

Information technology — Open Distributed Processing — Reference model: Overview

ICS 35.100.01

NO COPYING WITHOUT BSI PERMISSION EXCEPT AS PERMITTED BY COPYRIGHT LAW



Document Sharing Hub

National foreword

This British Standard reproduces verbatim ISO/IEC 10746-1:1998 and implements it as the UK national standard.

The UK participation in its preparation was entrusted to Technical Committee IST/15, Software Engineering, which has the responsibility to:

- aid enquirers to understand the text;
- present to the responsible international/European committee any enquiries on the interpretation, or proposals for change, and keep the UK interests informed;
- monitor related international and European developments and promulgate them in the UK.

A list of organizations represented on this committee can be obtained on request to its secretary.

Cross-references

The British Standards which implement international publications referred to in this document may be found in the BSI Standards Catalogue under the section entitled “International Standards Correspondence Index”, or by using the “Find” facility of the BSI Standards Electronic Catalogue.

A British Standard does not purport to include all the necessary provisions of a contract. Users of British Standards are responsible for their correct application.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

Summary of pages

This document comprises a front cover, an inside front cover, pages i and ii, the ISO/IEC title page, pages ii to vi, pages 1 to 78 and a back cover.

This standard has been updated (see copyright date) and may have had amendments incorporated. This will be indicated in the amendment table on the inside front cover.

This British Standard, having been prepared under the direction of the DISC Board, was published under the authority of the Standards Committee and comes into effect on 15 October 1999

© BSI 04-2000

ISBN 0 580 35197 1

Amendments issued since publication

Amd. No.	Date	Comments

Contents

	Page
National foreword	Inside front cover
Foreword	vi
Text of ISO/IEC 10746-1	1

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*



Contents

	Page
Foreword	vi
Introduction	1
1 Scope and field of application	1
2 Normative references	2
2.1 Identical Recommendations International Standards	2
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	9
6.2.1 Object modelling	9
6.2.2 Viewpoint specifications	9
6.2.3 Distribution transparency	9
6.2.4 Conformance	10
6.3 Standards	10
6.3.1 The Reference Model	10
6.3.2 Specific standards	10
7 Foundations	11
7.1 Basic modelling concepts	11
7.1.1 Objects	11
7.1.2 Interfaces and interaction points	12
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	14
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	15
7.3.3 Contract	15
7.3.4 Liaison and binding	15
8 Architecture	16
8.1 Architectural framework	16
8.1.1 Viewpoints	16
8.1.2 Distribution transparencies	17
8.2 Enterprise language	18
8.3 Information language	20
8.4 Computational language	21
8.4.1 Computational interfaces	22
8.4.2 Binding model	22
8.4.3 Typing and subtyping for computational interfaces	24

	Page
8.4.4 Portability	25
8.5 Engineering language	25
8.5.1 Clusters, capsules and nodes	25
8.5.2 Channels	26
8.5.3 Interface references	30
8.5.4 Binding	31
8.5.5 Channel establishment	31
8.5.6 Management interfaces	32
8.5.7 Interceptors	32
8.5.8 Conformance points	34
8.6 Technology language	34
8.7 Consistency between viewpoints	34
8.7.1 Enterprise viewpoint consistency with other viewpoints	36
8.7.2 Correspondences between computational and engineering specifications	37
8.8 ODP functions	39
8.8.1 Management functions	39
8.8.2 Coordination functions	40
8.8.3 Repository functions	41
8.8.4 Security functions	41
8.9 ODP distribution transparencies	42
8.9.1 Access transparency	42
8.9.2 Failure transparency	42
8.9.3 Location transparency	42
8.9.4 Migration transparency	42
8.9.5 Persistence transparency	42
8.9.6 Relocation transparency	43
8.9.7 Replication transparency	43
8.9.8 Transaction transparency	43
9 Conformance assessment	43
9.1 Conformance assessment and the development process	43
9.2 Conformance assessment: Relevant relationships	44
9.3 Conformance points and related concepts	44
9.4 ODP conformance specifications	45
9.4.1 Level of abstraction	45
9.4.2 Use of multiple reference points	45
9.5 Conformance implications of viewpoint languages	46
9.6 Conformance assessment activities	46
10 Management of ODP systems	46
10.1 Management domains	47
10.2 Management policy	47
10.3 Modelling management structures	47
11 The use of standards in ODP systems	48
11.1 Enterprise viewpoint	48
11.1.1 Enterprise specification	48
11.1.2 The application of standards	49
11.2 Information viewpoint	49
11.2.1 Information specification	49

	Page
11.2.2 The application of standards	50
11.3 Computational viewpoint	50
11.3.1 Computational specification	50
11.3.2 The application of standards	51
11.4 Engineering viewpoint	51
11.4.1 Engineering specification	51
11.4.2 The application of standards	52
11.5 Technology viewpoint	53
11.5.1 Technology specification	53
11.5.2 The application of standards	53
12 Examples of ODP specifications	54
12.1 Multimedia Conferencing System	54
12.1.1 Introduction	54
12.1.2 Enterprise specification	55
12.1.3 Information specification	56
12.1.4 Computational specification	57
12.1.5 Engineering specification	60
12.1.6 Technology specification	61
12.2 Multiparty audio/video stream binding	62
12.2.1 General description	62
12.2.2 Enterprise specification	63
12.2.3 Information specification	64
12.2.4 Computational specification	67
12.2.5 Engineering specification	69
12.2.6 Technology specification	70
12.3 A management example — Metric Object	71
12.3.1 Enterprise specification	71
12.3.2 Information specification	72
12.3.3 Computational specification	73
12.4 Database example	74
12.4.1 Enterprise specification	74
12.4.2 Information specification	74
12.4.3 Computational specification	74
Annex A Bibliography	78
Figure 1 — Object composition	13
Figure 2 — Compound binding	23
Figure 3 — Capsules, clusters and nodes	27
Figure 4 — An example of a client-server channel	28
Figure 5 — An example of a multiple channel configuration	29
Figure 6 — An example of a group channel configuration	30
Figure 7 — In-line Interceptor — Technology boundary	33
Figure 8 — Split Interceptor — Administrative boundary	33
Figure 9 — Split Interceptor — Combined technology and administrative boundary	33
Figure 10 — A V1 view of a system	35
Figure 11 — A V2 view of a system	35
Figure 12 — Correspondences between different viewpoints of a system	36

	Page
Figure 13 — Example of correspondence between computational and engineering viewpoints	37
Figure 14 — One-to-one correspondence	38
Figure 15 — Many-to-one correspondence	39
Figure 16 — Example system configuration	48
Figure 17 — Enterprise viewpoint	49
Figure 18 — Computational viewpoint	51
Figure 19 — Engineering viewpoint	52
Figure 20 — Technology viewpoint	53
Figure 21 — Configuration of a Multimedia Conferencing System	55
Figure 22 — ODP Enterprise specification using OMT graphical notation	56
Figure 23 — Information specification	58
Figure 24 — Computational configuration of MMCS	59
Figure 25 — A computational representation of objects and interfaces	59
Figure 26 — Representation of an explicit binding	59
Figure 27 — MMCS configuration of objects involved in audio/video interchange	60
Figure 28 — Engineering support	61
Figure 29 — Multipoint channel for stream 2	61
Figure 30 — Multiparty audio/video stream binding	63
Figure 31 — The binding contract between users, customers and provider	65
Figure 32 — Invariant schema of binding contract	65
Figure 33 — Information to computational mapping	67
Figure 34 — Objects involved in a multiparty audio/video exchange	68
Figure 35 — Computational to engineering mapping	69
Figure 36 — Engineering to technology mapping	71
Figure 37 — Order processing system — Enterprise specification	75
Figure 38 — Order processing system — Information specification	76
Figure 39 — Order processing system — Computational specification	77
Table 1 — IDL specification of Stream Binding Control Interface	68

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.

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, *Information technology — Open Systems Interconnection — Remote Procedure Call (RPC) — Part 2: Interface Definition Notation*¹⁾.
- 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:

¹⁾ To be published.

- 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;
- 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 *interactions* 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.

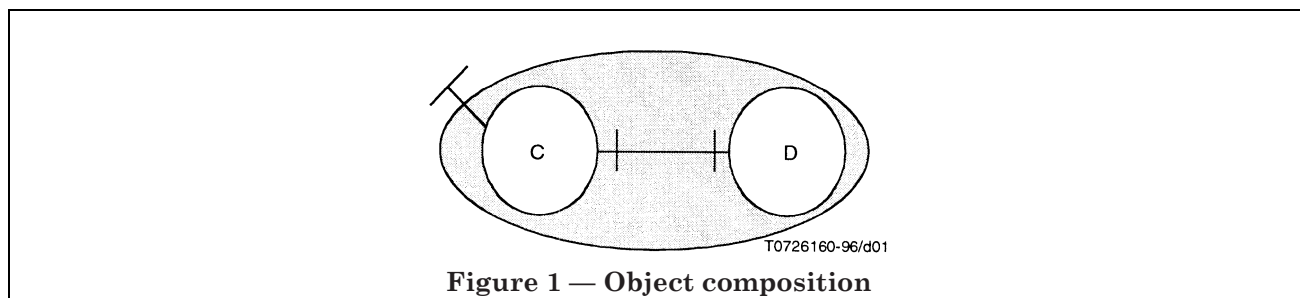
7.2 Specification concepts

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.



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.