Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (0223) 323010**
INTERNATIONAL: **+44 223 323010**
FAX: **+44 223 359779**
E-MAIL: **apm@ansa.co.uk**

ANSA Phase III

RM-ODP Part 2 Descriptive Model 1993

Abstract

This is the Yokohama version of Part 2 of the Reference Model for Open Distributed Processing.It has been reformatted from the ascii version but not edited (some cross-references have been corrected). The figures were redrawn and are from the Frame version of the 1992 CD. The official version of Part 2 can be obtained from the ODP anonymous FTP site <brolga.cc.uq.oz.au> in directory RM-ODP.

APM.1045.00.03 **Draft** 26 August 1993

Standards Contribution

Distribution:
Supersedes:
Superseded by:

ISO/IEC COMMITTEE DRAFT

DRAFT ITU-T RECOMMENDATION

Information Technology - Basic Reference Model of Open Distributed Processing Part 2: Descriptive Model

0 Introduction

The rapid growth of distributed processing has lead to a need for a co-ordinating 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, interoperability and portability can be integrated.

The Basic Reference model of Open Distributed Processing (RM-ODP), Recommendations X.901 to X.905 | 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 modelling concepts presented can be used to specify all ODP viewpoints, although not every concept is necessarily applicable to every viewpoint.

The RM-ODP consists of:

Recommendation 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 intended to be understood and applied by its users, who may include standards writers and architects of ODP systems. It also contains an enumeration of required areas of standardization expressed in terms of the reference points for conformance identified in Recommendation X.903 | ISO/IEC 10746-3. This part is not normative.

Recommendation X.902 | ISO/IEC 10746-2: **Descriptive model**: 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 Recommendation X.903 | ISO/IEC 10746-3 and to establish requirements for new specification techniques. This part is normative.

Recommendation X.903 | ISO/IEC 10746-3: **Prescriptive model**: 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 Recommendation X.902 | ISO/IEC 10746-2. This part is normative.

Recommendation X.904 | ISO/IEC 10746-4: **Architectural semantics, specification techniques and formalisms**: contains a formalization of the ODP modelling concepts defined in Recommendation X.902 | ISO/IEC 10746-2 clauses 8 and 9. The formalization is achieved by interpreting each concept in terms of the constructs of the different standardized formal description techniques. This part is normative.

This Recommendation | part of ISO/IEC 10746 does not contain any annexes.

1 Scope and field of application

Recommendation X.902 | ISO/IEC 10746-2 consists of three main sections, covering the following areas:

a) the concepts which are needed to perform the modelling of ODP systems (in sections one and two);

b) the principles of conformance to ODP systems and the way in which they are applied (in section three).

The concepts defined in sections one and two are used in the Basic Reference Model of Open Distributed Processing to support the definitions of

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

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

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

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

2 References

There are no normative references in Recommendation X.902 | this part of ISO/IEC 10746.

3 Definitions and abbreviations

3.1 Definitions from other documents

There are no definitions from other documents in Recommendation X.902 | this part of ISO/IEC 10746.

3.2 Background definitions

3.2.1 Distributed processing: Distributed processing comprises that class of information processing activities in which discrete components may be located at more than one location, or where there is any reason which necessitates explicit communication among the components.

3.2.2 Open Distributed Processing: Distributed processing designed to conform to ODP standards.

3.2.3 ODP System: A system (see 5.5) which conforms to the requirements of ODP standards**.**

3.2.4 ODP standards: The Reference Model of ODP and those standards that comply with it, directly or indirectly.

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

Although information will necessarily have a representation form 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 representation forms 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 Framework (of viewpoints): a set of inter-related viewpoints such that correspondences may be asserted between specifications in different viewpoints. Concepts in one specification may be abstractions or refinements of concepts in another.

3.3 Abbreviations

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

Section one: Underlying modelling framework

4 Modelling concepts

This section and the following one “Architectural Concepts” define the concepts required to build and interpret the ODP Prescriptive Model (Recommendation X.903 | ISO/IEC 10746-3). The Prescriptive Model is a document which specifies the characteristics for distributed processing to be open. It has the following properties:

a) it is a complete set of modelling concepts that are relevant for all viewpoints of any conforming ODP system;

b) it is structured according to clause 4 of Recommendation X.902 | this part of ISO/IEC 10746;

c) it is written using a mixture of natural or formal, textual or graphical languages.

Each part of the model may be expressed in different languages, provided the division between each part is clear, and each may be consistently interpreted by its own semantics. These languages may be both graphical and textual. The language employed is used to maximize clarity, while minimizing ambiguity and inconsistency, so as to minimize confusion.

The modelling concepts required by the ODP Prescriptive Model may be characterized according to the model’s purpose. The modelling concepts may be categorized as follows:

d) **Basic interpretation concepts**: concepts for the interpretation of the modelling constructs of any ODP modelling language. These concepts are described in clause 5.

e) **Basic linguistic concepts**: concepts related to languages; the grammar of any language for the ODP Prescriptive Model must be described in terms of these concepts. These concepts are described in clause 6.

f) **Basic modelling concepts**: concepts for building the ODP Prescriptive Model; the modelling constructs of any language must be based on these concepts. These concepts are described in clause 7.

g) **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 8.

h) **Architectural concepts**: structuring 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 section two (clauses 9 to 14).

5 Basic interpretation concepts

Although much of the ODP Prescriptive Model will be concerned with defining formal constructs, the semantics of the model and any modelling languages used will 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, nor that they be used as the basis of formal definition of other concepts.

Any modelling activity will:

a) identify elements of the universe of discourse;

b) identify one or more pertinent levels of abstraction.

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

5.1 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.

5.2 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.

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

5.4 Atomicity: The property of anything that cannot be subdivided at the level of abstraction at which it is considered.

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

5.5 System: Something of interest both as a whole and as composed of parts. Therefore a system may be referred to as an entity and any component part may also be referred to as an entity.

The term system may often refer to an information system, but can equally validly be applied to any composite of interest. 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 ODP-RM and ODP standards are concerned with specific systems, i.e. information processing systems.

5.6 Architecture (of a system): A set of rules to define the structure of systems, their aggregation and the interrelationships between them.

6 Basic linguistic concepts

Whatever the concepts or semantics of a modelling language for the ODP Prescriptive Model, it 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 Prescriptive Model to the interpretation concepts.

6.1 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.

6.2 Name: A term which has been associated with an entity by some act of naming, which may be explicit or by application of some convention.

6.3 Identifier: A term which unambiguously refers to an entity.

6.4 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 Basic modelling concepts

All the basic concepts given in this clause are represented in models by the corresponding terms.

The concepts are defined here in English, but will be expressed in use in a particular specification notation. The detailed interpretation of the terms will depend on the 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.

7.1 Action: Something which happens.

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

An action is not necessarily directly observable; i.e. the environment (see 7.3) of an object may not be able to take part in an action when the action is internal to the object.

NOTES

1- When used without qualification, action means “action occurrence”.

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

7.2 Object: A model of an entity. An object is characterized by its behaviour (see 7.8) and, dually, by its state (see 7.13). 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 7.10) with its environment (see 7.3).

An object interacts (see 7.10) with its environment at its interaction points (see 7.11). The location of an object is the union of the locations of the actions in which the object may take part (see 7.4).

Informally, an object is said to perform functions and offer services (an object which make a function available is said to offer a service), but, for modelling purposes, these are specified in terms of the behaviour of the object and of its interfaces.

An object can perform more than one function. A function can be performed by the cooperation of several objects.

NOTES

1 - The concept of service and function are used informally to express the purpose of a piece of standardization. In the ODP family of standards, function and service are expressed formally in terms of the specification of the behaviour of objects and of the interfaces which they support. A special activity or purpose of an object is called a function.

2 - The expression “use of a function” is a shorthand for the interaction with an object which performs the function. Thus the expression “structure of functions” means the structure of a set of objects which provides various functions.

7.3 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 7.10), representing the process of observation (see 7.9).

7.4 Location: An interval of arbitrary size in time and space at which an action can occur.

NOTE - The extent of the interval in time and space is chosen to reflect the requirements of a particular specification task and the properties of a particular specification language. A single location in one specification may be subdivided in either time or space (or both) in another specification.

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

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

7.7 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).

7.8 Behaviour (of an object): A collection of actions with a set of constraints on the circumstances in which they may possibly occur.

The nature of the constraints which may be expressed depends on the specification language in use.

In general, there exists a set of possible sequences of observable actions (see 7.9) consistent with any given behaviour.

A behaviour may include internal actions.

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

NOTE - The composition (see 8.1) 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.

7.9 Observable action: An action is described as observable with respect to a given object if it can occur only when the object and its environment are ready to participate in that action.

NOTE - An unobservable or internal action always occurs within an object without the participation of its environment.

7.10 Interaction: An observable action normally involving two or more objects. However, an object may interact with itself.

NOTES

1 - Self-interaction implies that the object concerned plays more than one role (see 8.10) in the interaction.

2 - Interactions may be labelled in terms of cause and effect relationships between the participating objects. The concepts that support this are discussed in 12.3.

7.11 Interaction point: A point at which interactions may occur.

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.

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

NOTE - Communications may be labelled in terms of a cause and effect relationship between the participating objects. Concepts to support this are discussed in 12.3.

7.13 State (of an object): At a given instant in time, the condition of an object that determines the set of all sequences of actions in which the object can take part.

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, only indirectly as a result of the interactions in which the object takes part.

7.14 Interface: An abstraction of the behaviour of an object obtained by hiding observable actions of that object outside a specified subset.

The set of interfaces of an object constitutes a partition of the observable actions of that object.

NOTE - The abstraction performed to isolate the behaviour of the interface consists of the suppression of the identity and description of all actions which do not occur in the interface. Instead, these actions become internal actions in the interface behaviour; behaviour at other interfaces of the object introduces non-determinism as far as the interface being considered is concerned.

8 Specification concepts

8.1 Composition:

a) (of objects) A combination of two or more objects yielding a new object. The characteristics of the new object are determined by the objects being combined and by the way they are combined.

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 means 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 - Action and activity are degenerate cases of behaviour.

8.2 Composite object: An object expressed as a composition.

8.3 Decomposition: The specification of a given object as a composition.

Composition and decomposition are dual terms and dual specification activities.

8.4 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 set of criteria imposes constraints on the allowed behaviour of the environment. If the set of criteria is 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 set of 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 we have coerced behavioural compatibility. If no modification is necessary, then we have natural behavioural compatibility.

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

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 8.7) and template types (see 8.16) 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 8.9). b) if U and V are object template types, S and T are said to be behaviourally compatible if their corresponding templates are.

8.5 Refinement: Refinement is 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 clause 8.4), 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.

8.6 Trace: A record of an object's observable behaviour, 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.

8.7 <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>s can be any of:

object

interface

action

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

A template may be specified using parameters.

The definition given here is generic; the precise form of the template will depend on the specification technique used. In particular, this applies to the parameter types when a template is specified using parameters.

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

An observable action template describes a set of observable actions.

8.8 Interface signature: A specified subset of the observable action templates for a given object.

For each interface there is a corresponding signature, but there may be many interfaces with the same signature.

8.9 Instantiation (of an <X> template): The process which, using a given <X> template and other necessary information, results in the existence of a new <X>. <X> can be any of:

object

interface

If <X> is an object, it is instantiated in its initial state, and can thereafter participate in interactions.

The new <X> may be called an instantiation of the given <X> template. The new <X> exhibits the features specified in the given template.

The definition given here is generic; the precise nature of instantiation will depend on the specification language used and the chosen form of the <X> template. Typically, instantiation may involve actualization of parameters, using the information available during the instantiation process.

Actualization of parameters during an <X> template instantiation may in turn involve other <X> template instantiations or binding (see 12.4) of existing interfaces.

8.10 Role: Identifier for a behaviour, which may appear as a parameter in a template for a composite object, and which is associated with one of the component objects of the composite object.

Specification of a template as a composition of roles enables the instantiation process to be explained as the association of a specific component of the resultant composite object with each role. The association of a component object with a role may result from the actualization of a parameter.

8.11 Creation (of an <X>): The process of instantiation, performed by an object, resulting in the existence of a new <X>. <X> can be any of:

object

interface

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.

8.12 Deletion (of an <X>): The process, performed by an object, of destroying an instantiated <X>. <X> can be any of:

object

interface

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

8.13 Introduction (of an object): The process of instantiation of an object, when it is achieved by a mechanism which is not covered by the model.

NOTE - Any object is either created or introduced, but not both.

8.14 Type (of an <X>): A predicate characterizing an <X>. An <X> is of the type, or satisfies the type, if the predicate holds for that <X>.

The notion of type implicitly classifies the entities into categories, some of which may be of interest to the specifier (see the concept of Class 8.15).

<X> can be any of:

object

interface

action

8.15 Class (of <X>s): The set of all <X>s satisfying a type. The elements of the set are referred to as members of the class. <X> may be any of:

object

interface

action

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.

8.16 Template type (of a <X>): A predicate expressing that an <X> is an instance of an <X> template.

A template-type is a special kind of type:

a) it is used for typing <X>s; and

b) it is associated with a specific template that has been defined.

Different representations of a template-type may be logically equivalent (for example because the templates used in each representation are equivalent).

8.17 Template class (of a <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.

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.

8.18 Subclass/superclass: One class is a subclass of another class (and the second is a superclass of the first) precisely when the first class is a subset of the second.

The subclass relation can be seen to induce a preorder (a reflexive, transitive relation) on a set of templates.

NOTE - Subclass and instance (see 8.17) are defined in terms of each other (via other concepts). As a result, this part of the standard defines only their relationship. Complete definitions can be provided in each specification technique by defining the preorder mentioned above. Figure 1 illustrates the relationships between some of these concepts.

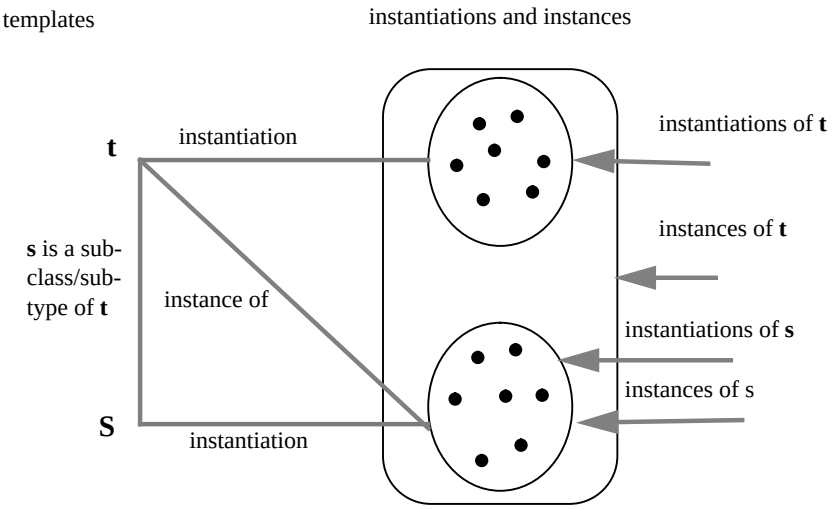

Figure 1: Relationship between templates, instantiations and instances

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.

8.19 Subtype/supertype: All <X>s which satisfy one type also satisfy a second type if and only if the first is a subtype of the second, and the second is a supertype of the first.

One type is a subtype of a second type precisely when the class associated with the first is a subclass of the class associated with the second.

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

The object template subtype/supertype relation does not necessarily coincide with behavioural compatibility. If S and T are object templates such that the S template type is a subtype of the T template type, an S-instantiation is in general not behaviourally compatible with a T-instantiation.

NOTE - Since each template determines a single template type and template class, we may shorten statements of the form “the class of template A is a subclass of template B” to “template A is a subclass of template B” or “template A is a subtype of template B”.

8.20 Instance (of an <X> template): An <X> is an instance of a specified <X> template when it is an instantiation of some <X> template that is a subclass of the specified <X> template.

Since the subclass relation is reflexive, for an <X> to be an instance of an <X> template it is sufficient for an <X> to be an instantiation of the specified <X>-template. Since a template defines a single template class, we may talk about an <X> being an instance of an <X> template class. An instance of a template class is an instance of all the superclasses of the class.

Since the object template subtype/supertype relation does not necessarily coincide with behavioural compatibility, instances of a given object template need not be behaviourally compatible with instantiations of the same template. They do coincide provided

a) a transitive behavioural compatibility relation is considered, and

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

8.21 Derived class/parent class: If the template of a class is an incremental modification of the template of a second class (i.e. results from a change to the template of the second class), then the first is a derived class of the second class, and the second class is a parent class of the first.

The incremental modification relating templates must ensure that self-reference or recursion in the template of the parent 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 parent template to obtain the derived one.

Classes can be arranged in an inheritance hierarchy according to derived class/parent class relationships. These relationships are not necessarily transitive.

It is possible for one class to be a subtype 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.

8.22 Incremental inheritance: The derivation of a new template (the derived template) by incrementally modifying an existing template (the parent template).

The type of the derived template can be referred to as the derived type. The type of the parent template is the parent type. If a template has more than one immediate parent template, then the inheritance resulting is said to be multiple.

NOTE - In a wider community, there are two main varieties of inheritance; these correspond in ODP to incremental inheritance and to the subtype/supertype relation.

8.23 Inclusion polymorphism: The property that instances of one class are behaviourally compatible with instances of a second class.

8.24 Invariant: A predicate that is true for the entire life time of an object.

8.25 Precondition: A predicate that must be true for an action to occur.

8.26 Postcondition: A predicate that must be true immediately after the occurrence of an action.

Section two: Architectural concepts

9 Organizational concepts

9.1 <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 programs 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) replicated group: a communicating group whose purpose is to provide a certain level of tolerance against some faults.

9.2 Configuration (of objects): A collection of objects able to interact at interfaces. For each interface in the configuration, the set of objects involved is determined.

A configuration goes beyond the specification of the behaviour of the individual objects. The specification of the configuration may be static or may be in terms of the operation of dynamic mechanisms which change the configuration, such as object binding and unbinding (see 12.4).

A configuration can be expressed in terms of the concepts of concurrent composition. However, there is a need to study whether the composition operations are necessarily associative. The process of composition generates an equivalent object for the configuration.

9.3 <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.

The controlling object knows 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 9.5) 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 - Examples of specialized domains are**:**

Domain	**Relationship**	**Member Class**	**Controlling Class**
Security authorization domain	authorized by	processing object	security authority
Management domain	managed by	managed object	managing object
Addressing domain	address allocated by	addressed object	addressing authority object
Naming domain	name allocated by	named object	name authority object

9.4 Subdomain: A domain which is a subset of a given domain.

9.5 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 or communicating 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 composed in 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. In Figure 2, objects A and B and B and C can communicate at all times, but objects A and C cannot.

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.

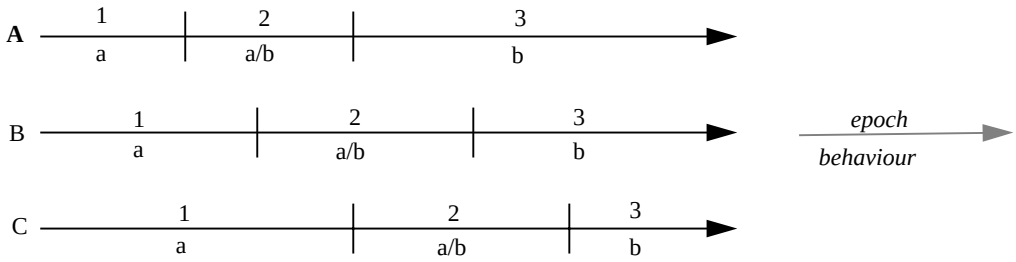

Figure 2: Communication of objects in different epochs

9.6 .Conformance point: An interaction point declared in a standard as a point at which behaviour may be observed for the purposes of conformance testing

9.7 Reference point: A potential conformance point defined in an architecture for selection as an actual 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 Properties of systems and objects

This clause describes the properties of an ODP system or part of an ODP system.

10.1 Transparencies

10.1.1 Transparency: The property of hiding from a particular user the potential behaviour of some parts of the system.

The concept of transparency has much wider applicability than simply to the modelling of ODP systems. Transparency results from the normal process of abstraction and is found in many areas outside distribution. Consideration here is limited to the requirements for various forms of distribution transparency.

10.2 Obligations

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

10.2.2 Contract: An agreement between a set of objects governing part of their collective behaviour. A contract places obligations on the objects involved.

The components of a contract may include:

a) a template, that specify the different roles that objects involved in the contract may assume;

b) quality of service attributes (see 10.2.3);

c) indications of duration or periods of validity;

d) indications of behaviour which invalidates the contract.

NOTES

1 - Objects in a contract need not be hierarchically related, but may be in a relation 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 - The notion of a contract is important to capture the idea that the behaviour of an application or an object must be evaluated against precise expectations.

4 - A simple example of a contract is provided by a class template. A class 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, a class template leaves unspecified the behaviour of an object under certain environmental circumstances (e.g. particular interactions); the contract only extends to the specified behaviour.

10.2.3 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, etc..

10.2.4 Policy: A prescriptive relation between one or more objects and some behaviour which establishes a norm for the correctness of the behaviour. A policy can be expressed as an obligation, a permission or a prohibition.

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

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

10.2.6 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.

10.3 Temporal properties

10.3.1 Persistence: The property of an object that continues to exist across changes of context (see 12.2.3) or of epoch.

10.3.2 Isochronicity: The property of a sequence of actions in which every adjacent pair of actions in the sequence occupy unique, equally-sized, adjacent locations in time.

11 Naming and identification

11.1 Name space: A set of terms usable as names, defined by a given predicate.

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

NOTE - All naming is relative to a naming context.

11.3 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.

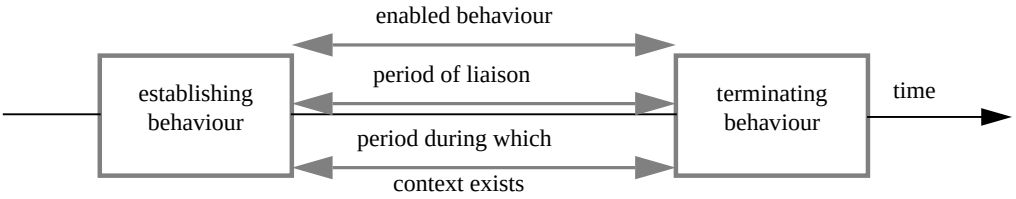

Figure 3: Liaison and related concepts

11.4 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. The name resolution process provides necessary but not necessarily sufficient information to access the designated entity.

12 Concepts for behaviour

12.1 Localization of behaviour

12.1.1 Chain (of actions): A sequence of actions where, for each adjacent pair of actions, occurrence of the first action makes possible occurrence of the second action.

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

12.1.3 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.

12.1.4 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.

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

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

12.1.7 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 acitivity, if one of these actions is included in the subgraph, then both must be included in the subgraph.

12.1.8 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.

12.2 Contractual behaviour

The concepts introduced here are illustrated in Figure 3.

12.2.1 Establishing behaviour: The behaviour by which a given contract is put in place between given objects.

An establishing behaviour can be:

a) explicit, i.e. result from the interactions of objects that will take part in the contract; or

b) implicit, i.e. be performed by an external agency (e.g. a third party object, not taking part in the contract) or the contract may have been put in place in a previous epoch.

The motivation of an establishing behaviour is to establish some future activity. An establishing behaviour results in the objects involved agreeing on a given contract governing their future behaviour.

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.

12.2.2 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.

12.2.3 Context: The knowledge that a particular contract is in place, and thus that a particular behaviour may occur.

This may be viewed as a restriction and/or enhancement of the potential behaviour of an object relative to some overall scope for behaviour arising from the establishment of a particular contract. An object may be in a number of contexts simultaneously, in which case the behaviour is constrained to the intersection of the behaviour of each individually.

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

12.2.4 Liaison: The relationship between a set of objects which results from the performance of some establishing behaviour; the state of having a 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 12.4.2);

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 behaviour 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 it.

4 - A liaison establishes a context in which a sequence of engagement liaisons can take place. Each engagement liaison establishes a context in which a behaviour characteristic of the contract takes place (see Figure 4).

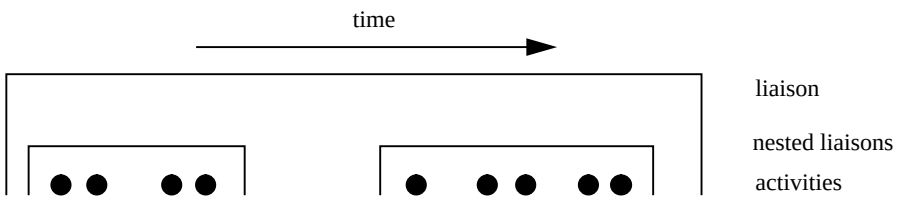

Figure 4: Relation of liaisons

5 - Thus there is a duality between context and contractual obligation acceptance of specification and enabled behaviour. Although Figure 4 shows only two levels of liaison, in practice structures may be arbitrarily nested and an engagement liaison at one level can also agree a contract which permits an inner level liaison.

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

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

12.3 Causality

Identification of causal relationships allows the categorization of roles of interacting objects. This clause gives an initial set of roles.

Causality is a constraint to which the behaviour of each of the participating objects conform during the interaction. Causality will be (specified) identified in the definition of classes (or sub classes) to which interacting objects belong, or in the refinement of templates for their classes (or subclasses).

12.3.1 Initiating object: An object causing a communication.

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

12.3.3 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.

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

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

12.3.5 Client object: An object which requests the performance of some function by another object.

12.3.6 Server object: An object which performs some function 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.

12.4 Establishing interactions

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

12.4.2 Binding (noun): A context, resulting from a given establishing behaviour.

When a binding exists, there is an appearance of an end-to-end path between the objects bound.

Establishing behaviour, 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 enabling behaviour.

Enabled behaviour (and, by analogy, 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 context, either object could validly have initiated the establishing behaviour).

12.4.3 Binding precondition: A set of conditions required for the successful execution of a binding process.

A necessary but not sufficient precondition for the establishing behaviour is the possession of interface references (i.e. identifiers) for all the interfaces involved by the objects performing the establishing behaviour.

12.4.4 Unbinding (verb): A terminating behaviour.

12.4.5 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 some reference to a potential interface which is claimed to meet some statement of requirements (i.e. offer a potential contract);

b) importing: the provision of some reference to a potential interface which matches a given statement of requirements, allowing a future binding interaction (i.e. the establishment of a contract).

12.5 Reliability

12.5.1 Failure:

a) (noun) Deviation of an object behaviour from compliance with the specification of its correct behaviour.

b) (verb) Transition from correct behaviour to incorrect behaviour.

NOTES

1 - 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); stopping failures (when the object does not take part in any further action); omission failure (a special case of stopping failure where expected correct behaviour does not take place); crash failures (persistent omission failures); timing failures (incorrectness due to untimely behaviour).

2 - 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.

12.5.2 Error: Part of an object state which is liable to lead to failures. A manifestation of a fault 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. An error may be handled before leading to failure.

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

12.5.3 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 an error. 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).

12.5.4 Stability: An object has stability with respect to a given failure mode if it cannot exhibit that failure mode.

13 Management concepts

13.1 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.

13.2 Communication management: Management of communication objects within an ODP system.

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

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

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

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

13.6 Notification: An interaction initiated by an object operating in a managed role.

14 Object classes

14.1 Communications objects: Objects which support the conveyance of information between objects involved in a communication.

A configuration of objects may be transformed into a different configuration with explicit communication objects by a sequence of decompositions and compositions. A related process may result in the identification of selected objects and their communication environment.

14.2 Infrastructure objects: Objects concerned with how a system of objects is to be implemented or supported.

14.3 Device objects: Objects that encapsulate physical resources.

A device object frequently encapsulates some form of transducer to manipulate physical phenomena.

14.4 Interpreter objects: Objects that support the interpretation of executable object templates. When interpreting executable templates, an interpreter object behaviour is equivalent to one or more threads.

Section three: Conformance in ODP

15 ODP approach to conformance

15.1 Conformance to ODP standards

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

Conformance is expressed in an ODP standard if and only if that standard contains a conformance statement.

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. Only standards which contain a conformance statement are entitled “specifications”.

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) is established during the standardization process. This is called compliance. If a specification is compliant, directly or indirectly with some other standards then the propositions which are true in those standards is also true in 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

Recommendation X.902 | ISO/IEC 10746-2 defines four classes of reference points at which conformance tests can be applied.

A conformance point is a reference point where a test can be made of an object to see if it meets a set of conformance criteria. A conformance statement for an object must identify where the conformance point is, and what criteria are satisfied at this point.

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.

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 computerized part of the system and the outside world. This may be a human-computer interface or a collection of sensors and actuators, such as an industrial robot. A robotic interface conformance requirement is expressed in terms of the interaction of the robot with its environment. 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.

NOTE - For example, a perceptual reference point may be established in a graphics standard.

15.3.3 Interworking reference point: A reference point at which a communication interface can be established to allow interoperability between two 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 ability to configure an object at any one of a number of different reference points before the function to be accessed is used.

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.

There are two roles in the testing process: the initiator and the tester. Both make 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).

The first role is that of the implementor, who constructs an implementation on the basis of the specification. 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 real 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, the primitive terms might be a set of interactions between objects. The implementor wishing to confirm 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 second role is that of the tester, who observes the system under test. Testing involves some shared behaviour between the tester and the system under test. If this behaviour is given a causal labelling, there is a spectrum of testing types from

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

2) 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.

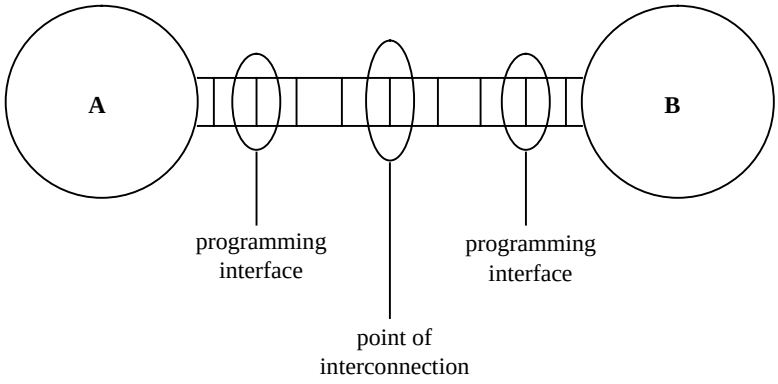

Figure 5: Reference points in a computational model

15.6 The result of testing

The testing process succeeds if all the checks against the specification succeed. However, it may 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 computational model description of 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 (see Figure 5).

In an engineering viewpoint description of the same system, interconnected components may have more than one component on the communication path between them (see Figure 6).

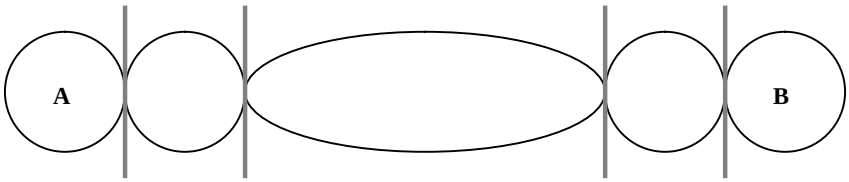

Figure 6: An engineering elaboration of figure 5

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 Figure 7, the testing for correct behaviour of the interconnection of objects A and B will clearly require the testing that statements about interface I_1 are true, but it will also, in general, require observation of interfaces I_2 and I_3 so that statements about the composition and statements relating I_1 to I_2 and I_1 to I_3 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

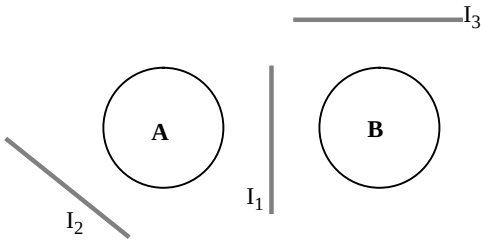

Figure 7: Interfaces

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 to concepts defined

<X> Template	8.7	7
<X>-Domain	9.3	11
<X>-Group	9.1	11
Abstraction	5.3	3
Action	7.1	4
Activity	7.7	5
Application management	13.1	17
Architecture (of a system)	5.6	4
Atomicity	5.4	4
Basic interpretation concepts	5	3
Basic linguistic concepts	6	4
Basic modelling concepts	7	4
Behaviour (of an object)	7.8	5
Behavioural compatibility	8.4	6
Binding (noun)	12.4.2	16
Binding (verb)	12.4.1	16
Binding precondition	12.4.3	16
Causality	12.3	16
Chain (of actions)	12.1.1	14
Change of configuration	15.4	20
Class (of <X>s)	8.15	8
Classes of reference points	15.3	19
Client object	12.3.5	16
Communication	7.12	6
Communication management	13.2	17
Communications objects	14.1	18
Composite object	8.2	6
Composition	8.1	6
Concepts for behaviour	12	14
Configuration (of objects)	9.2	11
Conformance point	9.6	12
Conformance to ODP standards	15.1	19
Consumer object (with respect to communication)	12.3.4	16
Context	12.2.3	15
Contract	10.2.2	12
Contractual behaviour	12.2	14
Creation (of an <X>)	8.11	8
Decomposition	8.3	6
Deletion (of an <X>)	8.12	8
Derived class/parent class	8.21	10
Device objects	14.3	18
Dividing action	12.1.3	14
Enabled behaviour	12.2.2	15
Entity	5.1	3
Environment (of an object)	7.3	5

Epoch	9.5	11
Error	12.5.2	17
Establishing behaviour	12.2.1	14
Establishing interactions	12.4	16
Failure	12.5.1	17
Fault	12.5.3	17
Forking action	12.1.4	14
Head action	12.1.6	14
Identifier	6.3	4
Inclusion polymorphism	8.23	10
Incremental inheritance	8.22	10
Infrastructure objects	14.2	18
Initiating object	12.3.1	16
Instance (of an <X> template)	8.20	10
Instantiation (of an <X> template)	8.9	7
Interaction	7.10	5
Interaction point	7.11	6
Interchange reference point	15.3.4	19
Interface	7.14	6
Interface signature	8.8	7
Interpreter objects	14.4	18
Interworking reference point	15.3.3	19
Introduction	0	1
Introduction (of an object)	8.13	8
Invariant	8.24	10
Isochronicity	10.3.2	13
Joining action	12.1.2	14
Liaison	12.2.4	15
Localization of behaviour	12.1	14
Location	7.4	5
Location in space	7.5	5
Location in time	7.6	5
Managed role	13.4	17
Management concepts	13	17
Management information	13.3	17
Managing role	13.5	18
Migratability	15.4.2	20
Modelling concepts	4	3
Name	6.2	4
Name resolution	11.4	14
Name space	11.1	13
Naming and identification	11	13
Naming context	11.2	13
Naming graph	11.3	13
Notification	13.6	18

Object	7.2	5
Object classes	14	18
Obligation	10.2.1	12
Obligations	10.2	12
Observable action	7.9	5
ODP approach to conformance	15	19
Organizational concepts	9	11
Perceptual reference point	15.3.2	19
Permission	10.2.5	13
Persistence	10.3.1	13
Policy	10.2.4	13
Portability	15.4.1	20
Postcondition	8.26	10
Precondition	8.25	10
Producer object (with respect to communication)	12.3.3	16
Programmatic	15.3.1	19
Programmatic reference point	15.3.1	19
Prohibition	10.2.6	13
Properties of systems and objects	10	12
Proposition	5.2	3
Quality of service	10.2.3	13
Reference point	9.7	12
References	2	2
Refinement	8.5	7
Relation between reference points	15.7	21
Reliability	12.5	17
Responding object	12.3.2	16
Role	8.10	8
Scope and field of application	1	1
Sentence	6.4	4
Server object	12.3.6	16
Spawn action	12.1.5	14
Specification concepts	8	6
Stability	12.5.4	17
State (of an object)	7.13	6
Subactivity	12.1.7	14
Subclass/superclass	8.18	9
Subdomain	11	
Subtype/supertype	8.19	9
System	5.5	4
Template class (of a <X>)	8.17	9
Template type (of a <X>)	8.16	9
Temporal properties	10.3	13
Term	6.1	4
Terminating behaviour	12.2.5	16
Testing and reference points	15.2	19
The conformance testing process	15.5	20
The result of testing	15.6	21

Thread	12.1.8	14
Trace	8.6	7
Transparencies	10.1	12
Transparency	10.1.1	12
Type (of an <X>)	8.14	8
Unbinding (verb)	12.4.4	16