

INTERNATIONAL STANDARD

ISO/IEC
10746-3

First edition
1996-09-15

Information technology — Open Distributed Processing — Reference Model: Architecture

*Technologies de l'information — Traitement distribué ouvert — Modèle de
référence: Architecture*



Reference number
ISO/IEC 10746-3:1996(E)

AVA **Documents**
Document Sharing Hub

Contents

	<i>Page</i>
1 Scope	1
2 Normative references	1
2.1 Identical Recommendations International Standards	1
2.2 Paired Recommendations International Standards equivalent in technical content	2
3 Definitions	2
3.1 Descriptive definitions	2
3.2 Abbreviations	3
4 Framework	3
4.1 Viewpoints	4
4.1.1 Concepts	4
4.1.2 Using viewpoints	4
4.2 ODP viewpoint languages	5
4.2.1 Concept	5
4.2.2 Using viewpoint languages	5
4.3 ODP functions	5
4.4 ODP distribution transparencies	5
4.4.1 Concepts	5
4.4.2 Using distribution transparency	6
4.5 Standards derived from the framework	6
4.6 Conformance	7
5 Enterprise language	7
5.1 Concepts	7
5.2 Structuring rules	7
5.3 Conformance and reference points	8
6 Information language	8
6.1 Concepts	8
6.2 Structuring rules	9
6.3 Conformance and reference points	9
7 Computational language	10
7.1 Concepts	10
7.2 Structuring rules	11
7.2.1 Naming rules	11
7.2.2 Interaction rules	12
7.2.2.1 Signal interaction rules	12
7.2.2.2 Stream interaction rules	12
7.2.2.3 Operation interaction rules	12

© ISO/IEC 1996

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

ISO/IEC Copyright Office • Case postale 56 • CH-1211 Genève 20 • Switzerland

Printed in Switzerland

	7.2.2.4	Parameter rules.....	13
	7.2.2.5	Flows, operations and signals	13
7.2.3		Binding rules.....	13
	7.2.3.1	Implicit binding rules for server operation interfaces	13
	7.2.3.2	Primitive binding rules.....	14
	7.2.3.3	Compound binding rules.....	14
7.2.4		Type rules	14
	7.2.4.1	Signature subtyping rules for signal interfaces	15
	7.2.4.2	Signature subtyping rules for stream interfaces	15
	7.2.4.3	Signature subtyping rules for operation interfaces.....	15
7.2.5		Template rules.....	16
	7.2.5.1	Computational object template rules.....	16
	7.2.5.2	Computational interface instantiation	16
	7.2.5.3	Computational object template instantiation.....	16
7.2.6		Failure rules	16
7.2.7		Portability rules	17
7.3		Conformance and reference points	17
8		Engineering language.....	17
8.1		Concepts.....	18
8.2		Structuring rules.....	19
	8.2.1	Channel rules	20
		8.2.1.1 Stubs.....	21
		8.2.1.2 Binders	22
		8.2.1.3 Protocol objects.....	22
		8.2.1.4 Interceptors	22
	8.2.2	Interface reference rules.....	22
	8.2.3	Distributed binding rules.....	24
	8.2.4	Relocation rules	24
	8.2.5	Cluster rules	24
	8.2.6	Capsule rules	25
	8.2.7	Node rules	26
	8.2.8	Application management rules.....	27
	8.2.9	Failure rules	28
8.3		Conformance and reference points	28
9		Technology language	29
9.1		Concepts.....	29
9.2		Structuring rules.....	29
9.3		Conformance and reference points	29
10		Consistency rules	29
	10.1	Computational and information specification correspondences	30
	10.2	Engineering and computational specification correspondences.....	30
11		ODP functions.....	31
12		Management functions	32
	12.1	Node management function	32
		12.1.1 Thread management.....	32
		12.1.2 Clock access and timer management	32
		12.1.3 Channel creation and interface location.....	32
		12.1.4 Capsule template instantiation and capsule deletion.....	33
	12.2	Object management function	33
	12.3	Cluster management function	33
		12.3.1 Cluster checkpoint	34
		12.3.2 Cluster deletion, deactivation and failure.....	34
		12.3.3 Cluster reactivation and recovery	34
		12.3.4 Cluster migration	34

12.4	Capsule management function	34
12.4.1	Cluster template instantiation.....	35
12.4.2	Capsule deletion.....	35
13	Coordination functions.....	35
13.1	Event notification function.....	35
13.1.1	Concepts.....	35
13.1.2	Rules	35
13.2	Checkpoint and recovery function	36
13.2.1	Checkpointing	36
13.2.2	Recovery	36
13.3	Deactivation and reactivation function	37
13.3.1	Deactivation	37
13.3.2	Reactivation	37
13.4	Group function.....	37
13.4.1	Concepts.....	37
13.4.2	Rules	37
13.5	Replication function	37
13.6	Migration function	38
13.6.1	Replication	38
13.6.2	Deactivation and reactivation.....	38
13.7	Transaction function	38
13.7.1	Concepts.....	38
13.7.2	Rules	38
13.8	ACID transaction function.....	39
13.9	Engineering interface reference tracking function.....	39
14	Repository functions	40
14.1	Storage function	40
14.1.1	Concepts.....	40
14.1.2	Rules	40
14.2	Information organization function	40
14.3	Relocation function	40
14.3.1	Concepts.....	41
14.3.2	Rules	41
14.4	Type repository function.....	41
14.4.1	Rules	41
14.5	Trading function.....	41
14.5.1	Concepts.....	42
14.5.2	Rules	42
15	Security functions.....	42
15.1	Concepts.....	42
15.2	Access control function.....	42
15.3	Security audit function	43
15.4	Authentication function	43
15.5	Integrity function	43
15.6	Confidentiality function	44
15.7	Non-repudiation function.....	44
15.8	Key management function	45
16	ODP distribution transparency	45
16.1	Access transparency.....	46
16.2	Failure transparency.....	46
16.2.1	Concepts.....	46
16.2.2	Rules	46
16.2.2.1	Replication	46
16.2.2.2	Checkpoint and recovery	46
16.3	Location transparency	47

16.4	Migration transparency	47
16.4.1	Concepts.....	47
16.4.2	Rules	47
16.5	Persistence transparency	47
16.5.1	Concepts.....	47
16.5.2	Rules	47
16.6	Relocation transparency	48
16.7	Replication transparency	48
16.7.1	Concepts.....	48
16.7.2	Rules	48
16.8	Transaction transparency	48
16.8.1	Concept	48
16.8.2	Rules	48
Annex A –	Formal computational supertype/subtype rules	49
A.1	Notations and conventions.....	49
A.2	Type system	49
A.2.1	Typing rules	50
A.2.2	Type definitions	51
A.2.3	An algorithm for type checking	51
A.3	Signal interface signature types	52
A.4	Operation interface signature types.....	53
A.5	Stream interface types.....	53
A.6	Example	54
Index of	definitions	55

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 10746-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 21, *Open Systems Interconnection, data management and open distributed processing*, in collaboration with ITU-T. The identical text is published as ITU-T Recommendation X.903.

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 forms an integral part of this part of ISO/IEC 10746.

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 Recs. X.901 to X.904 | ISO/IEC 10746, 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 Reference Model 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 this Recommendation | International Standard. This part is not normative.
- ITU-T Rec. X.902 | ISO/IEC 10746-2: **Foundations**: Contains the definition of the concepts and analytical framework for normalised description of (arbitrary) distributed processing systems. It introduces the principles of conformance to ODP standards and the way in which they are applied. This is only to a level of detail sufficient to support this Recommendation | International Standard and to establish requirements for new specification techniques. This part is 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 Recommendation X.902 | ISO/IEC 10746-2. This part is normative.
- ITU-T Rec. X.904 | ISO/IEC 10746-4: **Architectural semantics**: Contains a formalization of the ODP modelling concepts defined in ITU-T Recommendation X.902 | ISO/IEC 10746-2, clauses 8 and 9. The formalization is achieved by interpreting each concept in terms of the constructs of the different standardized formal description techniques. This part is normative.

This Recommendation | International Standard contains one annex (this annex forms an integral part of the Reference Model).

This page intentionally left blank

INTERNATIONAL STANDARD

ITU-T RECOMMENDATION

INFORMATION TECHNOLOGY – OPEN DISTRIBUTED PROCESSING – REFERENCE MODEL: ARCHITECTURE

1 Scope

This ITU-T Recommendation | International Standard:

- defines how ODP systems are specified, making use of concepts in ITU-T Recommendation X.902 | ISO/IEC 10746-2;
- identifies the characteristics that qualify systems as ODP systems.

It establishes a framework for coordinating the development of existing and future standards for ODP systems and is provided for reference by those standards.

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 | Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunications 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.810 (1995) | ISO/IEC 10181-1:1996, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Overview*.
- ITU-T Recommendation X.811 (1995) | ISO/IEC 10181-2:1996, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Authentication framework*.
- ITU-T Recommendation X.812 (1995) | ISO/IEC 10181-3:1996, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Access control framework*.
- ITU-T Recommendation X.813¹⁾ | ISO/IEC 10181-4...¹⁾, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Non-repudiation framework*.
- ITU-T Recommendation X.814 (1995) | ISO/IEC 10181-5:1996, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Confidentiality framework*.
- ITU-T Recommendation X.815 (1995) | ISO/IEC 10181-6:1996, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Integrity framework*.
- ITU-T Recommendation X.816 (1995) | ISO/IEC 10181-7:1996, *Information technology – Open Systems Interconnection – Security frameworks for open systems: Security audit framework*.
- ITU-T Recommendation X.902 (1995) | ISO/IEC 10746-2:1996, *Information technology – Open distributed processing – Reference Model: Foundations*.

¹⁾ Presently at the stage of draft.

2.2 Paired Recommendations | International Standards equivalent in technical content

- CCITT Recommendation X.800 (1991), *Security architecture for Open Systems Interconnection for CCITT Applications*.
- ISO 7498-2:1989, *Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture*.

3 Definitions

For the purposes of this Recommendation | International Standard, the following definitions apply.

3.1 Descriptive definitions

This Reference Model makes use of the following term defined in ITU-T Rec. X.200 | ISO/IEC 7498-1:

- transfer syntax.

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.811 | ISO/IEC 10181-2:

- claimant;
- exchange authentication information;
- principal;
- trusted third party.

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.812 | ISO/IEC 10181-3:

- access control information;
- access decision function;
- access enforcement function;
- initiator;
- target.

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec X.813 | ISO/IEC 10181-4:

- evidence generator;
- evidence user;
- evidence verifier;
- (non-repudiable data) originator;
- (non-repudiable data) recipient;
- non-repudiation evidence;
- non-repudiation service requester;
- notary.

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.814 | ISO/IEC 10181-5:

- confidentiality-protected data;
- hide;
- originator;
- recipient;
- reveal.

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.815 | ISO/IEC 10181-6:

- integrity-protected data;
- originator;
- recipient;
- shield;
- validate.

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.816 | ISO/IEC 10181-7:

- alarms collector function;
- alarm examiner function;
- audit trail examiner function;
- audit trail archiver function;
- audit recorder function;
- audit trail examiner function;
- audit trail collector function.

This Recommendation | International Standard makes use of the following terms defined in ISO/IEC 11170-1 Key Management Framework:

- key generation;
- key registration;
- key certification;
- key deregistration;
- key distribution;
- key storage;
- key archiving;
- key deletion.

This Reference Model makes use of the terms defined in ITU-T Rec. X.902 | ISO/IEC 10746-2 shown in Figure 1.

3.2 Abbreviations

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

- ODP Open Distributed Processing.
- OSI Open Systems Interconnection.

4 Framework

This Reference Model defines a framework comprising:

- five *viewpoints*, called enterprise, information, computational, engineering and technology which provide a basis for the specification of ODP systems;
- a *viewpoint language* for each viewpoint, defining concepts and rules for specifying ODP systems from the corresponding viewpoint;
- specifications of the *functions* required to support ODP systems;
- *transparency prescriptions* showing how to use the ODP functions to achieve distribution transparency.

The architecture for ODP systems and the composition of functions is determined by the combination of the computational language, the engineering language and the transparency prescriptions.

abstraction;	failure;	perceptual reference point;
action;	fault;	permission;
activity;	<x> group;	persistence;
architecture;	identifier;	policy;
atomicity;	information;	producer object;
behaviour;	initiating object;	programmatic reference point;
binding;	instance;	prohibition;
class;	instantiation;	Quality of Service;
client object;	interaction;	reference point;
communication;	interchange reference point;	refinement;
communications management;	interface;	role;
composition;	interface signature;	server object;
configuration;	interworking reference point;	spawn action;
conformance point;	introduction;	stability;
consumer object;	invariant;	state;
contract;	liaison;	subdomain;
creation;	location in space;	subtype;
data;	location in time;	supertype;
decomposition;	name;	system;
deletion;	naming context;	<x> template;
distributed processing;	naming domain;	term;
distribution transparency;	notification;	thread;
<x> domain;	object;	trading;
entity;	obligation;	type;
environment;	ODP standards;	viewpoint.
error;	ODP system;	
establishing behaviour;	open distributed processing;	

Figure 1 – Terms taken from ITU-T Rec. X.902 | ISO/IEC 10746-2

4.1 Viewpoints

4.1.1 Concepts

4.1.1.1 Enterprise viewpoint: A viewpoint on an ODP system and its environment that focuses on the purpose, scope and policies for that system.

4.1.1.2 Information viewpoint: A viewpoint on an ODP system and its environment that focuses on the semantics of information and information processing.

4.1.1.3 Computational viewpoint: A viewpoint on an ODP system and its environment which enables distribution through functional decomposition of the system into objects which interact at interfaces.

4.1.1.4 Engineering viewpoint: A viewpoint on an ODP system and its environment that focuses on the mechanisms and functions required to support distributed interaction between objects in the system.

4.1.1.5 Technology viewpoint: A viewpoint on an ODP system and its environment that focuses on the choice of technology in that system.

4.1.2 Using viewpoints

The enterprise, information, computational, engineering and technology viewpoints have been chosen as a necessary and sufficient set to meet the needs of ODP standards. Viewpoints can be applied, at an appropriate level of abstraction, to a complete ODP system, in which case the environment defines the context in which the ODP system operates. Viewpoints can also be applied to individual components of an ODP system, in which case the component's environment will include some abstraction of both the system's environment and other system components.

NOTE – The process of abstraction might be such that the system's environment and the other system components are composed into a single object.

4.2 ODP viewpoint languages

4.2.1 Concept

4.2.1.1 <Viewpoint> language: Definitions of concepts and rules for the specification of an ODP system from the <viewpoint> viewpoint; thus: **engineering language:** definitions of concepts and rules for the specification of an ODP system from the engineering viewpoint.

4.2.2 Using viewpoint languages

This Reference Model defines a set of five languages, each corresponding to one of the viewpoints defined in 4.1.1. Each language is used for the specification of an ODP system from the corresponding viewpoint. These languages are:

- the enterprise language (defined in clause 5);
- the information language (defined in clause 6);
- the computational language (defined in clause 7);
- the engineering language (defined in clause 8);
- the technology language (defined in clause 9).

Each language uses concepts taken from ITU-T Rec. X.902 | ISO/IEC 10746-2, and introduces refinements of those concepts, prescriptive rules and additional viewpoint-specific concepts relevant to the nature of the specifications concerned. These additional concepts are, in turn, defined using concepts from ITU-T Rec. X.902 | ISO/IEC 10746-2.

A system specification comprises one or more viewpoint specifications. These specifications must be mutually consistent. Rules for the consistent structuring of viewpoint specifications are given in clause 10. The specifier must demonstrate by other means that terms in the specifications are used consistently. A specification of a system using several viewpoint specifications will often restrict implementations more than a specification using fewer viewpoint specifications. Objects identified in one viewpoint can be specified using the viewpoint language associated with that viewpoint or using the viewpoint languages associated with other viewpoints. It is not necessary to specify an object fully from every viewpoint in order to achieve a mutually consistent set of viewpoint specifications.

NOTES

- 1 The list of terms taken from ITU-T Rec X.902 | ISO/IEC 10746-2 are listed in Figure 1.
- 2 The qualification of a term from ITU-T Rec X.902 | ISO/IEC 10746-2 by the name of a viewpoint (e.g. as in “**computational** object”) is interpreted as using of the term from ITU-T Rec X.902 | ISO/IEC 10746-2, subject to whatever additional provisions are specified in the identified viewpoint language.
- 3 The unqualified use of a term from ITU-T Rec X.902 | ISO/IEC 10746-2 in a viewpoint specification (e.g. “interface”) is interpreted as if the term had been qualified by the name of the viewpoint (i.e. “computational interface”), if the associated viewpoint language places additional constraints on the term.

4.3 ODP functions

4.3.1 ODP function: A function required to support Open Distributed Processing.

4.3.2 Using ODP functions

This Reference Model specifies, in clauses 11 to 15, the functions required to achieve Open Distributed Processing.

Each ODP function description contains:

- an explanation of the use of the function for open distributed processing;
- prescriptive statements, about the structure and behaviour of the function, sufficient to ensure the overall integrity of the Reference Model;
- a statement of other ODP functions upon which it depends.

4.4 ODP distribution transparencies

4.4.1 Concepts

4.4.1.1 Access transparency: A distribution transparency which masks differences in data representation and invocation mechanisms to enable interworking between objects.

4.4.1.2 Failure transparency: A distribution transparency which masks, from an object, the failure and possible recovery of other objects (or itself), to enable fault tolerance.

4.4.1.3 Location transparency: A distribution transparency which masks the use of information about location in space when identifying and binding to interfaces.

4.4.1.4 Migration transparency: A distribution 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.

4.4.1.5 Relocation transparency: A distribution transparency which masks relocation of an interface from other interfaces bound to it.

4.4.1.6 Replication transparency: A distribution 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.

4.4.1.7 Persistence transparency: A distribution 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 a system is unable to provide it with processing, storage and communication functions continuously.

4.4.1.8 Transaction transparency: A distribution transparency which masks coordination of activities amongst a configuration of objects to achieve consistency.

4.4.2 Using distribution transparency

Distribution transparency is an important end-user requirement in distributed systems. This Reference Model defines a set of distribution transparencies which make it possible to implement ODP systems which are distribution transparent from the point of view of users of those systems. Distribution transparency is selective; the Reference Model includes rules for selecting and combining distribution transparencies in ODP systems.

This Reference Model contains, for each distribution transparency defined in 4.4.1.1 to 4.4.1.8, definitions of both:

- a schema for expressing requirements for the particular transparency;
- a refinement process for transforming a specification which contains requirements for the particular distribution transparency to a specification which explicitly realizes the masking implied by that transparency.

NOTES

1 In some cases (e.g. access transparency) the schema is null; in others (e.g. transaction transparency) the schema contains one or more parameters dictating the precise form of transparency required.

2 The refinement process typically involves introducing additional behaviour, including the use of one or more ODP functions into the specification.

The specifications of the refinement processes in clause 16 are prescriptive to the level required to ensure overall integrity of the Reference Model.

4.5 Standards derived from the framework

This Reference Model provides a framework for the definition of new standards and the use of existing standards as ODP standards.

ODP standards are any of:

- standards for components of ODP systems;
- standards for composing ODP system components;
- standards for modelling and specifying ODP systems.

ODP standards:

- use the enterprise language to specify policies;
- use the information language to specify consistent use and interpretation of information in, and between, standards;
- use the computational language to specify the configuration and behaviour of interfaces;
- use the engineering language to specify the infrastructures they require;
- use the technology language to specify conformance to international, private, or consensual specifications.

Standards for methodology, modelling, programming, implementation and testing of ODP systems use the framework as a whole.

ODP standards can be based on a subset of this Reference Model (e.g. by excluding some forms of interaction, particular functions or transparencies). Such standards can also extend upon this Reference Model, provided that the extensions they introduce do not change or contradict its provisions. Extensions will relate new terms to terms defined in this Reference Model: for example, by introducing new types and new type rules.

ODP standards comply with all the prescriptive statements in this Reference Model.

4.6 Conformance

The enterprise, information, computational and engineering languages are used to specify the conformance requirements for ODP systems. The technology language can be used to assert conformance to ODP standards in ODP systems. Each interface which is defined as a conformance point has an information specification to enable interpretation of interactions of that interface. The rules for identifying conformance points are given in the computational and engineering languages.

An ODP system conforms to an ODP standard if it satisfies the conformance requirements of that standard.

5 Enterprise language

The enterprise language comprises concepts, rules and structures for the specification of an ODP system from the enterprise viewpoint.

An enterprise specification defines the purpose, scope and policies of an ODP system.

In this Reference Model, prescription in the enterprise viewpoint is restricted to a small basic set of concepts and rules addressing the scope and nature of enterprise specifications.

5.1 Concepts

The enterprise language contains the concepts of ITU-T Rec. X.902 | ISO/IEC 10746-2 and those defined here, subject to the rules of 5.2.

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

5.1.2 <X> federation: A community of <x> domains.

5.2 Structuring rules

An enterprise specification defines, and the enterprise language is able to express, the purpose, scope and policies of an ODP system in terms of each of the following items:

- roles played by the system;
- activities undertaken by the system;
- policy statements about the system, including those relating to environment contracts.

In an enterprise specification, an ODP system and the environment in which it operates are represented as a community. At some level of description the ODP system is represented as an enterprise object in the community. The objectives and scope of the ODP system are defined in terms of the roles it fulfils within the community of which it is part, and policy statements about those roles. A community is defined in terms of each of the following elements:

- the enterprise objects comprising the community;
- the roles fulfilled by each of those objects;
- policies governing interactions between enterprise objects fulfilling roles;
- policies governing the creation, usage and deletion of resources by enterprise objects fulfilling roles;
- policies governing the configuration of enterprise objects and assignment of roles to enterprise objects;
- policies relating to environment contracts governing the system.

A role is defined in terms of the permissions, obligations, prohibitions and behaviour of the enterprise object fulfilling the role. An enterprise object can fulfil one or more roles in a community, and the roles which it can fulfil are determined by the contract on which the community is based. While it is part of one community the enterprise object can

continue to fulfil roles in other communities, subject to the provisions in the contracts of the communities involved. The enterprise object can fulfil different roles in different communities. Interactions between enterprise objects fulfilling appropriate roles within different communities can be considered as interactions between those communities.

NOTE 1 – Examples of roles include policy administrator, president, service provider, owner, manager, shareholder, consumer.

NOTE 2 – Examples of environment contracts in enterprise specifications include safety requirements, legislative requirements and codes of practice.

NOTE 3 – In an enterprise specification the term “<x> object”, where <x> is a role, is interpreted as meaning “an enterprise object fulfilling an <x> role”; where an enterprise object fulfils multiple roles, the names can be concatenated, e.g. “owner driver object”.

When fulfilling a role, an object becomes subject to permissions, obligations and prohibitions by delegation or transfer. In some roles, objects are permitted to change policy. There are five fundamental types of actions with respect to contractual matters:

- an object incurs an obligation to another object (it must currently be permitted to incur the obligation);
- an object fulfils an obligation to another object;
- an object waives an obligation to another object;
- an object acquires permission from another object to perform some action it was previously forbidden to perform;
- an object is forbidden to perform an action it was previously permitted to perform.

NOTE 4 – An important special case of acquisition is where the permitted action is performative, i.e. when an object in a subordinate role is enabled to issue further permissions or obligations on behalf of an object fulfilling a superior role. This leads to the notion of agency or delegation.

Obligations include accounting and charging for the use of resources. Billing and payment is modelled as the reassignment of resources between objects in accordance with the roles they fulfil.

A resource is either consumable or non-consumable. A consumable resource deletes itself after some amount of use. In <x> federation, the objective defines the resources each <x> domain in the federation shares with the other members of the federation. The objective can leave each domain with a defined degree of autonomy in the use of its own resources. The establishing behaviour for an <x> federation can allow autonomy for each participating <x> domain in deciding whether or not to become part of the federation.

5.3 Conformance and reference points

Conformance statements in the enterprise language require that the behaviour of an ODP system is conformant to a particular set of objectives and policies.

An implementor claiming conformance must identify the engineering reference points which give access to the system and the engineering, computational and information specifications which apply at them. By this act the identified reference points become conformance points. The interactions at these conformance points can then be interpreted in enterprise language terms to check that the enterprise specification is not violated.

Enterprise specifications can be applied to all four classes of reference point (programmatic, perceptual, interworking and interchange reference points) identified in ITU-T Rec. X.902 | ISO/IEC 10746-2.

6 Information language

The information language comprises concepts, rules and structures for the specification of an ODP system from the information viewpoint.

An information specification defines the semantics of information and the semantics of information processing in an ODP system.

In this Reference Model, prescription in the information viewpoint is restricted to a small basic set of concepts and rules addressing the scope and nature of information specifications.

6.1 Concepts

The information language contains the concepts of ITU-T Rec. X.902 | ISO/IEC 10746-2 and those defined here, subject to the rules of 6.2.

6.1.1 Invariant schema: A set of predicates on one or more information objects which must always be true. The predicates constrain the possible states and state changes of the objects to which they apply.

NOTE – Thus, an invariant schema is the specification of the types of one or more information objects that will always be satisfied by whatever behaviour the objects might exhibit.

6.1.2 Static schema: A specification of the state of one or more information objects, at some point in time, subject to the constraints of any invariant schemata.

NOTE – Thus, a static schema is the specification of the types of one or more information objects at some particular point in time. These types are subtypes of the types specified in the invariant schema.

6.1.3 Dynamic schema: A specification of the allowable state changes of one or more information objects, subject to the constraints of any invariant schemata.

NOTES

1 Behaviour in an information system can be modelled as transitions from one static schema to another, i.e. reclassification of instances from one type to another.

2 In the information language, a state change involving a set of objects can be regarded as an interaction between those objects. Not all of the objects involved in the interaction need change state; some of the objects may be involved in a read-only manner.

6.2 Structuring rules

An information specification defines the semantics of information and the semantics of information processing in an ODP system in terms of a configuration of information objects, the behaviour of those objects and environment contracts for the system.

An information object template references static, invariant and dynamic schemata. The relationships between information objects can be modelled as part of the state of those information objects. Information objects are either atomic or are represented as a composition of component information objects. The state of the composite object is represented by the combined state of its component information objects. An atomic information object template represents a concept for which there is no model at a particular level of abstraction. A composite information object represents a derived concept expressed in terms of other concepts. Since object composition includes encapsulation, an information object that is a component of one composite object cannot be a component of another. Therefore, information objects resulting from the instantiation of a composite information object template only exist as part of the instantiated composite object and have no meaning outside it.

Allowable state changes specified by a dynamic schema can include the creation of new information objects and the deletion of information objects involved in the dynamic schema. Allowable state changes can be subject to ordering and temporal constraints.

NOTE 1 – The result of accessing the state of one or more information objects can be modelled as the creation of a new information object.

In an information specification, the configuration of information objects and the behaviour of those objects need not be suitable for distribution (e.g. there need not be any concept of failure or location for information interactions).

NOTE 2 – If an information notation uses the concept of interface, any interfaces defined cannot themselves be reference points; thus there is no commitment to the interfaces appearing in an implementation.

6.3 Conformance and reference points

Conformance statements in information specifications require that the behaviour of an ODP system is conformant to a particular set of invariant, static and dynamic schemata.

An implementor claiming conformance must list the relevant engineering reference points which give access to the system and the engineering and computational specifications which apply to them. By this act the identified reference points become conformance points. The interactions at these conformance points can then be interpreted in information language terms to check that they are consistent with the invariant, static and dynamic schemata.

Information specifications can be applied to all four classes of reference point (programmatic, perceptual, interworking, and interchange reference points) identified in ITU-T Rec. X.902 | ISO/IEC 10746-2.

7 Computational language

The computational language comprises concepts, rules and structures for the specification of an ODP system from the computational viewpoint.

A computational specification defines the functional decomposition of an ODP system into objects which interact at interfaces.

In the computational viewpoint, applications and ODP functions consist of configurations of interacting computational objects.

7.1 Concepts

The computation language contains the concepts of ITU-T Rec. X.902 | ISO/IEC 10746-2 and those defined here, subject to the structuring rules of 7.2.

7.1.1 Signal: An atomic shared action resulting in one-way communication from an initiating object to a responding object.

NOTE – A signal is an interaction.

7.1.2 Operation: An interaction between a client object and a server object which is either an interrogation or an announcement.

7.1.3 Announcement: An interaction – the **invocation** – initiated by a client object resulting in the conveyance of information from that client object to a server object, requesting a function to be performed by that server object.

7.1.4 Interrogation: An interaction consisting of:

- one interaction – the **invocation** – initiated by a client object, resulting in the conveyance of information from that client object to a server object, requesting a function to be performed by the server object, followed by
- a second interaction – the **termination** – initiated by the server object, resulting in the conveyance of information from the server object to the client object in response to the invocation.

NOTE – In interrogations, invocations and terminations are always paired. Announcements do not have terminations. Thus there is no possibility of an operation consisting of an invocation followed by a sequence of associated terminations.

7.1.5 Flow: An abstraction of a sequence of interactions, resulting in conveyance of information from a producer object to a consumer object.

NOTE – A flow may be used to abstract over, for example, the exact structure of a sequence of interactions, or over a continuous interaction including the special case of an analogue information flow.

7.1.6 Signal interface: An interface in which all the interactions are signals.

7.1.7 Operation interface: An interface in which all the interactions are operations.

7.1.8 Stream interface: An interface in which all the interactions are flows.

7.1.9 Computational object template: An object template which comprises a set of computational interface templates which the object can instantiate, a behaviour specification and an environment contract specification.

7.1.10 Computational interface template: An interface template for either a signal interface, a stream interface or an operation interface. A computational interface template comprises a signal, a stream or an operation interface signature as appropriate, a behaviour specification and an environment contract specification.

7.1.11 Signal interface signature: An interface signature for a signal interface. A signal interface signature comprises a finite set of action templates, one for each signal type in the interface. Each action template comprises the name for the signal, the number, names and types of its parameters and an indication of causality (initiating or responding, but not both) with respect to the object which instantiates the template.

7.1.12 Operation interface signature: An interface signature for an operation interface. An operation interface signature comprises a set of announcement and interrogation signatures as appropriate, one for each operation type in the interface, together with an indication of causality (client or server, but not both) for the interface as a whole, with respect to the object which instantiates the template.

Each **announcement signature** is an action template containing both the name of the invocation and the number, names and types of its parameters.

Each **interrogation signature** comprises an action template with the following elements:

- the name of the invocation;
- the number, names and types of its parameters;
- a finite, non-empty set of action templates, one for each possible termination type of the invocation, each containing both the name of the termination and the number, names and types of its parameters.

7.1.13 Stream interface signature: An interface signature for a stream interface. A stream interface comprises a finite set of action templates, one for each flow type in the stream interface. Each action template for a flow contains the name of the flow, the information type of the flow, and an indication of causality for the flow (i.e. producer or consumer but not both) with respect to the object which instantiates the template.

NOTES

1 The phrase “complementary interface signature to *X*”, where *X* is itself an interface signature describes an interface signature identical to *X* in all respects except causality, which is opposite to that in *X*.

2 Many Interface Definition Languages (IDLs) capture only the action templates of a signature and depend upon the context in which the IDL is used to determine the causality that is to be applied.

7.1.14 Binding object: A computational object which supports a binding between a set of other computational objects.

NOTE – Binding objects are subject to special provisions (see 7.2.3).

7.2 Structuring rules

A computational specification describes the functional decomposition of an ODP system, in distribution transparent terms, as:

- a configuration of computational objects (including binding objects);
- the internal actions of those objects;
- the interactions that occur among those objects;
- environment contracts for those objects and their interfaces.

A computational specification is constrained by the rules of the computational language. These comprise:

- interaction rules (see 7.2.2), binding rules (see 7.2.3) and type rules (see 7.2.4) that provide distribution transparent interworking;
- template rules (see 7.2.5) that apply to all computational objects;
- failure rules (see 7.2.6) that apply to all computational objects and identify the potential points of failure in computational activities.

Portability rules (see 7.2.7) are provided to give guidance to developers of ODP Portability standards.

A computational specification defines an initial set of computational objects and their behaviour. The configuration will change as the computational objects:

- instantiate further computational objects;
- instantiate further computational interfaces;
- perform binding actions;
- effect control functions upon binding objects;
- delete computational interfaces;
- delete computational objects.

7.2.1 Naming rules

Each kind of name defined in the computational language has an associated context, as follows:

- a signal name in a signal interface signature is an identifier in the context of that signature;
- a flow name in a stream interface signature is an identifier in the context of that signature;
- an invocation name in an operation interface signature is an identifier in the context of that signature;
- a termination name in an operation interface signature is an identifier in the context of the operation template in which it appears;
- the name of a parameter in a signal template is an identifier in the context of that template;

- the name of a parameter in an invocation template in an operation interface signature is an identifier in the context of that template;
- the name of a parameter in a termination template in an operation interface signature is an identifier in the context of that template;
- the name of a parameter in a signal template in a signal interface signature is an identifier in the context of that template.

NOTE 1 – Thus signal names are distinct in any signal interface signature but signals in different signatures can have the same name, and so forth.

A computational interface identifier is unambiguous within its context (i.e. cannot be associated with more than one computational interface in that context). The choice of contexts for computational interface identifiers is a matter of language design and therefore beyond the scope of this Reference Model. Hence the Reference Model places no constraints on the extent of contexts for computational interface identifiers. Thus no reliance can be placed on:

- the extent of naming contexts for computational interface identifiers (e.g. assumptions about them being related to engineering language structures such as nodes or communications domains);
- the uniqueness of computational interface identifiers (i.e. synonyms are permitted);
- a computational interface identifier denoting the same computational interface in all places where the identifier appears (i.e. names need not be “global”).

NOTE 2 – A particular computational notation may not have explicit terms denoting computational identifiers; therefore in that notation computational interface identifiers are implicit; however they remain subject to the rules given above.

7.2.2 Interaction rules

Each interaction of a computational object occurs at one of its computational interfaces. The computational language imposes constraints on the behaviour permitted at a computational interface. Interaction at an unbound interface fails. The binding rules (see 7.2.3) impose constraints on how interfaces can be bound.

The interaction part of the computational language supports three models of interaction, each of which has an associated kind of computational interface:

- signals and signal interfaces;
- flows and stream interfaces;
- operations and operation interfaces.

In addition to the different kinds of interaction supported, the interaction models differ in their failure properties. The participants in a flow or operation can have an inconsistent view of an interaction at different times, especially when failures have occurred. In contrast to streams and operations, there is no concept of partial failure of a signal – a signal either succeeds or fails identically for both participants in the interaction.

7.2.2.1 Signal interaction rules

A computational object offering a signal interface of a given signal interface type:

- initiates signals that have initiating causality in the interface’s signature;
- responds to signals that have responding causality in the interfaces’s signature.

7.2.2.2 Stream interaction rules

A computational object offering a stream interface:

- generates flows that have producer causality in the interface’s signature;
- receives flows that have consumer causality in the interface’s signature.

7.2.2.3 Operation interaction rules

A client object using an operation interface invokes the operations named in the interface’s signature. A server object offering an operation interface expects any of the operations named in the interface’s signature. In the case of an interrogation, the server responds to the invocation by initiating any one of the terminations named for the operation in the server interface signature. The client expects any of the terminations named for the operation in the client interface signature. The duration of the operation is arbitrary unless required otherwise by environment contracts applicable to the objects and interfaces involved.

NOTE – If a client thread invokes a chain of interrogations, the two way “handshake” of invocation and termination ensures that operations will be responded to by the server in the same order as the client initiated them. If the client invokes a chain of announcements (or a chain containing both announcements and invocations) there is no handshake to guarantee the order in which the server responds to the announcement, unless this is implicit in the environment contracts applicable to the interaction. There are no ordering guarantees for either interrogations or announcements that are in different descendant activities of a previous *invoking action*.

7.2.2.4 Parameter rules

The parameters for signals, invocations and terminations can include identifiers for computational interfaces and computational interface signature types.

NOTE 1 – The possibility of computational interface signature parameters makes the computational signature type systems higher order. Explicit representation of signature types is required, for example in trading, where the parameters of import and export operations include signature types:

```
trader.import (T: Type, ....) : (service: T) -> failed (reason: String)
trader.export (T: Type, service: T) : (....) -> failed (reason: String)
```

This introduces a need for dynamic signature subtype checking (see 7.2.5.1).

A formal parameter that is an identifier for a computational interface is qualified by a computational interface signature type. The corresponding actual parameter must reference an interface with that interface signature type (or one of its subtypes). The actual parameter can only be used as if it referenced a computational interface with the same signature type as the formal parameter (or one of the formal parameter's supertypes). After an interaction, both initiator and responder can reference the identified interface, although possibly by different computational interface identifiers.

NOTE 2 – This rule prevents the user of the interface referenced by the actual parameter being able to perform additional interactions beyond those in the formal interface signature type, even if the interface referenced by the actual parameter is a subtype of the interface signature type associated with the formal parameter.

7.2.2.5 Flows, operations and signals

Flows and operations can be defined in terms of signals. This enables signal interfaces to be used as a basis for explaining multi-party binding; end-to-end Quality of Service characteristics and compound bindings between different kinds of interface (e.g. stream to operation interface bindings).

A definition of flows using signals depends upon the details of the interactions abstracted in the specification of the stream interface concerned and therefore is beyond the scope of this Reference Model.

Operations can be modelled by signals by introducing corresponding signal interfaces to both the client operation interface and server operation interface involved:

- in a signal interface corresponding to a client operation interface there is a signal (**invocation submit**) corresponding to each invocation with the same parameters, and, in the case of an interface containing interrogations, a signal (**termination deliver**) corresponding to each possible termination with the same parameters as that termination;
- in the signal interface corresponding to a server operation interface there is a signal (**invocation deliver**) corresponding to each invocation with the same parameters, and, in the case of an interface containing interrogations, a signal (**termination submit**) corresponding to each possible termination with the same parameters as that termination.

This creates an equivalence between the resulting set of signals and the set of invocations and terminations in the operation interfaces being described.

7.2.3 Binding rules

In this Reference Model binding is defined with reference to binding actions. Use of such actions is called **explicit binding**. There are two kinds of binding actions: **primitive binding actions** and **compound binding actions**.

A primitive binding action binds two computational object directly. A compound binding action can be expressed in terms of primitive binding actions linking two or more computational objects via a binding object. The presence of a binding object in a computational binding gives the means to express configuration and Quality of Service control (see 7.2.3.3).

In notations which have no terms for expressing binding actions, binding is implicit. **Implicit binding** for other than server operation interfaces is not defined in the Reference Model because in other cases it is not self-evident where the initiative in binding should be placed relative to subsequent interaction. The additional information needed can be supplied in an explicit binding action.

7.2.3.1 Implicit binding rules for server operation interfaces

If an invocation by a client object references a server operation interface to which the client is not bound, implicit binding is required. Setting up an implicit binding involves the following procedure if a suitable client operation interface bound to the server does not exist:

- create a client operation interface of complementary signature type to the server interface;
- bind the client operation interface to the server operation interface;