ISO/IEC JTC1/SC7/WG19 HYD-011

ITU-T X.902 | ISO/IEC 10746-2

Date: 2010-04-20

ISO/IEC JTC 1/SC 7

Software Engineering

Secretariat: Canada (SCC)

Doc Type:	International Standard
Title:	ITU-T X.902 | ISO/IEC 10746-2 Information Technology – Open Distributed Processing – Reference Model – Foundations
Source:	Project editors
Project:	10746-2
Status:	Final Draft International Standard (FDIS)
Action:	For FDIS Ballot review and comments
Distribution:	SC 7 and ITU
Medium:	E
Number of Pages:	-
Version:	v01.04
Address reply to:	av@lcc.uma.es

ISO/IEC JTC1/SC7 Secretariat
École de technologie supérieure
1100, rue Notre-Dame Ouest
Montréal, Québec
Canada H3C 1K3

Tel. +1 514 396 8632
Fax. +1 514 396 8684

CONTENTS

Summary

This Recommendation | International Standard contains the definition of the concepts and analytical framework for normalized 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 Rec. X.903 | ISO/IEC 10746-3 and to establish requirements for new specification techniques.

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 of ODP 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 (RM-ODP), Recs. ITU-T 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 RM-ODP consists of:

- Rec. ITU-T 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 the RM-ODP 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 Rec. ITU-T X.903 | ISO/IEC 10746-3. This part is not normative.
- Rec. ITU-T X.902 | ISO/IEC 10746-2: **Foundations**: Contains the definition of the concepts and analytical framework for normalized 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 Rec. ITU-T X.903 | ISO/IEC 10746-3 and to establish requirements for new specification techniques. This part is normative.
- Rec. ITU-T 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 Rec. ITU-T X.902 | ISO/IEC 10746-2. This part is normative.
- Rec. ITU-T X.904 | ISO/IEC 10746-4: **Architectural semantics**: Contains a formalization of the ODP modelling concepts defined in this Recommendation | International Standard (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 does not contain any annexes.

INTERNATIONAL STANDARD

ITU-T RECOMMENDATION

INFORMATION TECHNOLOGY – OPEN DISTRIBUTED PROCESSING – REFERENCE MODEL: FOUNDATIONS

1 Scope

This ITU-T Recommendation | International Standard covers the concepts which are needed to perform the modelling of ODP systems (see clauses 6 to 14), and the principles of conformance to ODP systems (see clause 15).

The concepts defined in clauses 6 to 14 are used in the Reference Model of Open Distributed Processing to support the definitions of:

a) the structure of the family of standards which are subject to the Reference Model;

b) the structure of distributed systems which claim compliance with the Reference Model (the configuration of the systems);

c) the concepts needed to express the combined use of the various standards supported;

d) the basic concepts to be used in the specifications of the various components which make up the open distributed system.

Clause 15 defines how the various standards supported constrain an implementation and how such an implementation can be tested.

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 Recommendations and Standards 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

– Rec. ITU-TX.903 (1995) | ISO/IEC 10746-3:1996, *Information technology – Open distributed processing – Reference Model: Architecture.*

3 Definitions

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

3.1 Definitions from other Recommendations | International Standards

There are no definitions from other Recommendations | International Standards in this Recommendation | International Standard.

3.2 Background definitions

3.2.1 distributed processing: Information processing in which discrete components may be located in different places, and where communication between components may suffer delay or may fail.

3.2.2 ODP standards: This Reference Model and those standards that comply with it, directly or indirectly.

3.2.3 open distributed processing: Distributed processing designed to conform to ODP standards.

3.2.4 ODP system: A system (see 6.5) which conforms to the requirements of ODP standards.

3.2.5 information: Any kind of knowledge that is exchangeable amongst users, about things, facts, concepts and so on, in a universe of discourse.

Although information will necessarily have some forms of representation to make it communicable, it is the interpretation of this representation (the meaning) that is relevant in the first place.

3.2.6 data: The representations of information dealt with by information systems and users thereof.

3.2.7 viewpoint (on a system): A form of abstraction achieved using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within a system.

3.2.8 Viewpoint correspondence: A statement that some terms or other linguistic constructs in a specification from one viewpoint are associated with (e.g. describe the same entities as) terms or constructs in a specification from a second viewpoint. The forms of association that can be expressed will depend on the specification technique used.

NOTE – The terms associated by a correspondence need not necessarily be expressed using a single specification technique. The correspondence may associate a term in one specification technique with a term in some different specification technique. Rather than linking every individual pair of terms, general correspondences can also be expressed between specification techniques themselves. For example, composition operators defined in different specification techniques can be associated, implying correspondences wherever these operators are used to link terms in the respective viewpoints.

4 Abbreviations

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

ODP	Open Distributed Processing
OSI	Open Systems Interconnection
PICS	Protocol Implementation Conformance Statement
PIXIT	Protocol Implementation Extra Information for Testing
RM-ODP	Reference Model of Open Distributed Processing
TP	Transaction Processing

5 Categorization of concepts

The modelling concepts defined in this Recommendation | International Standard are categorized as follows:

a) *Basic interpretation concepts:* Concepts for the interpretation of the modelling constructs of any ODP modelling language. These concepts are described in clause 6.

b) *Basic linguistic concepts:* Concepts related to languages; the grammar of any language for the specification of the ODP architecture must be described in terms of these concepts. These concepts are described in clause 7.

c) *Basic modelling concepts:* Concepts for building the ODP architecture; the modelling constructs of any language must be based on these concepts. These concepts are described in clause 8.

d) *Specification concepts:* Concepts related to the requirements of the chosen specification languages used in ODP. These concepts are not intrinsic to distribution and distributed systems, but they are requirements to be considered in these specification languages. These concepts are described in clause 9.

e) *Structuring concepts:* Concepts that emerge from considering different issues in distribution and distributed systems. They may or may not be directly supported by specification languages adequate for dealing with the problem area. Specification of objects and functions that directly support these concepts must be made possible by the use of the chosen specification languages. These concepts are described in clauses 10 to 14.

f) *Conformance concepts:* Concepts necessary to explain the notions of conformance to ODP standards and of conformance testing. These concepts are defined in clause 15.

Rec. ITU-T X.903 | ISO/IEC 10746-3 uses the concepts in this Recommendation | International Standard to specify the characteristics for distributed processing to be open. It is organized as a set of viewpoint languages. Each viewpoint language refines concepts from the set defined in this Recommendation | International Standard. It is not necessary for all viewpoint languages to adopt the same notations. Different notations may be chosen as appropriate to reflect the

requirements of the viewpoint. These notations may be natural, formal, textual or graphical. However, it will be necessary to establish correspondences between the various languages to ensure overall consistency.

6 Basic interpretation concepts

Although much of the ODP Architecture is concerned with defining formal constructs, the semantics of the architectural model and any modelling languages used have to be described. These concepts are primarily meta-concepts, i.e. concepts which apply generally to any form of modelling activity. It is not intended that these concepts will be formally defined, or that they be used as the basis of formal definition of other concepts.

Any modelling activity identifies:

a) elements of the universe of discourse;

b) one or more pertinent levels of abstraction.

The elements of the universe of discourse are entities and propositions.

6.1 **Entity**: Any concrete or abstract thing of interest. While in general the word entity can be used to refer to anything, in the context of modelling it is reserved to refer to things in the universe of discourse being modelled.

6.2 **Proposition**: An observable fact or state of affairs involving one or more entities, of which it is possible to assert or deny that it holds for those entities.

6.3 **Abstraction**: The process of suppressing irrelevant detail to establish a simplified model, or the result of that process.

6.4 **Atomicity**: An entity is atomic at a given level of abstraction if it cannot be subdivided at that level of abstraction.

Fixing a given level of abstraction may involve identifying which elements are atomic.

6.5 **System**: Something of interest as a whole or as comprised of parts. Therefore a system may be referred to as an entity. A component of a system may itself be a system, in which case it may be called a subsystem.

NOTE – For modelling purposes, the concept of system is understood in its general, system-theoretic sense. The term “system” can refer to an information processing system but can also be applied more generally.

6.6 **Architecture (of a system)**: A set of rules to define the structure of a system and the interrelationships between its parts.

7 Basic linguistic concepts

Whatever the concepts or semantics of a modelling language for the ODP Architecture, the language will be expressed in some syntax, which may include linear text or graphical conventions. It is assumed that any suitable language will have a grammar defining the valid set of symbols and well-formed linguistic constructs of the language. The following concepts provide a common framework for relating the syntax of any language used for the ODP Architecture to the interpretation concepts.

7.1 **Term**: A linguistic construct which may be used to refer to an entity.

The reference may be to any kind of entity including a model of an entity or another linguistic construct.

7.2 **Sentence**: A linguistic construct containing one or more terms and predicates; a sentence may be used to express a proposition about the entities to which the terms refer.

A predicate in a sentence may be considered to refer to a relationship between the entities referred to by the terms it links.

7.3 **Model**: A system of postulates, value declarations and inference rules presented as a description of a state of affairs (Universe of Discourse).

NOTE – Construction of a model allows precise description and reasoning about the state of affairs.

7.4 **Specification**: A concrete representation of a model in some notation. Being in the real world, a specification can be inspected, manipulated or communicated.

NOTES

1 The specification may itself be an entity in the universe of discourse of the model it represents, but in simple cases it will generally only be modelled in a separate universe of discourse addressing the system development process.

2 The specification can be instantiated by one or more implementations, particularly, for example, in the specification of commodity software products. Each instantiation of the specification will, in general, represent a separate universe of discourse and so

lead to a separate set of entities with the relationships defined in the specification. Thus declaration of, for example, a singleton object (such as the ODP system) in a specification will lead to a separate ODP system instance each time the specification is implemented. This specification-instantiation distinction should be distinguished from the familiar type-instance distinctions between terms within the specification.

3 The relationship between a specification and its implementation underlies the conformance architecture defined in clause 15.

7.5 Notation: A means of concrete representation for a particular type of a model, expressed as a grammar and suitable glyphs for its terminal symbols.

NOTE – One notation may be capable of representing a number of types of models, or of representing a specific viewpoint on a more general model.

8 Basic modelling concepts

The detailed interpretation of the concepts defined in this clause will depend on the specification language concerned, but these general statements of concept are made in a language-independent way to allow the statements in different languages to be interrelated.

The basic concepts are concerned with existence and activity: the expression of what exists, where it is and what it does.

8.1 Object: A model of an entity. An object is characterized by its behaviour (see 8.7) and, dually, by its state (see 8.8). An object is distinct from any other object. An object is encapsulated, i.e. any change in its state can only occur as a result of an internal action or as a result of an interaction (see 8.3) with its environment (see 8.2).

An object interacts with its environment at its interaction points (see 8.12).

Depending on the viewpoint, the emphasis may be placed on behaviour or on state. When the emphasis is placed on behaviour, an object is informally said to perform functions and offer services (an object which makes a function available is said to offer a service, see 13.3.1). For modelling purposes, these functions and services are specified in terms of the behaviour of the object and of its interfaces (see 8.5). An object can perform more than one function. A function can be performed by the cooperation of several objects.

NOTES -- The expression “use of a function” is a shorthand for the interaction with an object which performs the function.

8.2 Environment (of an object): The part of the model which is not part of that object.

NOTE – In many specification languages, the environment can be considered to include at least one object which is able to participate without constraint in all possible interactions (see 8.3), representing the process of observation.

8.3 Action: Something which happens.

Every action of interest for modelling purposes is associated with at least one object.

The set of actions associated with an object is partitioned into **internal actions** and **interactions**. An internal action always takes place without the participation of the environment of the object. An interaction takes place with the participation of the environment of the object.

NOTES

1 “Action” means “action occurrence” not “action type”. That is to say, different actions within a specification may be of the same type but still distinguishable in a series of observations. Depending on context, a specification may express that an action has occurred, is occurring or may occur.

This usage of action occurrence needs to be seen in the light of the notes on specification in 7.4. Thus the specification of a firework may require it to produce five flashes and a bang, which are six actions where flash and bang are action types. However, each member of a box of fireworks conforming to this specification will produce its own copy of this behaviour.

2 The granularity of actions is a design choice. An action need not be instantaneous. Actions may overlap in time.

3 Interactions may be labelled in terms of cause and effect relationships between the participating objects. The concepts that support this are discussed in 13.3.

4 An object may interact with itself, in which case it is considered to play at least two roles in the interaction.

5 Involvement of the environment represents observability. Thus, interactions are observable whereas internal actions are not observable, because of object encapsulation. In most specification techniques observability is an implicit property of the environment and, therefore, it is not necessary to model the observer explicitly; however, there may, in some circumstances, be a need to include an explicit observer object in the specification, thereby increasing the cardinality of all interactions.

6 Observability of an action may depend on the level of specification. For instance, an action specification at one level of abstraction or in one viewpoint may correspond to a specification of multiple concurrent actions at a different level of abstraction or in another viewpoint. For example, a basic single function of a system in one viewpoint may be realized by multiple concurrent actions in a different viewpoint, defining a grid computing or sensor network, each one executing at the same time on network-connected computers in different locations. In this case, the observability of the occurrence of the basic single action can be deduced from the observability of those other multiple concurrent actions.

8.4 **Event**: The fact that an action has taken place. When an event occurs, the information about the action that has taken place becomes part of the state of the system and may thus subsequently be communicated in other interactions. Such a communication is called an event notification; it carries the information about the event from the object that performs or observes it to other objects that have a need to take action as a result of it.

NOTES

1 An action changes the state of the objects participating in it; an event is the fact that the action has occurred; an event notification is a communication about the event, caused by some action; the receipt of the notification changes the state of objects not participating in the original action.

2 An event notification may convey information about the fact that an internal action has occurred. For example, an internal action may change the availability of some server and a subsequent event notification may convey this fact to its potential clients.

8.5 **Interface**: An abstraction of the behaviour of an object that consists of a subset of the interactions of that object together with a set of constraints on when they may occur.

Each interaction of an object belongs to a unique interface. Thus, the interfaces of an object form a partition of the interactions of that object.

NOTES

1 An interface constitutes the part of an object behaviour that is obtained by considering only the interactions of that interface and by hiding all other interactions. Hiding interactions of other interfaces will generally introduce non-determinism as far as the interface being considered is concerned.

2 The phrase “an interface between objects” is used to refer to the binding (see 13.5.2) between interfaces of the objects concerned. In the two-party case, such bindings normally link interfaces with complementary causalities. For example, in a client-server binding (see 13.4.5 and 13.4.6), a client initiating interface is bound to a server providing interface. In many specification languages, the fact that the client has an initiating interface is not explicit, but is indicated by stating a requirement for the kind of server needed if the client is to operate successfully, i.e. the concept of a required interface.

3 An interface of an object may be used by other objects. Using interfaces provided by other objects may constitute a part of the object’s behaviour.

4 If an interface is provided by an object, part of the providing object’s behaviour is triggered when this interface is used by other objects. If an object uses an interface of some providing object, this is expressed by its behaviour involving an interaction which forms part of its own initiating interface. The interaction in the first object’s initiating interface is associated with the corresponding interaction in the other object’s providing interface as a result of the binding process between the two interfaces. An object may provide both initiating and providing interfaces

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

8.7 **Behaviour (of an object)**: A collection of actions with a set of constraints on when they may occur.

The specification language in use determines the constraints which may be expressed. Constraints may include for example sequentiality, non-determinism, concurrency or real-time constraints.

A behaviour may include internal actions.

The actions that actually take place are restricted by the environment in which the object is placed.

NOTES

1 The composition (see 9.1) of a collection of objects implicitly yields an equivalent object representing the composition. The behaviour of this object is often referred to simply as the behaviour of the collection of objects.

2 Action and activity are degenerate cases of behaviour.

3 In general, several sequences of interactions, called traces (see 9.7) are consistent with a given behaviour.

8.8 **State (of an object)**: At a given instant in time, the condition of an object that determines the set of all sequences of actions (or traces) in which the object can participate.

Since, in general, behaviour includes many possible series of actions in which the object might take part, knowledge of state does not necessarily allow the prediction of the sequence of actions which will actually occur.

State changes are effected by actions; hence a state is partially determined by the previous actions in which the object took part.

Since an object is encapsulated, its state cannot be changed directly from the environment, but only indirectly as a result of the interactions in which the object takes part.

8.9 **Communication**: The conveyance of information between two or more objects as a result of one or more interactions, possibly involving some intermediate objects.

NOTES

1 Communications may be labelled in terms of a cause and effect relationship between the participating objects. Concepts to support this are discussed in 13.3.

2 Every interaction is an instance of a communication.

3 Any communication can be seen as an interaction by abstracting away intermediate objects involved in the communication.

4 Any communication (and, hence, any interaction) can be provided by a wide range of technologies such as remote invocation, message transfer, etc.

8.10 **Location in space**: An interval of arbitrary size in space at which an action can occur.

8.11 **Location in time**: An interval of arbitrary size in time at which an action can occur.

NOTES

1 The extent of the interval in time or space is chosen to reflect the requirements of a particular modelling task and the properties of a particular specification technique. A single location in one specification may be subdivided in either time or space (or both) in another specification. In a particular specification, a location in space or time is defined relative to some suitable coordinate system.

2 By extension, the location of an object is the union of the locations of the actions in which the object may take part.

8.12 **Interaction point**: A location at which there exists a set of interfaces.

At any given location in time, an interaction point is associated with a location in space, within the specificity allowed by the specification language in use. Several interaction points may exist at the same location. An interaction point may be mobile.

8.13 **Relation**: An association between two or more domains of entities. In RM-ODP, relations can be defined for, at least, objects, interfaces and actions.

8.14 **Relationship**: An association between two or more entities.

NOTE – Relationships are instances of relations.

9 Specification concepts

9.1 Composition

a) (Of objects) – A combination of two or more objects yielding a new object, at a different level of abstraction. The characteristics of the new object are determined by the objects being combined and by the way they are combined. The behaviour of a composite object is the corresponding composition of the behaviour of the component objects.

b) (Of behaviours) – A combination of two or more behaviours yielding a new behaviour. The characteristics of the resulting behaviour are determined by the behaviours being combined and the way they are combined.

NOTES

1 Examples of combination techniques are sequential composition, concurrent composition, interleaving, choice, and hiding or concealment of actions. These general definitions will always be used in a particular sense, identifying a particular means of combination.

2 In some cases, the composition of behaviours may yield a degenerate behaviour, e.g. deadlock, due to the constraints on the original behaviours.

9.2 Composite object: An object expressed as a composition.

9.3 Decomposition

a) (Of an object) – The specification of a given object as a composition.

b) (Of a behaviour) – The specification of a given behaviour as a composition.

Composition and decomposition are dual terms and represent dual specification.

9.4 **Behavioural compatibility**: An object is behaviourally compatible with a second object with respect to a set of criteria (see Notes) if the first object can replace the second object without the environment being able to notice the difference in the objects' behaviour on the basis of the set of criteria.

Typically, the criteria impose constraints on the allowed behaviour of the environment. If the criteria are such that the environment behaves as a tester for the original object, i.e. the environment defines the smallest behaviour that does not constrain the behaviour of the original object, the resulting behavioural compatibility relation is called extension.

The criteria may allow the replacement object to be derived by modification of an otherwise incompatible object in order that it should be an acceptable replacement. An example of such a modification might be hiding of additional parameters on certain interactions. In this way, an interaction of the new object can be made to look like an interaction of the original object. In such cases behavioural compatibility is called **coerced behavioural compatibility**. If no modification is necessary, behavioural compatibility is called **natural behavioural compatibility**.

The concept of behavioural compatibility defined above on objects applies equally well to the behavioural compatibility of templates and of template types.

Behavioural compatibility is reflexive, but not necessarily symmetric or transitive (though it may be either or both).

NOTES

1 The set of criteria depends on the language in use and the testing theory applied.

2 Behavioural compatibility (with respect to a set of criteria) can be defined on template (see 9.13) and template types (see 9.22), thus:

a) if S and T are object templates, S is said to be behaviourally compatible with T if and only if any S-instantiation is behaviourally compatible with some T-instantiation (see 9.16);

b) if U and V are object template types, U and V are said to be behaviourally compatible if their corresponding templates are *behaviourally compatible*.

9.5 Interoperability: Capability of objects to collaborate, that is, the capability mutually to communicate information in order to exchange events, proposals, requests, results, commitments and flows.

NOTE – The term covers interoperability for different areas of concern (syntactic, semantic, pragmatic, etc.).

9.6 Refinement: The process of transforming one specification into a more detailed specification. The new specification can be referred to as a refinement of the original one. Specifications and their refinements typically do not coexist in the same system description. Precisely what is meant by a more detailed specification will depend on the chosen specification language.

For each meaning of behavioural compatibility determined by some set of criteria (see 9.4), a specification technique will permit the definition of a refinement relationship. If template X refines a template Y, it will be possible to replace an object instantiated from Y by one instantiated from X in the set of environments determined by the selected definition of behavioural compatibility. Refinement relationships are not necessarily either symmetric or transitive.

9.7 Trace: A record of an object's interactions, from its initial state to some other state.

A trace of an object is thus a finite sequence of interactions. The behaviour uniquely determines the set of all possible traces, but not vice versa. A trace contains no record of an object's internal actions.

9.8 <X> pattern: The abstract specification of a composition of objects that results in any instance of the composition having a given property, named by X.

NOTES

1 A pattern cannot, by itself, be instantiated (see <X> template, 9.13).

2 This definition is a generalization of the well-known concept of a design pattern. There <X> is the pattern name e.g. the factory design pattern.

9.9 Type (of an <X>): A predicate characterizing a collection of <X>s. An <X> is of the type, or satisfies the type, if the predicate holds for that <X>. A specification defines which of the terms it uses have types, i.e. are <X>s. In RM-ODP, types are needed for, at least, objects, interfaces and actions.

The notion of type classifies the entities into categories, some of which may be of interest to the specifier (see the concept of class in 9.10).

9.10 Class (of <X>s): The set of all <X>s satisfying a type (see 9.9). The elements of the set are referred to as members of the class.

NOTES

1 A class may have no members.

2 Whether the size of the set varies with time depends on the definition of the type.

9.11 Subtype/supertype: A type A is a subtype of a type B, and B is a supertype of A, if every <X> which satisfies A also satisfies B.

The subtype and supertype relations are reflexive, transitive and anti-symmetric.

9.12 Subclass/superclass: One class A is a subclass of another class B, and B is a superclass of A, precisely when the type associated with A is a subtype of the type associated with B.

NOTE – A subclass is by definition a subset of any of its superclasses.

9.13 <X> Template: The specification of the common features of a collection of <X>s in sufficient detail that an <X> can be instantiated using it. <X> can be anything that has a type (see 9.9).

An <X> template is an abstraction of a collection of <X>s.

A template may specify parameters to be bound at instantiation time.

The definition given here is generic; the precise form of a template will depend on the specification technique used. The parameter types (where applicable) will also depend on the specification technique used.

Templates may be combined according to some calculus. The precise form of template combination will depend on the specification language used.

9.14 Action signature: The specification of an action that comprises the name for the action, the number, names and types of its parameters, and an indication of the causality of the object that instantiates the action template.

NOTES

1 Action signatures focus just on the syntactic aspects of the specification of actions, while action templates cover all aspects. Hence, an action template normally comprises the specification of the action signature, together with further information such as a behavioural specification and an environment contract specification, for instance.

2 The inclusion, in an action signature, of information about the number of parameters is optional.

9.15 Interface signature: The set of action signatures associated with the interactions of an interface.

An object may have many interfaces with the same signature.

NOTE – Usually, interface signatures are part of the specification of interface templates. Thus, an interface template comprises the specification of the interface signature, a behaviour specification and an environment contract specification

9.16 Instantiation (of an <X> template): An <X> produced from a given <X> template and other necessary information. This <X> exhibits the features specified in the <X> template. <X> can be anything that has a type (see 9.9).

The definition given here is generic: how to instantiate an <X> template depends on the specification language used. Instantiating an <X> template may involve actualization of parameters, which may in turn involve instantiating other <X> templates or binding of existing interfaces (see 13.5).

NOTES

1 Instantiating an action template just results in an action occurring. The phrase “instantiation of an action template” is deprecated. “Occurrence of an action” is preferred.

2 If <X> is an object, it is instantiated in its initial state. An object can participate in interactions immediately after its instantiation.

3 Instantiations from different templates may satisfy the same type. Instantiations from the same template may satisfy different types.

9.17 Role: A formal placeholder in the specification of a composite object. It identifies those aspects of the behaviour of some component object required for it to form part of the composite and links them as constraints on an actual object in an instance of the composite. In order to satisfy the specification, the actual object is required to exhibit the specified behaviour. It is then said to fulfil the role in the instance of the composite.

Thus, the specification of the composite object is expressed as a composition of roles, which parameterize it. Instantiation binds specific component objects to each of the role parameters in the specification of the resultant composite object.

NOTES

1 The metaphor on which the role concept is based is theatrical. The text of a play is expressed in terms of lines and actions associated with various roles, which are declared initially in a cast-list. Putting the play on involves assigning actors to the various roles, although one actor may play several minor roles, and the actor playing a role may change during the run of the production. Identifying the roles rather than the actors obviously makes the script more reusable.

2 Any dynamic agreement governing shared behaviour of two or more objects implicitly defines a template for a composite object and the roles of those objects in that composite object. Thus, roles are defined in interactions (8.3), contracts (11.2.1), liaisons (13.2.4) and bindings (13.5.2), amongst others. When roles are defined in such contexts, the term role should be appropriately qualified (e.g. interaction-role, contract-role, etc.).

9.18 Creation (of an <X>): Instantiating an <X>, when it is achieved by an action of objects in the model. <X> can be anything that can be instantiated, in particular objects and interfaces.

If <X> is an interface, it is either created as part of the creation of a given object, or as an additional interface to the creating object. As a result, any given interface must be part of an object.

9.19 Introduction (of an <X>): Instantiating an <X> when it is not achieved by an action of objects in the model.

NOTES

1 An <X> can be instantiated either by creation or introduction but not both.

2 Introduction does not apply to interfaces and actions since these are always supported by objects.

9.20 Deletion (of an <X>): The action of destroying an instantiated <X>. <X> can be anything that can be instantiated, in particular objects and interfaces.

If <X> is an interface, it can only be deleted by the object to which it is associated.

NOTE – Deletion of an action is not meaningful: an action just happens.

9.21 Instance (of a type): An <X> that satisfies the type.

9.22 Template type (of an <X>): A predicate defined in a template that holds for all the instantiations of the template and that expresses the requirements the instantiations of the template are intended to fulfill.

The object template subtype/supertype relation does not necessarily coincide with behavioural compatibility. Instances of a template type need not be behaviourally compatible with instantiations of the associated template. They do coincide if:

a) a transitive behavioural compatibility relation is considered; and

b) template subtypes are behaviourally compatible with their template supertypes.

NOTES

1 This concept captures the notion of substitubility by design.

2 The form of the predicate that expresses the template type depends on the specification language used.

3 As a shorthand, "instances of a template T" are defined to be "instances of the template type associated with template T".

4 Figure 1 illustrates the relationships between some of the concepts: template type, template class, etc. The set of instances of **t** contains both the set of instantiations of **t** and the sets of all instantiations of subtypes of **t**. The sets of instantiations of different templates are always disjoint.

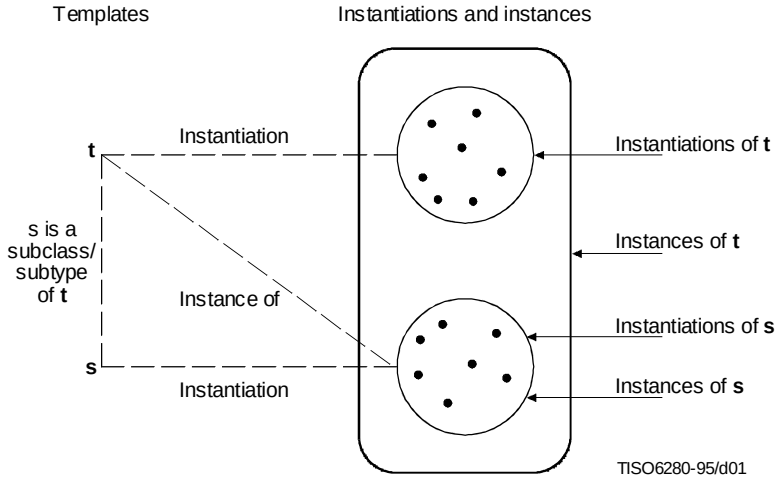

Figure 1 – Relationship between templates, instantiations and instances

9.23 Template class (of an <X>): The set of all <X>s satisfying an <X> template type, i.e. the set of <X>s which are instances of the <X> template. <X> can be anything that has a type (see 9.9).

Each template defines a single template class, so we may refer to instances of the template as instances of the template-class.

The notion of class is used to refer to a general classification of <X>s. Template class is a more restrictive notion where the members of a template class are limited to those instantiated from the template (or any of its subtypes), i.e. those <X>s which satisfy the <X> template type.

NOTE – Given a template type, we may shorten statements of the form "the template class associated with template A is a subclass of the template class associated with template B" to "template A is a subclass of template B" or "template A is a subtype of template B".

9.24 Derived class/base class: If a template A is an incremental modification of a template B, then the template class CA of instances of A is a derived class of the template class CB of instances of B, and the CB is a base class of CA.

The criteria for considering an arbitrary change to be an incremental modification would depend on metrics and conventions outside of this Recommendation | International Standard. If the criteria allow, a derived class may have several base classes.

The incremental modification relating templates must ensure that self-reference or recursion in the template of the base class becomes self-reference or recursion in the template of the derived class.

The incremental modification may, in general, involve adding to or altering the properties of the base template to obtain the derived template.

Classes can be arranged in an inheritance hierarchy according to derived class/base class relationships. This is the interpretation of inheritance in the ODP Reference Model. If classes can have several base classes, inheritance is said to be multiple. If the criteria prohibit suppression of properties from the base class, inheritance is said to be strict.

It is possible for one class to be a subclass of a second class without being a derived class, and to be a derived class without being a subclass. The inheritance hierarchy (where arcs denote the derived class relation) and the type hierarchy (where arcs denote the subtype or subclass relation) are therefore logically distinct, though they may coincide in whole or in part.

9.25 Factory (for an object): An object that, in response to an interaction initiated by its environment, creates a new object and returns a reference to it to the environment.

9.26 Component: An object that encapsulates its own template, so that the template can be interrogated by interaction with the component. The template and other instantiation parameters are expressed in a form that allows them to be updated during the lifetime of any system of which the component is to form a part, allowing alternative realizations of the component to be substituted.

9.27 Container (for a component): An object that can act as a factory and can provide the necessary environment for subsequent management of the components created by it. The container will, in response to an interaction initiated by its environment, provide information about the components it contains.

9.28 Invariant: A predicate that a specification requires to be true for the entire life time of a set of objects.

9.29 Precondition: A predicate that a specification requires to be true for an action to occur.

9.30 Postcondition: A predicate that a specification requires to be true immediately after the occurrence of an action.

10 Organizational concepts

10.1 <X> Group: A set of objects with a particular characterizing relationship <X>. The relationship <X> characterizes either the structural relationship among objects or an expected common behaviour of the objects.

NOTE – Examples of specialized groups are:

a) *Addressed group*: A set of objects that are addressed in the same way.

b) *Fault group*: A set of objects that have a common fault dependency. For example, it may be assumed that if a computer fails, all objects executing on that computer also fail.

c) *Communicating group*: A set of objects where all the objects participate in the same sequence of interactions with their environment.

d) *Fault tolerant replication group*: A communicating group whose purpose is to provide a certain level of tolerance against some faults.

10.2 Configuration (of objects): A collection of objects able to interact at interfaces. A configuration determines the set of objects involved in each interaction.

The specification of a configuration may be static or may be in terms of the operation of dynamic mechanisms which change the configuration, such as binding and unbinding (see 13.5).

NOTE – A configuration can be expressed in terms of the concepts of concurrent composition. The process of composition generates an equivalent object for the configuration, at a different level of abstraction.

10.3 <X> Domain: A set of objects, each of which is related by a characterizing relationship <X> to a controlling object.

Every domain has a controlling object associated with it.

The controlling object can determine the identities of the collection of objects which comprises the associated domain. The controlling object may communicate with a controlled object dynamically or it may be considered to have communicated in an earlier epoch (see 10.5) of the controlling object. Generally, the controlling object is not a member of the associated domain.

NOTES

1 In enterprise terms, various policies can be administered by the controlling object over the domain.

2 Domains can be disjoint or overlapping.

3 By definition, a domain is a group, but not vice versa.

4 Examples of specialized domains are:

Domain	Member Class	Relationship	Controlling Class
Security domain	processing object	subject to policy set by	security authority object
Management domain	managed object	subject to policy set by	management domain object
Addressing domain	addressed object	address allocated by	addressing authority object
Naming domain	named object	name allocated by	name authority object

10.4 **Subdomain**: A domain which is a subset of a given domain.

10.5 **Epoch**: A period of time for which an object displays a particular behaviour. Any one object is in a single epoch at one time, but interacting objects may be in different epochs at the time of interaction.

A change of epoch may be associated with a change in the type of the object, so as to support type evolution. Alternatively, a change of epoch may be associated with a phase in the behaviour of an object of constant type.

For a distributed system to function correctly, the objects composing its configuration must be consistent. Thus, as the whole system evolves through a series of epochs, the individual objects which interact must never be in epochs in which their behaviours are sufficiently different that their concurrent composition leads to a failure. This concept will support the formalization of concepts of version and extensibility.

NOTE – A specification language may need to express:

a) the way epochs are labelled;

b) the sequence of epochs, and whether all objects need to pass through all members of the sequence;

c) the rules for deriving the epoch of a composition from the epochs of its objects, particularly for configurations and complete systems;

d) whether identity of the epoch of an object is necessarily part of the state of that object;

e) whether objects can negotiate on the basis of their current epoch identities;

f) the relation of epoch to the concepts of local and global time.

10.6 **Reference point**: An interaction point defined in an architecture for selection as a conformance point in a specification which is compliant with that architecture.

Significant classes of reference point are identified in ODP; details of these, and of the relationship of modelling to conformance, are given in clause 15.

10.7 **Conformance point**: A reference point at which behaviour may be observed for the purposes of conformance testing.

11 Properties of systems and objects

This clause describes the properties which may apply to an ODP system or part of an ODP system.

11.1 Transparencies

11.1.1 **Distribution transparency**: The property of hiding from the user some specific aspects of the system's complexity needed to support distribution.

NOTES

1 Users may include for instance end-users, application developers and function implementors.

2 Transparencies are often related to structuring into viewpoints. The requirements for transparencies are drawn from one or more of the designer oriented viewpoints, and are expressed in terms of the properties object interactions are to have in those viewpoints. The templates for the mechanisms that can provide object interactions with the right guaranteed properties are defined in other viewpoints.

11.2 Policy concepts

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

The specification of a contract may include:

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

b) Quality of Service constraints (see 11.2.2 and 11.2.3);

c) indications of duration or periods of validity;

d) indications of behaviour which invalidates the contract;

e) liveness and safety conditions.

NOTES

1 Objects in a contract need not be hierarchically related, but may be related on a peer-to-peer basis. The requirements in a contract are not necessarily applicable in the same way to all the objects concerned.

2 A contract can apply at a given reference point in a system. In that case, it specifies the behaviour which can be expected at the reference point.

3 An object template provides a simple example of a contract. An object template specifies the behaviour common to a collection of objects. As such, it specifies what the environment of any such objects may assume about their behaviour. Note that, for partial specifications, an object template leaves unspecified the behaviour of an object under certain environmental circumstances (e.g. particular interactions); the contract only extends to the specified behaviour.

11.2.2 Quality of Service: A set of quality requirements on the collective behaviour of one or more objects.

Quality of Service may be specified in a contract or measured and reported after the event.

The Quality of Service may be parameterized.

NOTE – Quality of Service is concerned with such characteristics as the rate of information transfer, the latency, the probability of a communication being disrupted, the probability of system failure, the probability of storage failure, etc.

11.2.3 Environment contract: A contract between an object and its environment, including Quality of Service constraints, usage and management constraints.

Quality of Service constraints include:

– temporal constraints (e.g. deadlines);

– volume constraints (e.g. throughput);

– dependability constraints covering aspects of availability, reliability, maintainability, security and safety (e.g. mean time between failures).

Usage and management constraints include:

– location constraints (i.e. selected locations in space and time);

– distribution transparency constraints (i.e. selected distribution transparencies).

Quality of Service constraints can imply usage and management constraints. For instance, some Quality of Service constraints (e.g. availability) are satisfied by provision of one or more distribution transparencies (e.g. replication).

Environment constraints can describe both:

– requirements placed on an object's environment for the correct behaviour of the object;

– constraints on the object behaviour in a correct environment.

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

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

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

11.2.7 Rule: A constraint on a system specification. Where appropriate, a rule can be expressed as an obligation, a permission or a prohibition.

NOTE – Rules may apply to the structure, behaviour or other properties of the system, including for example Quality of Service

11.2.8 Policy: A constraint on a system specification foreseen at design time, but whose detail is determined subsequent to the original design, and capable of being modified from time to time in order to manage the system in changing circumstances.

NOTES

1 Policies can be applied in any viewpoint; examples are an enterprise delegation policy, a computational persistence policy or an engineering scheduling or quality support policy.

2 The expectation of change is fundamental to the concept of policy, and a rule that does not envisage change is not a policy.

3 Policies may be expressed in terms of obligations, permissions or prohibitions, but this is not necessary for simple policies.

11.2.9 Policy Declaration: An element in a specification defined in order to allow incorporation of future constraints, together with rules determining the allowed form of acceptable constraints and the circumstances in which such constraints can be applied.

11.2.10 Policy Value: The specific constraints associated with a policy in some particular epoch.

11.2.11 Policy Envelope: The set of acceptable policy values that could be applied at a particular policy declaration. Restricting policy values to be within the policy envelope allows future flexibility but guarantees that the required properties of the system design will be preserved by all valid policies.

11.2.12 Policy Setting Behaviour: The behaviour defined in a specification via which a policy may be changed. A policy setting behaviour can be both an establishing behaviour (see 13.2.1) and a terminating behaviour (see 13.2.5).

11.3 Temporal properties

11.3.1 Persistence: The property that an object continues to exist across changes of contractual context (see 13.2.3) or of epoch.

11.3.2 Isochronicity: A sequence of actions is isochronous if every adjacent pair of actions in the sequence occupy unique, equally-sized, adjacent intervals in time.

12 Naming concepts

NOTE – Naming concepts and the mechanisms that support them in the context of distributed systems are further specified in ITU-T Rec X.910 | ISO/IEC 14771:1999, *Information technology – Open Distributed Processing – Naming framework*.

12.1 Name: A term which, in a given naming context, refers to an entity.

12.2 Identifier: An unambiguous name, in a given naming context.

12.3 Name space: A set of terms usable as names.

12.4 Naming context: A relation between a set of names and a set of entities. The set of names belongs to a single name space.

12.5 Naming action: An action that associates a term from a name space with a given entity.

All naming actions are relative to a naming context.

12.6 Naming domain: A subset of a naming context such that all naming actions are performed by the controlling object of the domain (the name authority object).

NOTE – "Naming domain" is an instance of the <X> domain concept (see 10.3).

12.7 Naming graph: A directed graph where each vertex denotes a naming context, and where each edge denotes an association between:

– a name appearing in the source naming context; and

– the target naming context.

NOTE – The existence of an edge between two naming contexts in a naming graph means that the target naming context can be reached (identified) from the source naming context.

12.8 Name resolution: The process by which, given an initial name and an initial naming context, an association between a name and the entity designated by the initial name can be found.

NOTE – The name resolution process does not necessarily provide sufficient information to interact with the designated entity.

13 Concepts for behaviour

13.1 Activity structure

13.1.1 Chain (of actions): A sequence of actions within an activity where, for each adjacent pair of actions, occurrence of the first action is necessary for the occurrence of the second action.

13.1.2 Thread: A chain of actions, where at least one object participates in all the actions of the chain.

An object may have associated with it one single thread or many threads simultaneously.

13.1.3 Joining action: An action shared between two or more chains resulting in a single chain.

13.1.4 Dividing action: An action which enables two or more chains.

There are two cases of a dividing action, depending on whether the enabled chains are required to join eventually.

13.1.5 Forking action: A dividing action, where the enabled chains must (subject to failure) eventually join each other, i.e. the enabled chains cannot join other chains and they cannot terminate separately.

13.1.6 Spawn action: A dividing action, where the enabled chains will not join. The enabled chains may interact and they may terminate separately.

13.1.7 Head action: In a given activity, an action that has no predecessor.

13.1.8 Subactivity: A subgraph of an activity which is itself an activity and which satisfies the following condition. For any pair of fork-join actions in the parent activity, if one of these actions is included in the subgraph, then both must be included in the subgraph.

13.2 Contractual behaviour

The concepts introduced here are illustrated in Figure 2.

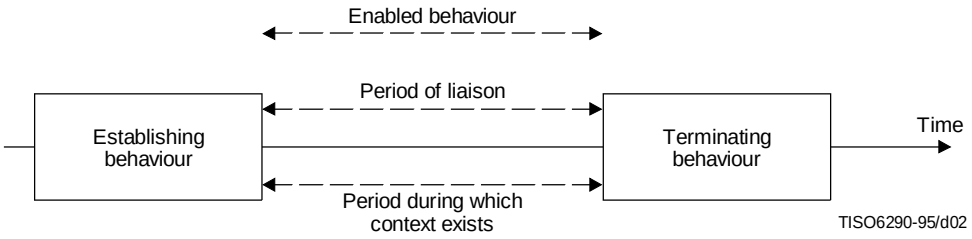

Figure 2 – Liaison and related concepts

13.2.1 Establishing behaviour: The behaviour by which a given contract is put in place between given objects. An establishing behaviour can be:

a) explicit, resulting from the interactions of objects that will take part in the contract; or

b) implicit, being performed by an external agency (e.g. a third party object, not taking part in the contract) or having been performed in a previous epoch.

NOTES

1 Negotiation is an example of a particular kind of establishing behaviour in which information is exchanged in the process of reaching a common view of permitted future behaviour.

2 Publication is an example of a particular kind of establishing behaviour in which information is distributed from one object to a number of others.

3 Explicit establishing behaviour must include an instantiation of the template associated with the contract. This may follow a possible negotiation/publication about which contract to set up and which template to instantiate, and with what parameters.

13.2.2 Enabled behaviour: The behaviour characterizing a set of objects which becomes possible as a result of establishing behaviour.

The enabled behaviour will not necessarily be the same for all objects involved in the contractual context.

13.2.3 Contractual context: The knowledge that a particular contract is in place, and thus that a particular behaviour of a set of objects is required.

An object may be in a number of contractual contexts simultaneously; the behaviour is constrained to the intersection of the behaviours prescribed by each contractual context.

NOTE – In OSI, the concept of a presentation context is an example of a contractual context which can be established at connection establishment time or subsequently.

13.2.4 Liaison: The relationship between a set of objects which results from the performance of some establishing behaviour; the state of having a contractual context in common.

A liaison is characterized by the corresponding enabled behaviour.

NOTES

1 Examples of liaisons which result from different establishing behaviours are:

a) a dialogue (as in OSI-TP);

b) a binding (see 13.5.2);

c) a distributed transaction (as in OSI-TP);

d) an (N)-connection (as in OSI);

e) an association between (N)-entities enabling them to participate in (N)-connectionless communication (as in OSI);

f) a relationship between files and processes which access the files.

2 Certain behaviours may be contingent on the establishment of multiple related liaisons. For example, a distributed transaction may depend on both the liaison between the transaction users and the supporting association. The liaison between the transaction users (the distributed transaction) may continue to exist, but be inactive, when the association is broken.

3 A liaison may involve more than two objects. The objects involved in a liaison do not necessarily all have equivalent roles. Thus, there may be liaisons for the collection or distribution of information. The number of participants and the roles of the participants are determined by the contract expressed by the liaison.

4 Agreeing a contract implies establishment of a contractual context and acceptance of any contractual obligation; the act of agreement to the contract makes the enabled behaviour possible for as long as the context or liaison exists. In practice, contexts may be arbitrarily nested and the enabled behaviour at an outer level may negotiate and agree a contract enabling further behaviour at an inner level of the hierarchy.

13.2.5 Terminating behaviour: The behaviour which breaks down a liaison and repudiates the corresponding contractual context and the corresponding contract.

A terminating behaviour must be explicitly identified as such in the contract if the establishing behaviour was explicit.

13.3 Service concepts

13.3.1 Service: A behaviour, triggered by an interaction, that adds value for the service users by creating, modifying, or consuming information; the changes become visible in the service provider's environment.

NOTES

1 Services are associated with interfaces and defined by the structural, behavioural and semantic rules of the interaction types involved.

2 A service can be characterized by a service type. A service is identifiable. A service may be composed of other services.

3 A service is in general invoked from within a liaison. Rules can be associated with the liaison, which refine the service for the duration of the liaison.

4 The service may be a complex behaviour, including both interactions and internal actions.

5 The provision of a service involves a collaboration between its provider and user. This collaboration may involve a complex series of interactions..

13.4 Causality

Identification of causality allows the categorization of roles of interacting objects. This clause gives a basic set of roles.

Causality implies a constraint on the behaviour of each of the participating objects while they are interacting. Causality will be identified in the definition of classes (or subclasses) to which interacting objects belong, or in the refinement of templates for their classes (or subclasses).

13.4.1 Initiating object (with respect to a communication): An object causing a communication.

NOTE – The identification of an initiating object with respect to a communication involves an interpretation of the intent of the communication.

13.4.2 Responding object: An object taking part in a communication, which is not the initiating object.

13.4.3 Producer object (with respect to communication): An object which is the source of the information conveyed.

The usage of this term does not imply any specific communication mechanism.

13.4.4 Consumer object (with respect to communication): An object which is a sink of the information conveyed.

The usage of this term does not imply any specific communication mechanism.

13.4.5 Client object: An object which requests that a service be performed by another object.

13.4.6 Server object: An object which performs some service on behalf of a client object.

Client/server relationships of a different nature (or level of abstraction) may exist between an object and different compositions of the objects with which it communicates.

NOTE – With respect to a specific interaction, client and server objects perform client and server roles respectively. However, a given object may be involved in a number of interactions and may perform client or server roles in each of these interactions; the interactions will, in general, be of different types and are not necessarily expressed at the same level of abstraction.

13.5 Establishing behaviours

13.5.1 Binding behaviour: An establishing behaviour between two or more interfaces (and hence between their supporting objects).

NOTE – "To bind" means "to execute a binding behaviour".

13.5.2 Binding: A contractual context, resulting from a given establishing behaviour.

Establishing behaviour, contractual context and enabled behaviour may involve just two object interfaces or more than two.

An object which initiates an establishing behaviour may or may not take part in the subsequent enabled behaviour.

Enabled behaviour (and, by analogy, contractual context) may be uniform (i.e. each participating object can do the same as every other) or non-uniform (i.e. one participating object has a different role from another, as in client and server).

There is no necessary correspondence between an object which initiates establishing behaviour and a particular role in non-uniform enabled behaviours (e.g. in a client-server contractual context, either object could validly have initiated the establishing behaviour).

13.5.3 Binding precondition: A set of conditions required for the successful execution of a binding behaviour.

The objects performing the binding behaviour must possess identifiers for all the interfaces involved in the binding. There may be additional preconditions.

13.5.4 Unbinding behaviour: A behaviour that terminates a binding, i.e. a terminating behaviour for the binding.

13.5.5 Trading: The interaction between objects in which information about new or potential contracts is exchanged via a third party object. It involves:

a) *Exporting:* The provision of an identifier to an interface which is claimed to meet some statement of requirements (i.e. offer a potential contract).

b) *Importing:* The provision of an identifier to an interface which matches a given statement of requirements, allowing a future binding behaviour to take place (i.e. the establishment of a contract).

13.6 Dependability

13.6.1 Failure: Violation of a contract.

NOTES

1 The behaviour specified in the contract is, by definition, the "correct behaviour". A failure is thus a deviation from compliance with the correct behaviour.

2 The ways an object can fail are called its failure modes. Several types of failure modes can be distinguished:

- arbitrary failures (non-compliance with the specification – the most general failure mode);
- omission failure (when expected interactions not take place);
- crash failures (persistent omission failures);
- timing failures (incorrectness due to untimely behaviour).

3 A failure can be perceived differently by different objects in the environment of the object that exhibits it. A failure may be: consistent if all the perceptions of the failure are the same; inconsistent if objects in the environment may have different perceptions of a given failure.

13.6.2 Error: Part of an object state which is liable to lead to failures. A manifestation of a fault (see 13.6.3) in an object.

NOTES

1 Whether an error will actually lead to a failure depends on the object decomposition, its internal redundancy, and on the object behaviour. Corrective action may prevent an error from causing a failure.

2 An error may be latent (i.e. not recognized as such) or detected. An error may disappear before being detected.

13.6.3 **Fault**: A situation that may cause errors to occur in an object.

NOTES

1 Faults causing an error may appear from the time an object is specified through to the time it is destroyed. Faults in an early epoch (e.g. design faults) may not lead to failure until a later epoch (e.g. execution time).

2 A fault is either active or dormant. A fault is active when it produces errors. The presence of active faults is determined only by the detection of errors.

3 Faults can be:

- accidental (that appear or are created fortuitously) or intentional (created deliberately);
- physical (due to some physical phenomena) or human-made (resulting from human behaviour);
- internal (part of an object state that may cause an error) or external (resulting from interference or interaction with the environment);
- permanent or temporary.

4 The definitions of fault, error and failure imply, recursively, causal dependencies between faults, errors and failures:

- a fault can lead to an error (it will lead to an error if it becomes active);
- an error can lead to a system's failure (it will lead to a failure unless the system can deal with it);
- a failure occurs when an error affects the correctness of the service delivered by a system (or system component).

13.6.4 **Stability**: The property that an object has with respect to a given failure mode if it cannot exhibit that failure mode.

14 Management concepts

Management in ODP is concerned with overall systems management, including application management and communication management.

14.1 **Application management**: The management of applications within an ODP system. Some aspects of applications management are common to all applications and are termed application independent management. Those aspects which are specific to a given application are termed application specific management.

14.2 **Communication management**: Management of objects which support the communication between objects within an ODP system.

14.3 **Management information**: Knowledge concerning objects which are of relevance to management.

14.4 **Managed role**: The view of the management interface of an object which is being managed within an ODP system.

NOTE – Where the object provides OSI communication services, OSI Management refers to the management interface as a managed object.

14.5 **Managing role**: The view of an object which is performing managing actions.

14.6 **Management notification**: An event notification initiated by an object operating in a managed role.

15 ODP approach to conformance

NOTE – The definitions of concepts in clauses 6-14 do not apply in this clause, and terms such as "interface", "system", and "role" are used in their normal English sense.

15.1 Conformance to ODP standards

Conformance relates an implementation to a standard. Any proposition that is true of the specification must be true in its implementation.

A conformance statement is a statement that identifies conformance points of a specification and states the behaviour which must be satisfied at these points. Conformance statements will only occur in standards which are intended to constrain some feature of a real implementation, so that there exists, in principle, the possibility of testing.

The RM-ODP identifies certain reference points in the architecture as potentially declarable as conformance points in specifications. That is, as points at which conformance may be tested and which will, therefore, need to be accessible for test. However, the requirement that a particular reference point be considered a conformance point must be stated explicitly in the conformance statement of the specification concerned.

Requirements for the necessary consistency of one member of the family of ODP standards with another (such as the RM-ODP) are established during the standardization process. Adherence to these requirements is called **compliance**.

If a specification is compliant, directly or indirectly, with some other standards, then the propositions which are true of those standards are also true of a conformant implementation of the specification.

15.2 Testing and reference points

The truth of a statement in an implementation can only be determined by testing and is based on a mapping from terms in the specification to observable aspects of the implementation.

At any specific level of abstraction, a test is a series of observable stimuli and events, performed at prescribed points known as reference points, and only at these points. These reference points are accessible interfaces. A system component for which conformance is claimed is seen as a black box, testable only at its external linkages. Thus, for example, conformance to OSI protocol specifications is not dependent on any internal structure of the system under test.

15.3 Classes of reference points

A conformance point is a reference point where a test can be made to see if a system meets a set of conformance criteria. A conformance statement must identify where the conformance points are, and what criteria are satisfied at these points. Four classes of reference points at which conformance tests can be applied are defined.

15.3.1 Programmatic reference point: A reference point at which a programmatic interface can be established to allow access to a function. A programmatic conformance requirement is stated in terms of a behavioural compatibility with the intent that one object be replaced by another. A programmatic interface is an interface which is realized through a programming language binding.

NOTE – For example, a programmatic reference point may be established in a database standard to support a language binding at some level of abstraction.

15.3.2 Perceptual reference point: A reference point at which there is some interaction between the system and the physical world.

NOTES

1 A perceptual reference point may be e.g. a human-computer interface or a robotic interface (specified in terms of the interactions of the robot with its physical environment).

2 A human-computer interface perceptual conformance requirement is stated in terms of the form of information presented to a human being and the interaction metaphor and dialogues the human may be engaged in. The specification of a human-computer interface in an ODP system specification is discussed in Rec. ITU-T X.903 | ISO/IEC 10746-3, Annex B.

3 A perceptual reference point may, for example, be established in a graphics standard.

15.3.3 Interworking reference point: A reference point at which an interface can be established to allow communication between two or more systems. An interworking conformance requirement is stated in terms of the exchange of information between two or more systems. Interworking conformance involves interconnection of reference points.

NOTE – For example, OSI standards are based on the interconnection of interworking reference points (the physical medium).

15.3.4 Interchange reference point: A reference point at which an external physical storage medium can be introduced into the system. An interchange conformance requirement is stated in terms of the behaviour (access methods and formats) of some physical medium so that information can be recorded on one system and then physically transferred, directly or indirectly, to be used on another system.

NOTE – For example, some information interchange standards are based on interchange reference points.

15.4 Change of configuration

The testing of conformance may take place at a single reference point, or it may involve some degree of consistency over use in a series of configurations involving several reference points. This may involve the testing of conformance to:

a) the requirement for a component to be able to operate after some preparatory process to adapt it to the local environment;

b) the requirement for a component to operate according to its specification at a particular reference point from initialization onwards;

c) the requirement for a component to continue to work when moved into a similar environment during operation.

The properties being tested above give rise to attributes of the objects or interfaces involved, as follows.

15.4.1 Portability: The property that the reference points of an object allow it to be adapted to a variety of configurations.

NOTE – If the reference point is a programmatic reference point, the result can be source-code or execution portability. If it is an interworking reference point, the result is equipment portability.

15.4.2 Migratability: The ability to change the configuration, substituting one reference point of an object for another while the object is being used.

15.5 The conformance testing process

Conformance is a concept which can be applied at any level of abstraction. For example, a very detailed perceptual conformance is expected to a standard defining character fonts, but a much more abstract perceptual conformance applies to screen layout rules.

The more abstract a specification is, the more difficult it is to test. An increasing amount of implementation-specific interpretation is needed to establish that the more abstract propositions about the implementation are in fact true. It is not clear that direct testing of very abstract specifications is possible at reasonable cost using currently available or foreseeable techniques.

Conformance testing can take many forms, mirroring different forms of specification, implementation and deployment processes. The description below concentrates on third-party testing, because this gives the clearest separation of roles, but the same distinctions and responsibilities can be identified in more integrated processes, such as those associated with modern, modularised, late-bound, loosely-coupled systems involving runtime conformance assurance performed by the distribution infrastructure, or otherwise

The testing process makes reference to a specification. To be complete, the specification must contain:

a) the behaviour of the object being standardized and the way this behaviour must be achieved;

b) a list of the primitive terms used in the specification when making the statements of behaviour;

c) a conformance statement indicating the conformance points, what implementations must do at them and what information implementors must supply (corresponding to the OSI notions of PICS and PIXIT).

In principle, there are two roles in providing a specification for testing: the system specifier and the system owner. The system specifier is responsible for the content of the specification and determines the possible behaviour of an implementation of that specification.

The system owner is responsible for the specific use to which the implementation will be put. The system owner may only be interested in a subset of the possible behaviour and, thus, may specify constraints on the scope of an implementation and, hence, on the nature of the conformance testing required.

There are two roles in the testing process: the implementor and the tester. The implementor constructs an implementation on the basis of the specification within the scope defined by the system owner. The implementor must provide a statement of a mapping from all the terms used in the specification to things or happenings in the real world. Thus, the interfaces corresponding to the conformance points must be indicated and the representation of signals given. If the specification is abstract, the mapping of its basic terms to the real world may itself be complex. For example, in a computational viewpoint specification (see Rec. ITU-T X.903 | ISO/IEC 10746-3), the primitive terms might be a set of interactions between objects. The implementor wishing to conform to the computational viewpoint specification would have to indicate how the interactions were provided, either by reference to an engineering specification or by providing a detailed description of an unstandardized mechanism (although this course limits the field of application of the implementation to systems in which there is an agreement to use the unstandardized mechanism).

The tester observes the system under test. Testing involves some shared behaviour between the testing process and the system under test. If this behaviour is given a causal labelling, there is a spectrum of testing types from:

a) passive testing, in which all behaviour is originated by the system under test and recorded by the tester;

b) active testing, in which behaviour is originated and recorded by the tester.

Normally, the specification of the system under test is in the form of an interface, as is the specification of the tester and test procedures. When testing takes place, these interfaces are bound.

The tester must interpret its observations using the mapping provided by the implementor to yield propositions about the implementation which can then be checked to show that they are also true in the specification, within the scope defined by the system owner.

15.6 The result of testing

The testing process succeeds if all the checks against the specification succeed. However, it can fail because:

a) the specification is logically inconsistent or incomplete, so that the propositions about the implementation cannot be checked (this should not occur);

b) the mapping given by the implementor is logically incomplete, so that it is inconsistent or observations cannot be related to terms in the specification; testing is impossible;

c) the observed behaviour cannot be interpreted according to the mapping given by the implementor. The behaviour of the system is not meaningful in terms of the specification, and so the test fails;

d) the behaviour is interpreted to give terms expressed in the specification, but these occur in such a way that they yield propositions which are not true in the specification, and so the test fails.

15.7 Relation between reference points

The flow of information between modelled system components may pass through more than one reference point. For example, a distributed system may involve interactions of two components A and B, but communication between them may pass in turn through a programmatic interface, a point of interconnection and a further programmatic interface.

A refinement of the same system may show interconnected components that have more than one component on the communication path between them.

In either case, conformance testing may involve:

a) testing the information flow at each of these reference points;

b) testing the consistency between events at pairs of reference points.

In general, testing for correct behaviour of a configuration of objects will require testing that statements about communication interfaces are true, but it will also require observation of other interfaces of these objects, so that the statements about the composition can also be checked.

The general notion of conformance takes into account the relation between several conformance points. Since the specification related to a given conformance point may be expressed at various levels of abstraction, testing at a given conformance point will always involve interpretation at the appropriate level of abstraction. Thus, the testing of the global behaviour requires coordinated testing at all the conformance points involved and the use of the appropriate interpretation at each point.

In particular, conformance of a template to a given programmatic interface can only be asserted when considering the language binding for the language in which the template has been written, and compliance of the written template to the language binding specification, which must itself be conformant with the abstract interface specification.

Index

NOTE – Associated with each index entry is the clause or subclause where the index entry is defined.