for security);
  - an *access control information* object comprising the information necessary to validate a request information object.
- Constraints on the configurations of information objects, for example:
  - the set of template classes of picture information objects for the Picture accessor is the same as that for the Picture provider;
  - the set of template classes of picture display objects for Picture user 1 is a subset of the set for the Picture provider;
  - the sets of template classes of picture display objects for Picture users 2 and 3 are subsets of the set for the Picture accessor.

For the system concerned, a (simplified) example of an information activity could be:

- creation of request information object with status invalidated;
- interaction of request information object and access control information object and change of status of request information object to validated;
- interaction of request information object with picture information objects and creation of picture display objects.

11.2.2 The application of standards

In the information description, common specifications could be developed for information objects. Standards would relate to the business and application area of concern.

11.3 Computational viewpoint

11.3.1 Computational specification

Figure 18 illustrates the object model for the computational specification of the system represented in Figure 16. The object model comprises a configuration of:

- computational objects User 1, 2 and 3, Picture display 1 and 2, Picture compose and Picture database;
- primitive bindings, for which there is no explicit statement of environment contracts, between interfaces of:
  - User 1 and Picture display 1 (up_1);
  - Users 2 and 3, and Picture display 2 (up_2 and up_3);
  - Picture display 1 and Picture compose (pc);
  - Picture database and Picture compose (dc);
- a compound binding involving a binding object (Binding) between the interfaces of Picture display 2 and Picture compose, with a statement of the environment contracts;
- the activities specified for the configuration of computational objects that realize the requirements specified in the enterprise and information specifications. The interfaces are reference points for conformance in ODP terms.

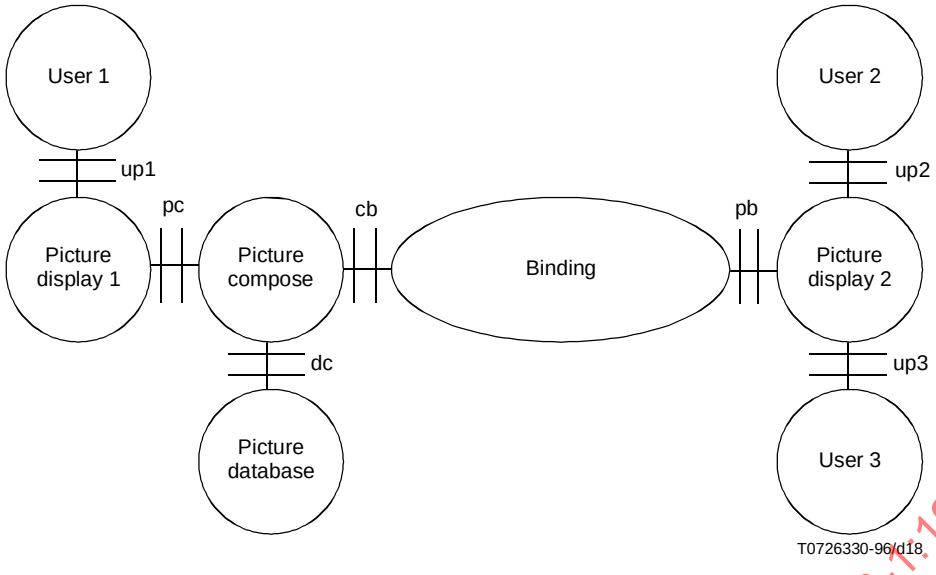


Figure 18 – Computational viewpoint

The system is visible from the computational viewpoint in terms of:

- specifications of the computational objects in terms of abstract specifications of the operations and behaviour that they support at their interfaces;
- a specification of the binding object template – this includes:
  - an abstract specification of the interface operations involved;
  - a specification of environmental contracts consistent with the enterprise QOS requirements;
- a specification of the abstract data types that correspond to the information objects identified from the information viewpoint;
- specifications of the activities that can occur to support the application.

11.3.2 The application of standards

In the computational description, common specifications could be developed for the interfaces identified in Figure 18, and would be defined in terms of abstract operations and abstract data types (corresponding to the information objects in the information description). Standards for the abstract operations could be common to several areas of application; standards for the abstract data types would be defined for the application area of concern. For example, a specification which is based on OMG/CORBA and is described in OMG/CORBA IDL is a computational description.

11.4 Engineering viewpoint

11.4.1 Engineering specification

Figure 19 illustrates (part of) the object model for the engineering specification of the system represented in Figure 16. In this object model:

- the basic engineering object User 1 represents Operator 1;
- the basic engineering objects Picture display, Picture compose and Picture database correspond to the computational objects Picture display, Picture compose and Picture database;

- the Stub, Binder and Protocol objects comprise part of a channel corresponding to the binding object in the Computational description;
- Node corresponds to the Server system in Figure 16 (although Figure 19 only illustrates part of the full configuration of capsules, clusters, channels, etc., for the Node).

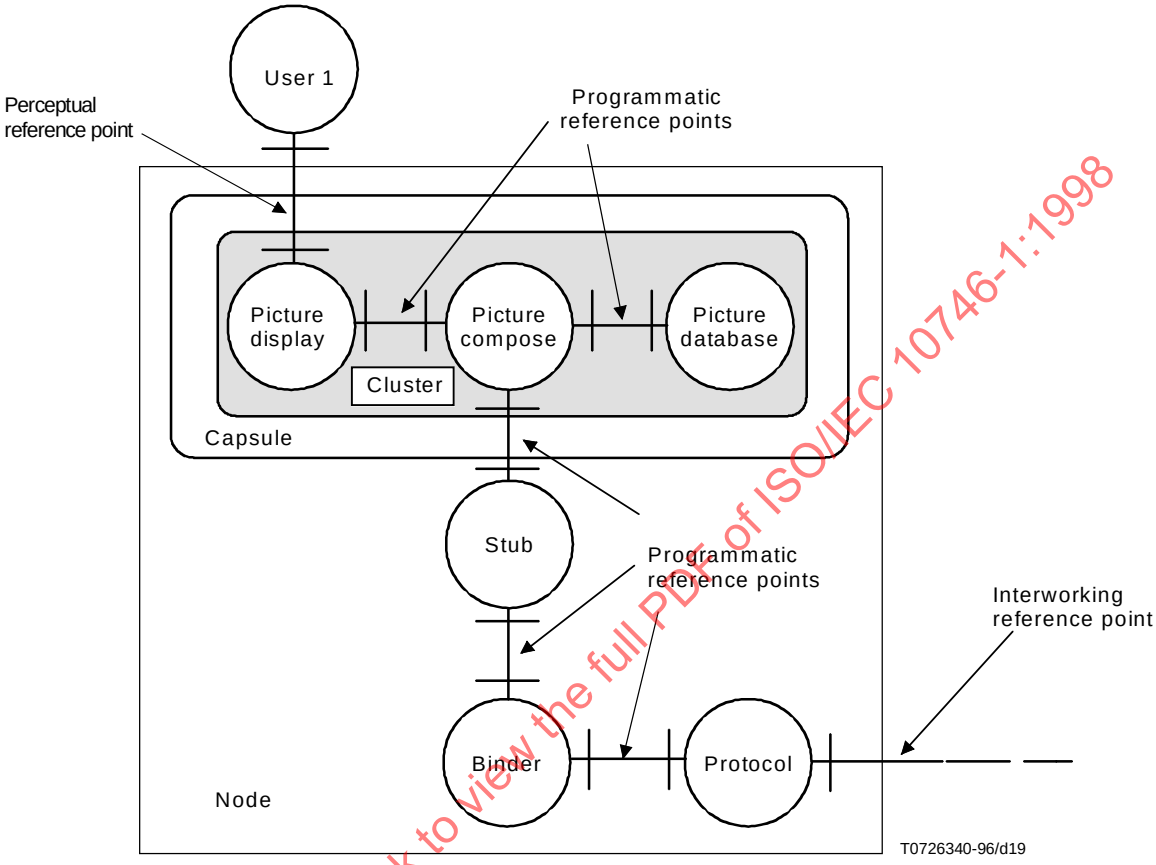


Figure 19 – Engineering viewpoint (part)

The system is visible from the engineering viewpoint in terms of:

- specifications of the behaviour of the engineering objects in the channel corresponding to the binding object in the computational viewpoint including:
  - specification of the protocols at the *interworking reference point* between the protocol objects (or between protocol object and interceptor);
  - specification of the concrete representation of the abstract data types identified in the computational description;
  - QOS requirements;
- for each reference point, a specification of the syntax in terms of which the behaviour at that reference point is expressed;
- constraints between the behaviour at the reference points that reflect the specified computational activities.

11.4.2 The application of standards

In the engineering description, common specifications could be developed for application at the reference points identified in Figure 19:

- for protocol specifications, and abstract and concrete transfer syntax specifications, that apply at the interworking reference point within the channel (that corresponds to the binding object in the Computational description), for example:
  - OSI profiles (e.g. an RDA A-profile, an appropriate T-profile and appropriate F-profiles) can be applied for the protocols and for the abstract and concrete transfer syntaxes for the establishment and maintenance of the channel;
  - standards and F-profiles for the abstract and concrete syntaxes corresponding to the abstract data types in the Computational description must be defined for the application area of concern.

    NOTE – The protocols and concrete syntaxes at the interworking reference point include those necessary to establish and maintain the channel as well as those that correspond to the abstract operations and abstract data types specified for the binding object.
- for API specifications (including abstract syntax specifications for data) that apply at the programmatic reference points, for example:
  - an SQL profile can be applied at the reference point between Picture compose and Picture database;
  - an API profile for the service supported by the RDA profile can be applied at the reference point between Picture compose and Stub;
  - an API profile for a windowing service can be applied at the reference point between Picture compose and Picture display - this could include graphics standards;
- for HCI specifications that apply at the *perceptual reference point* – a GUI standard for example.

11.5 Technology viewpoint

11.5.1 Technology specification

The system is visible in the technology viewpoint in terms of statements by the supplier about the conformance of his system. This will be expressed by:

- identifying conformance points for the system supplied that correspond to reference points in the Engineering description;
- stating the extra information for testing the conformance of the system to the behaviour specified for those conformance points.

Note that the supplier need not be required to make all reference points visible as conformance points.

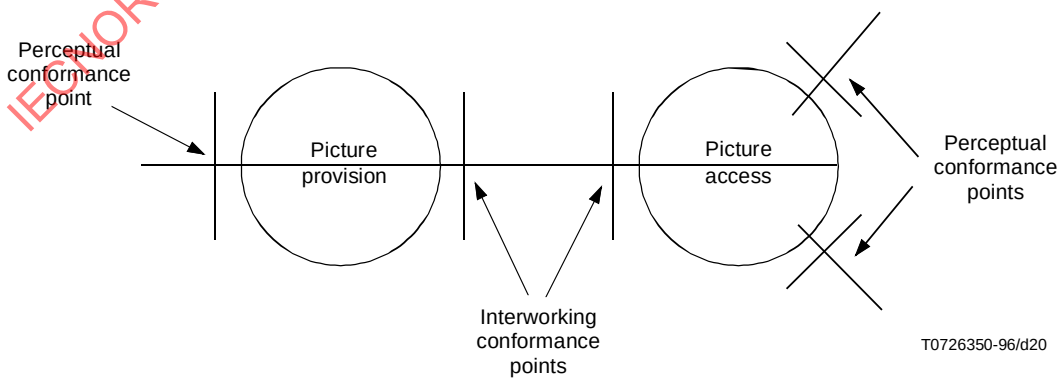


Figure 20 – Technology viewpoint (part)

11.5.2 The application of standards

In the technology description:

- an implementation of a specific OSI protocol is an example of the application of OSI standards;
- the conformance points identify the points at which the behaviour of the system can be observed;
- the standards that apply at those conformance points specify the syntax and order of exchanges in terms of which the behaviour is expressed.

There are, in addition, constraints placed on the behaviour at the conformance points by the requirement that the behaviour of the system is consistent with the standards specified for the other reference points in the Engineering description.

12 Examples of ODP specifications

This clause describes several examples of the use in system specification of the concepts and rules described in ITU-T Rec. X.902 | ISO/IEC 10746-2 and ITU-T Rec. X.903 | ISO/IEC 10746-3. The examples are simplifications and incomplete descriptions of real life systems but they serve to provide insight into the use of the ODP framework, they illustrate at a high level the application of key concepts from each viewpoint language and they illustrate the relationship between the viewpoint descriptions.

The example in 12.1 uses RM-ODP concepts and rules to design a **Multimedia Conferencing System** (MMCS) in the five viewpoints. The MMCS of concern allows real-time interworking between several users using multimedia information like text, video and audio.

The example in 12.2 specifies **multiparty audio/video exchange** in distributed systems – a specific component of an MMCS. It uses the "stream binding" concept as a basis for multiparty audio and video flow exchange, and provides a specification using the five ODP viewpoint languages. Special attention is paid to correspondences between the five viewpoint specifications, to ensure consistency between the specifications.

The example in 12.3 positions **management concepts** in the RM-ODP framework.

The example in 12.4 gives an overview of specification of a **Distributed Database.**

The following concepts are illustrated in the different viewpoint specifications of the examples:

Enterprise:

- communication/federation;
- the association of roles with enterprise objects;
- contract, template and policy.

Information:

- static schema;
- dynamic schema;
- invariant schema.

Computational:

- computational object specification including environment contract, behaviour;
- operation and stream interface specification;
- binding concept;
- interface references and interaction rules;
- transparencies.

Engineering:

- channel establishment (protocol, binder and stub objects);
- use of the engineering structuring rules and functions to specify the infrastructure compliant with the enterprise, information and computational specifications.

Technology:

- choices of specific hardware and software components compliant with the other viewpoint specifications;
- identification of conformance points implied by the technology choices.

12.1 Multimedia Conferencing System

12.1.1 Introduction

The Multimedia Conferencing System (MMCS) allows real-time interworking between several Users using multimedia information like text, video and audio. The service enables a group of persons that are physically distributed, to work together on a (multimedia) document and to communicate with one another. During a session new participants may join the conference or current participants may leave.

Furthermore, the service gives a User control over several service attributes, e.g. desired information types(s), quality of an information type, etc. The MMCS also provides a framework for various applications which have to cooperate.

Applications like video/audio conferencing, joint editing and electronic mail should be integrated from the Users' perspective. Extensions to the MMCS should be possible to achieve openness.

The MMCS configuration in Figure 21 may consist of workstations for representing video, text and audio. Furthermore cameras, microphones and a multimedia database (possibly distributed) are connected to a wide area network.

To specify this example, the relevant concepts and rules to specify MMCS will be applied for each ODP viewpoint. It is beyond the scope of this example to describe all aspects of a MMCS but rather to illustrate each viewpoint language to specify an open distributed application or an open distributed system. An object oriented analysis method OMT [Rumbaugh 91] is used to express the ODP enterprise and information specifications.

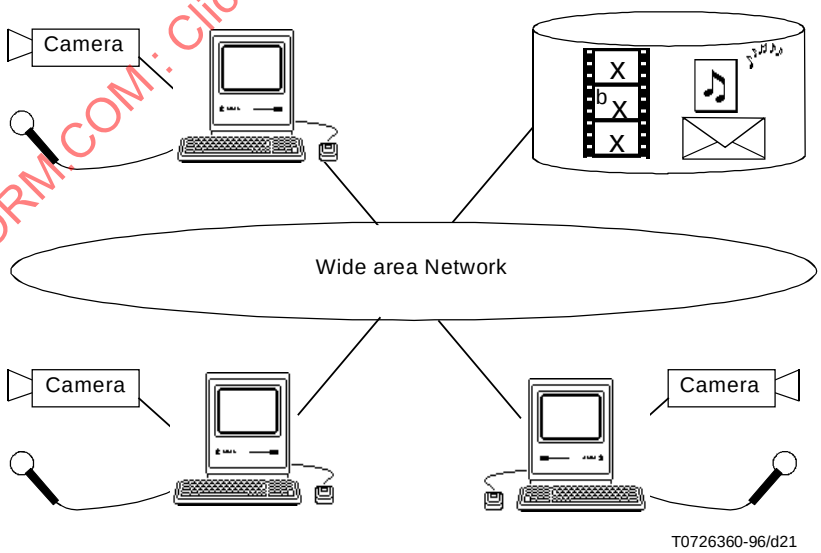


Figure 21 – Configuration of a Multimedia Conferencing System

12.1.2 Enterprise specification

The enterprise specification describes the objectives, policies and requirements of the concerned MMCS. The requirements and policies of the MMCS service are derived from the parties involved. They can be classified according to their role:

- User: A person or machine who uses services in order to satisfy some communication needs.
- Customer or subscriber: A person or organization that contracts services offered by Service providers.
- Service provider: An organization that commercially manages services offered to Customers according to contractual agreement.

To structure the requirements of a particular service according to roles, specialization of the generic roles introduced above may be needed. For example, for the MMCS the following two types of Users can be distinguished: session participant and session leader.

Then, the question arises "what should be described for each role involved in the service?". The RM-ODP provides indications which cover a broad range of policies and rules that are of interest for the description of distributed services. Generic rules and policies of interest are: resource usage, domain rules, conflict resolution rules, organization rules, business rules, transfer rules, security rules, QOS rules and management rules.

Figure 22 illustrates a simple enterprise specification using ODP enterprise concepts and expressed in OMT graphical notation.

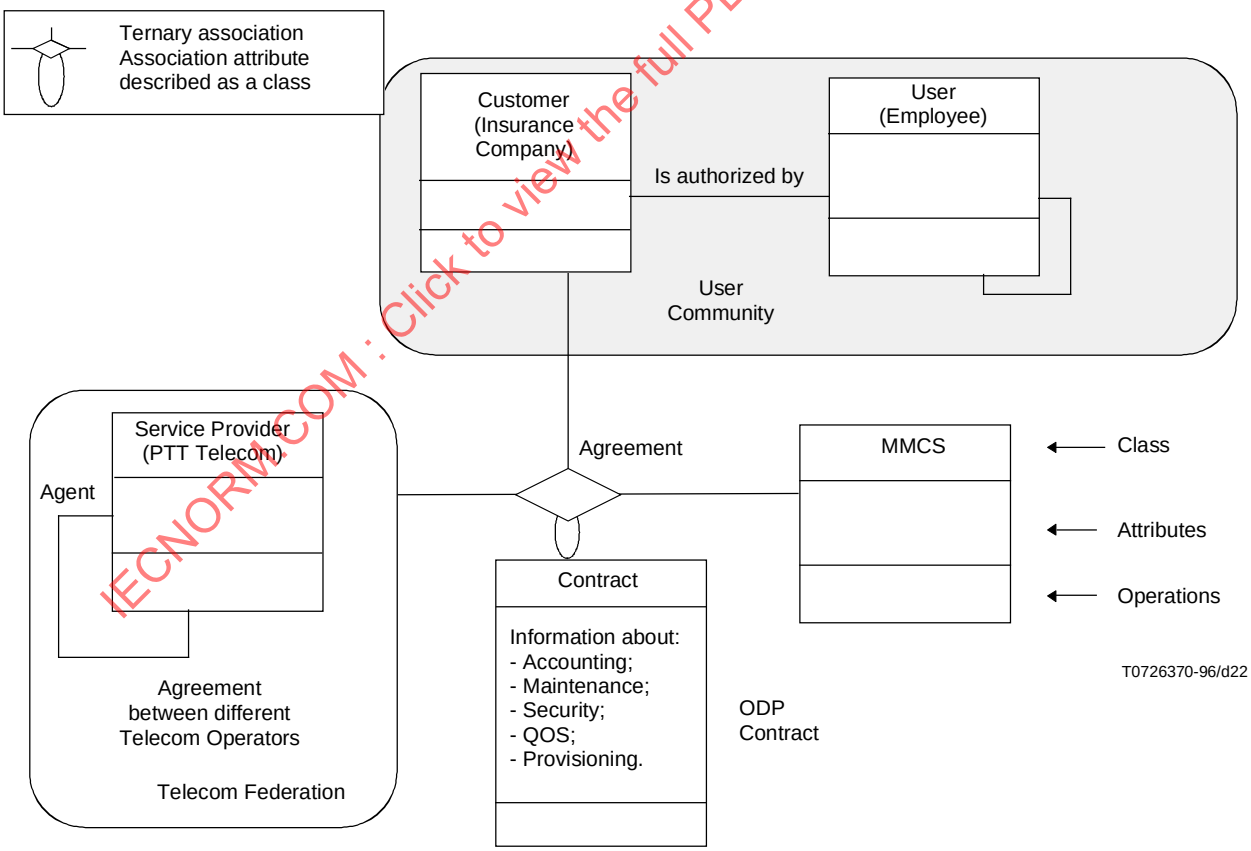


Figure 22 – ODP Enterprise specification using OMT graphical notation

A multimedia conferencing service takes place in a number of different countries. This can be interpreted as a number of Users interacting with their home Telecom Operator (Service Provider). For the purpose of expressing the MMCS in ODP terms, this can be represented by using the community and federation concepts. It can be stated that the Users exist in the User community and the Telecom Operators belong to a federation. The relationships and interactions between the two can be explored and, with respect to MMCS, agreement established by means of a contract.

The User, Service Provider and MMCS can all be described, in ODP terms, as enterprise objects with associated roles. For example, the User can be engaged in performative actions with the MMCS via the Customer and vice versa. These interactions change the so-called obligation, permission and prohibition relations between the User and MMCS. The group of Users interact with each other to form an ODP community because the Users have a common contract of obligation between roles fulfilled by the enterprise objects and a set of activities.

12.1.3 Information specification

In the information specification, the semantics and requirements for the processing of the service information are specified.

This is done using the *schema* concept. A local schema for each User role is defined, and a global schema, that represents information that holds for every User role and represents information concerning the service, is specified. Note that this information specification provides an example for a static schema, but does not provide examples of invariant and dynamic schemata.

Focusing on the MMCS class identified in the enterprise specification, a static schema for the system at the time when a session exists is represented by the OMT information specification shown in Figure 23.

For the Customer configuration parameters of MMCS are described. Also, information is included about limits to bandwidth allocation, list of registered end-Users enabled to initiate a teleconference, list of options allowed for registered end-Users, etc. Such Customer information can be regarded as attributes of a Customer in an OMT object model. A User information object is connected to a Customer information object in the sense that it can participate in a conference only if in the Enterprise specification the corresponding end-User is authorized by the corresponding Customer. The Leading user is the User controlling the conference.

12.1.4 Computational specification

The information specification is developed to be consistent with the enterprise and information specifications. The mapping between information objects and computational objects is not necessarily one-to-one. The information specification is essentially different from a computational specification: in particular, computational objects are specified in terms of interfaces and information objects are not.

The correspondences between the information specification and the computational specification must be specified in each case so that consistency between the specifications can be ascertained. This is an important task to be performed by the service designer. Figure 24 illustrates the mapping for MMCS.

The OMT analysis is used to identify and design the computational objects. Those computational objects are identified while regrouping elements that are functionally linked. The OMT analysis also enables some choices to be made for the engineering configuration and the technological support.

The grouping of classes into computational objects is a decision taken by the service designer not concerned with distribution aspects:

- the OMT objects, classes and associations concerning the User and the Leading user are taken into account for the design of the computational User object;
- the OMT objects, classes and associations concerning the conference (e.g. MMCS, Session, Subsession) are taken into account for the design of the computational conference object;
- the OMT objects, classes and associations concerning the connection (e.g. connection manager) are taken into account for the design of the computational bindings and will be helpful in choosing the engineering configuration (e.g. multicast facilities).

For each computational object identified resulting from this analysis it is necessary to define interfaces for its interactions with other computational objects (e.g. conference operations or operations to send audio/video flows). In general, objects and interfaces can be graphically represented as shown in Figure 25.

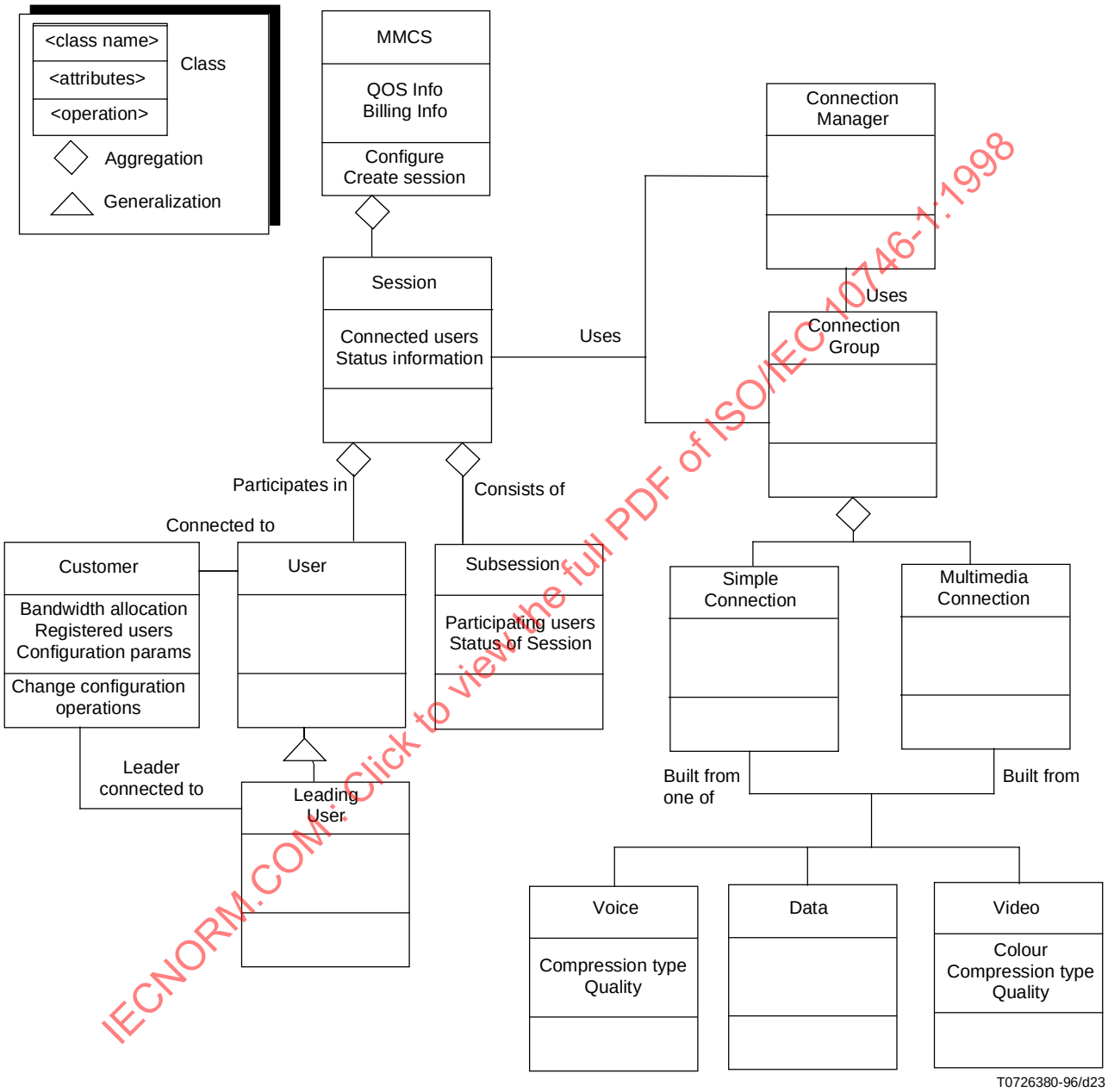


Figure 23 – Information specification

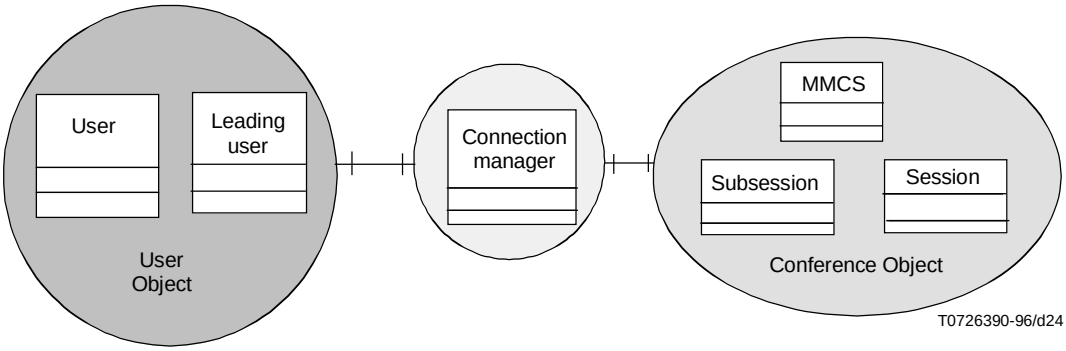


Figure 24 – Computational configuration of MMCS

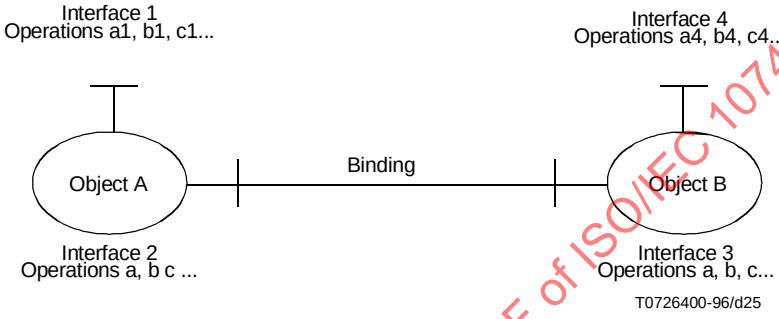


Figure 25 – A computational representation of objects and interfaces

Focusing on interactions that are possible via binding objects, these are established by explicit binding actions that allow the User to specify the binding required between the computational objects. A binding object resulting from explicit binding can support operation interchange between computational objects or a stream if the interchange concerns continuous information flows.

Figure 26 shows the graphical notation to represent an explicit binding, in this case a stream binding. The binding object controls and manages the interactions between the computational objects interfaces it encompasses. Control operations are performed through the binding control interface.

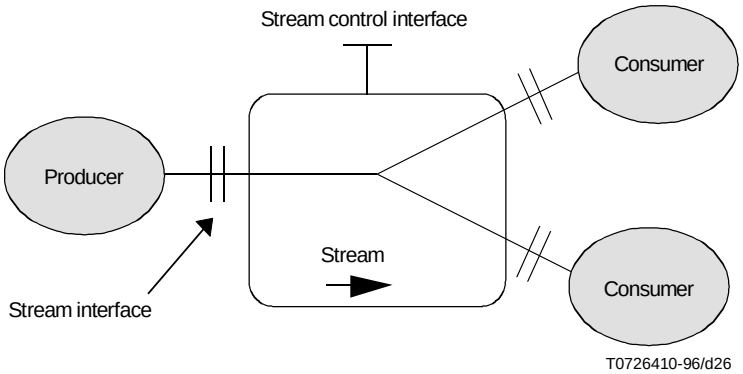


Figure 26 – Representation of an explicit binding (e.g. stream)

The computational representation of computational objects involved in audio/video interchange in MMCS is shown in Figure 27. Two main objects are identified: the User object and the Conference object.

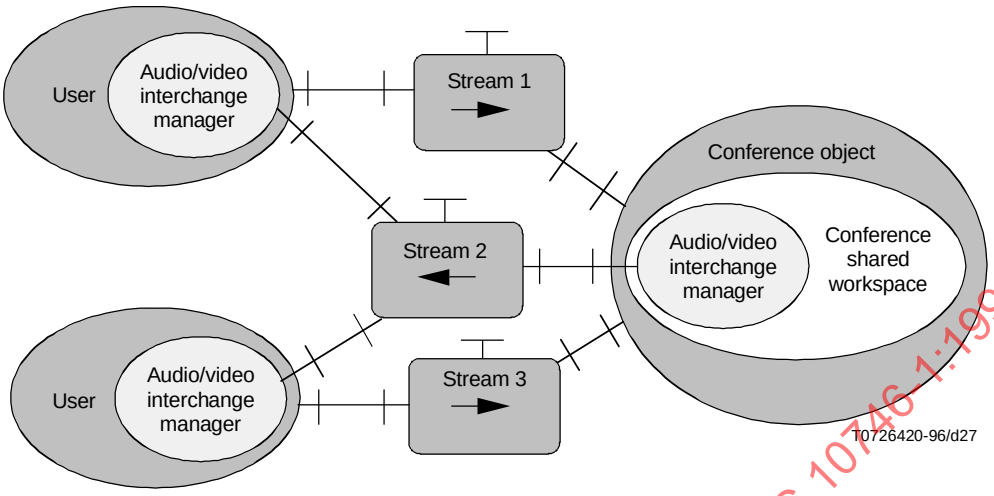


Figure 27 – MMCS configuration of objects involved in audio/video interchange

The User object enables the connection of end-Users to the conference and provides the tools necessary for end-Users to interact with each other via the conference object and other User objects. The User object provides end-Users with operations like joining the conference, editing facilities, tools to exchange video and audio.

The conference object contains the functionalities required for a multimedia conference. It contains, in particular, a conference shared workspace which is in charge of dispatching audio and video flows between User objects.

The audio/video interchange managers are in charge of sending and receiving audio/video during the conference.

Stream 1 and stream 3 objects represent the audio/video flow from User objects to the shared workspace. Stream 2 represents multicasting of audio/video flows to all User objects.

Different control actions on the flows, like dynamic control of QOS and synchronization between audio and video flows, are performed through the stream control interfaces.

12.1.5 Engineering specification

The ODP engineering language enables the modelling of the service machine that supports the execution of the computational specification. Figure 28 shows a simplified engineering architecture. The major element of this architecture is the nucleus that controls resource utilization and enables communication between different engineering objects. Some functions, called ODP functions, common to a broad range of distributed services like trading or management functions, are available for distributed applications.

The distributed service is composed of basic engineering objects that are the run-time representation (e.g. C++ executable piece of code) of a computational specification. A binding between objects located in different nuclei is reflected by means of a channel between those objects. Focusing on the audio/video managers and stream 2 of Figure 27, the corresponding engineering support is represented in Figure 28.

A multipoint channel (engineering representation of the stream) is established between the audio/video interchange managers. This channel is linked to the nuclei concerned.

The stub objects provide adaptation functions to support distribution transparency (e.g. data format conversion from one video signal coding to a different one).

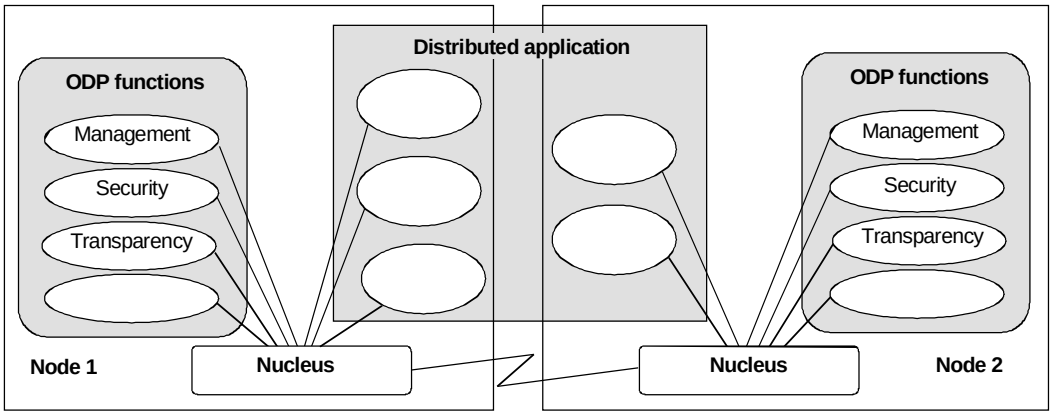


Figure 28 – Engineering support

The binding objects verify the compatibility of the interfaces to be linked and maintain the integrity of the binding between the audio/video interchange managers.

Protocol objects interact with each other to convey information necessary to support distribution transparent interactions between the audio/video interchange managers. The protocol objects are chosen to respect the QOS constraints (e.g. real-time, security).

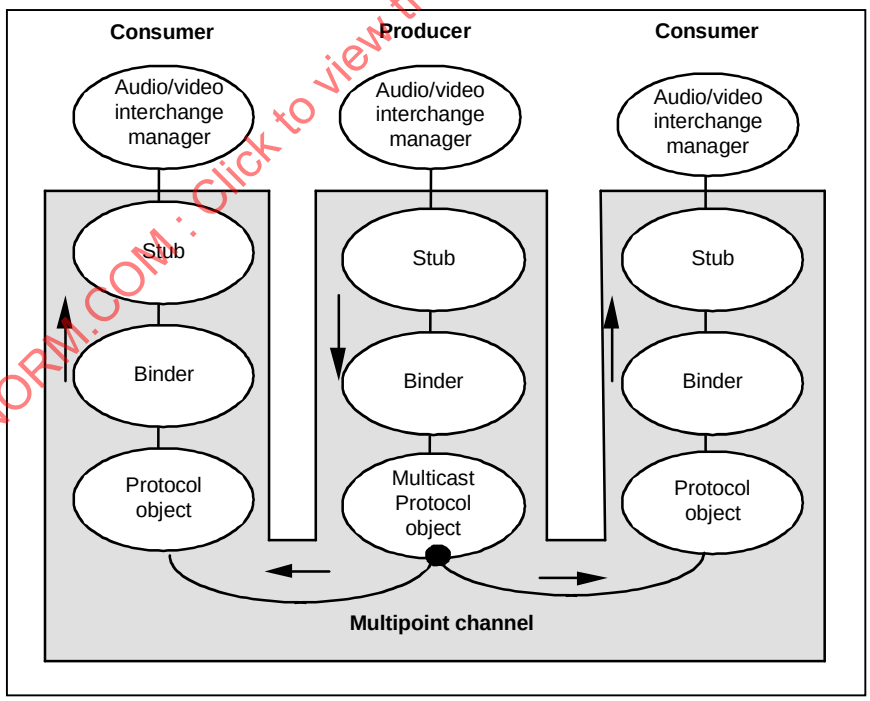


Figure 29 – Multipoint channel for stream 2

12.1.6 Technology specification

The technology specification of MMCS specifies the implementation of the system in terms of hardware and software components. Reference points identified in the other viewpoint specifications are defined as conformance points at which the behaviour of the MMCS can be verified.

The technology specification also expresses requirements for adequate workstations to represent video, audio and text. High speed and Wide Area Networks are required to support the communication needs by providing sufficient bandwidth to transport video/audio and text to the participants located in different places.

12.2 Multiparty audio/video stream binding

This example is concerned with the specification of the multiparty stream binding used in the system introduced in the previous example.

The approach followed in this example is to outline the problem domain in computational terms and then provide the five corresponding viewpoint specifications.

First, some additional concepts and rules applicable to this problem are defined. In the enterprise specification, the roles of *stakeholders*[1)] (user, customer, provider) are introduced to structure the problem domain in more detail. OMT [Rumbaugh 91] and IDL were used as particular notations to express information and computational specifications, respectively.

For the information specification, a number of relations are introduced between classes of the invariant schema. These relations are derived from a set of basic relations of OMT and are usually parameterized with text to make their meaning precise.

In the computational specification, the audio/video controller and dispatcher is introduced to handle the multiparty audio/video stream binding.

For the engineering specification, a specialized stream channel is introduced to transport continuous flows. A configuration of engineering objects is presented for the support of the multiparty audio/video stream binding, including objects to control and coordinate multiple stream channels. It would have been possible to specify the controller and dispatcher in a distributed way. However, this additional complexity was not introduced in this example.

The implementation described in the technology viewpoint is used to validate the modelling process along the ODP viewpoints. It is likely that more specific hardware/software would be used in a operating environment to meet, e.g. the strict performance requirements on multimedia applications.

12.2.1 General description

The exchange of continuous media in distributed multimedia applications is complex. For example, in a real-time multimedia conferencing application, the participants are separated geographically and communicate by exchanging real-time video and audio. The audio-visual exchange should be as natural and flexible as possible. This implies that requirements like lip synchronization and synchronization of display across multiple workstations need to be taken into account while specifying the multimedia conferencing application.

To fulfill these requirements, the multimedia conferencing application has stringent network performance and synchronization requirements on the exchange of audio and video flows. Furthermore, the number of exchanged flows and corresponding quality can change during the lifetime of the conference. This is due to the fact that the application provides control operations to join or leave the conference, and to modify the QOS of flows.

To address this complex functionality, RM-ODP defines the notion of binding object in the computational language. RM-ODP provides the theoretical computational concept without the specific refinements required in a given problem area. In this example the binding object is specified in terms of five ODP viewpoint specifications of a particular binding object, i.e. multiparty audio/video binding object have been provided [Gay 95]. This object manages the stream interfaces which are used for the real-time multiparty audio/video interactions. Also control operations can be performed on the multiparty binding object.

Figure 30 shows the computational representation of the multiparty audio/video binding object and its environment.

1) Stakeholder is a telecommunication concept that denotes an organization or person that has a commercial or regulatory interest in telecommunication services.

The rectangle in the middle denotes the multiparty audio/video binding object. Its environment (gray areas) consists of application and system parts and the supporting network infrastructure. The ⊥ symbols denote stream interfaces via which audio/video producers and consumers exchange audio and/or video (1). The multiparty audio/video binding object manages the interactions between the stream interfaces it encompasses. It encapsulates the mechanisms that are used for this, and it abstracts away from distribution aspects. The ⊥ symbol on top of the rectangle, denotes the stream control interface of the binding object. Via this interface, the multiparty audio/video binding object provides operations, 3 and 4, to the environment that controls its functioning.

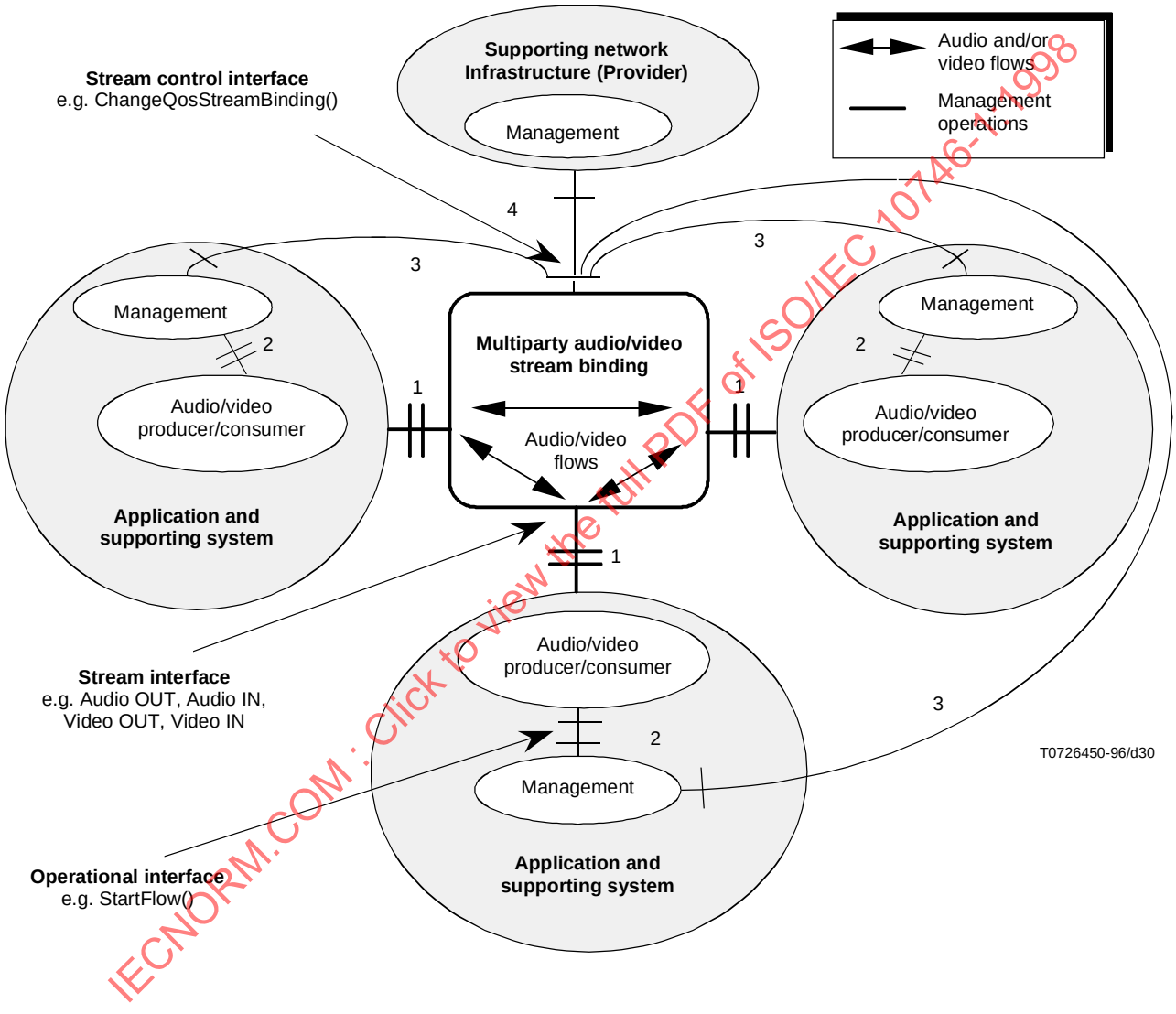


Figure 30 – Multiparty audio/video stream binding

12.2.2 Enterprise specification

The enterprise specification provides a description of the requirements and objectives that the environment imposes on the system to be designed. It justifies the design of a system. The enterprise concepts of enterprise objects fulfilling roles of *performative actions* are used to describe the multiparty audio/video binding.