INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.930
(09/98)

SERIES X: DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

Open distributed processing

Information technology – Open distributed processing – Interface references and binding

ITU-T Recommendation X.930

(Previously CCITT Recommendation)

ITU-T X-SERIES RECOMMENDATIONS

DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

PUBLIC DATA NETWORKS	
Services and facilities	X.1–X.19
Interfaces	X.20–X.49
Transmission, signalling and switching	X.50–X.89
Network aspects	X.90–X.149
Maintenance	X.150–X.179
Administrative arrangements	X.180–X.199
OPEN SYSTEMS INTERCONNECTION	
Model and notation	X.200–X.209
Service definitions	X.210–X.219
Connection-mode protocol specifications	X.220–X.229
Connectionless-mode protocol specifications	X.230–X.239
PICS proformas	X.240–X.259
Protocol Identification	X.260–X.269
Security Protocols	X.270–X.279
Layer Managed Objects	X.280–X.289
Conformance testing	X.290–X.299
INTERWORKING BETWEEN NETWORKS	
General	X.300–X.349
Satellite data transmission systems	X.350–X.399
MESSAGE HANDLING SYSTEMS	X.400–X.499
DIRECTORY	X.500–X.599
OSI NETWORKING AND SYSTEM ASPECTS	
Networking	X.600–X.629
Efficiency	X.630–X.639
Quality of service	X.640–X.649
Naming, Addressing and Registration	X.650–X.679
Abstract Syntax Notation One (ASN.1)	X.680–X.699
OSI MANAGEMENT	
Systems Management framework and architecture	X.700–X.709
Management Communication Service and Protocol	X.710–X.719
Structure of Management Information	X.720–X.729
Management functions and ODMA functions	X.730–X.799
SECURITY	X.800–X.849
OSI APPLICATIONS	
Commitment, Concurrency and Recovery	X.850–X.859
Transaction processing	X.860–X.879
Remote operations	X.880–X.899
OPEN DISTRIBUTED PROCESSING	**X.900–X.999**

For further details, please refer to ITU-T List of Recommendations.

INTERNATIONAL STANDARD 14753

ITU-T RECOMMENDATION X.930

INFORMATION TECHNOLOGY – OPEN DISTRIBUTED PROCESSING – INTERFACE REFERENCES AND BINDING

Summary

Interface references are crucial to interworking between ODP systems and federation of groups of ODP systems. An interface reference embodies the information needed to establish bindings, including binding to objects in different management domains. An interface reference further embodies information required for the engineering mechanism to maintain computational bindings in the presence of distribution transparencies, such as migration transparency. (Example: keywords to keep include "engineering mechanism", "computational binding", "information", and "distribution transparency".) This Recommendation | International Standard includes:

- a framework for binding interfaces and a generic binding protocol (for both stream and operational interfaces);
- a specification of the generic information structure of interface references (for both stream and operational interfaces);
- representation(s) for interface references when transferred using standardized protocols;
- identification of procedures for the management and transfer of interface references with respect to individual transparencies;
- identification of node management interfaces related to binding and federation which create or transform interface references.

Source

The ITU-T Recommendation X.930 was approved on the 25th of September 1998. The identical text is also published as ISO/IEC International Standard 14753.

CONTENTS

Introduction

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

One of the components of the architecture is the ODP binding function. The binding function provides the means to establish liaisons and create channels across autonomous systems in order to support interworking and communication between objects. An interface reference embodies the information needed to establish bindings and further embodies the information required to maintain bindings between computational objects in the presence of distribution.

INTERNATIONAL STANDARD

ITU-T RECOMMENDATION

INFORMATION TECHNOLOGY – OPEN DISTRIBUTED PROCESSING – INTERFACE REFERENCES AND BINDING

1 Scope and Field of application

1.1 Scope

Interface references are crucial to interworking between ODP systems and federation of groups of ODP systems. An interface reference embodies the information needed to establish bindings, including binding to objects at nodes that support several different communication protocols and binding to objects in different management domains. An interface reference further embodies the information required for the engineering mechanism to maintain bindings between computational objects in the presence of distribution transparencies such as migration transparency. They are the foundation of ODP location and relocation transparency.

This Recommendation | International Standard includes:

- a framework for binding interfaces and a generic binding protocol (for both stream and operational interfaces);
- a specification of the generic information structure of interface references (for both stream and operational interfaces);
- representation(s) for interface references when transferred using standardized protocols;
- identification of procedures for the management and transfer of interface references with respect to individual transparencies;
- identification of node management interfaces related to binding and federation which create or transform interface references;
- identification of requirements for quality of service information and for invocation of QoS or related measurement procedures.

This Recommendation | International Standard provides an engineering description of the functionality needed to support the computational binding of objects in ODP systems. Security and support for group communication are important issues, but not within the scope of this Recommendation | International Standard.

1.2 Field of Application

This Recommendation | International Standard enables interworking between ODP systems.

2 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 the currently valid ITU-T Recommendations.

2.1 Identical Recommendations | International Standards

- ITU-T Recommendation X.901 (1997) | ISO/IEC 10746-1:1998, *Information technology – Open distributed processing – Reference Model:* Overview.
- ITU-T Recommendation X.902 (1995) | ISO/IEC 10746-2:1996, *Information technology – Open distributed processing – Reference Model:* Foundations.
- ITU-T Recommendation X.903 (1995) | ISO/IEC 10746-3:1996, *Information technology – Open distributed processing – Reference Model:* Architecture.
- ITU-T Recommendation X.910 (1998) | ISO/IEC 14771:1999, *Information technology – Open distributed processing – ODP Naming framework.*
- ITU-T Recommendation X.931 (1998) | ISO/IEC 14752:1999, *Information technology – Open distributed processing – Protocol Support for Computational Interactions.*
- ITU-T Recommendation X.950 (1997) | ISO/IEC 13235-1:1998, *Information technology – Open distributed processing – Trading function: Specification.*
- ITU-T Recommendation X.960[1)] | ISO/IEC 14769[1)], *Information technology – Open distributed processing – Type repository function.*
- ISO/IEC 9075-1[1)], *Information technology – Database language SQL – Part 1: Frame.*

2.2 Specifications of the Object Management Group

- CORBA: The Common Object Request Broker: Architecture and Specification, Revision 2.1, Object Management Group, August 1997 (OMG Doc Number Formal/97-09-01).

Temporary Note: A reference explanatory report is circulated with the DIS ballot on this specification.

3 Definitions

3.1 Definitions in this Recommendation | International Standard

This Recommendation | International Standard defines the following terms.

3.2 Definitions from other Recommendations | International Standards

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

- <X> domain;
- <X> template;
- action;
- activity;
- behaviour;
- binding;
- compliance;
- configuration;
- conformance point;
- contract;

1) To be published.

- contractual context;
- distribution transparency;
- environment contract;
- epoch;
- failure;
- interaction point;
- interface;
- interworking reference point;
- liaison;
- location;
- notification;
- policy;
- quality of service;
- role;
- subtype;
- type (of an X).

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

- <X> federation;
- announcement;
- basic engineering object;
- binder;
- channel;
- compound binding;
- engineering interface reference;
- explicit binding;
- implicit binding;
- interceptor;
- node;
- operation interface;
- protocol object;
- signal;
- signature;
- stub;
- stream interface.

4 Abbreviations

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

QoS	Quality of Service
ODP	Open Distributed Processing
IIOP-IOR	Internet Inter-ORB Protocol – Interoperable Object Reference

5 Conventions

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

In diagrams:

- objects are represented as ovals or circles;
- the symbol '$\perp$' protruding from an object represents an interface;
- the symbol '<>' represents a containment of objects;
- the symbol '♦ ' represents dependent containment between objects;
- the symbol 'n .. *' denotes that the cardinality of an association must exceed n.

6 Overview of interface references and binding

6.1 Rationale

The objective of ODP standardization is to develop standards that realize the benefits of distributing information processing services in a heterogeneous environment of IT resources and multiple organizational domains. These standards address constraints on system specifications and provide a system infrastructure that addresses difficulties inherent in the design and programming of distributed systems.

Distributed systems are important because of a growing need to interconnect information processing systems. This need arises because of organizational trends (such as downsizing), which demand the exchange of information not only between groups within an organization but also between cooperating organizations. Technological advances make it possible to respond to these trends giving increased importance to information networks and personal workstations, and enabling distributed applications to be constructed across large configurations of interconnected systems.

In order to set up cooperation between organizations and their information systems, the parties must define and agree on a relationship and then maintain it. This relationship is often defined as a contract in commercial environments. To achieve cooperation between systems, after some initial contact, agreements must be made, contracts negotiated and interfaces defined, created and made available. Interworking between ODP systems requires standardized communication methods between objects that reside at autonomous systems.

This Recommendation | International Standard provides a framework for binding, including a refinement of the binding model of ITU-T Rec. X.903 | ISO/IEC 10746-3 and a generic structure of interface references. This Recommendation | International Standard is structured according to ODP viewpoints.

6.2 Overview of the binding process

6.2.1 Obtaining interface references

Whenever an interface is created (either explicitly, or during object creation), an interface reference for it is generated. This interface reference can be passed via existing communication channels from the object providing the interface. Its recipients can then pass it on, possibly via several steps, until it reaches some object which wishes to interact with the interface.

The interface reference contains sufficient information to initiate the binding process which makes interaction involving the interface possible. Often, an object will create a binding involving itself and an interface whose interface reference it has just received. However, in the general case, the creation of a binding involves a set of interfaces, not necessarily including an interface to the object performing the binding action. Such third party bindings may occur, for example, when setting up multimedia streams.

An object which is to create a binding must have information on:

a) the set of interface references for the interfaces to be bound;

b) the type of the binding needed, possibly in the form of a reference to a suitable binding template;

c) the quality of service required of the binding.

6.2.2 Binding process

This description is in terms of compound binding, in which a visible computational binding object is created. The simpler implicit or explicit primitive binding process is generally similar.

In a computational specification, an object creates a binding by performing a binding action. From an engineering point of view, it does this by invoking a binding factory, representing the mechanisms needed to allocate resources, negotiating detailed quality agreements and establishing communications paths.

In either viewpoint, the information outlined above is required to start the process. The result is the creation of a binding object and the return of an interface reference for a control interface which it provides. This control interface allows the initiator, or any other object it passes the reference to, to control the binding, to request notification of significant events, or to request destruction of the binding. The exact detail of this interface depends on the binding type.

Once the binding has been created, it can support the behaviour, in terms of operation invocations or stream flows, defined by the binding type.

6.2.3 Negotiating the properties of the binding

The way an object can interact with its environment depends on the capabilities (in terms of available protocols, stubs, etc.) of the infrastructure supporting the object, and on a set of quality of service constraints defined in the object's environment contract.

When an interface is created, the interface reference contains information about these capabilities, together with enough naming information to allow the interface to be located or relocated, and possibly some items from the object's environment contract to indicate levels of service which can be achieved. This information indicates properties of the interface which will be true for any binding it may become involved in, and therefore provides the starting point for the negotiation of binding properties.

In order to support interface reference passing and binding across federation boundaries, and to keep the size of interface references within reasonable bounds, the information to be passed may be included in shorthand form, or by reference to supporting services, rather than being explicitly encoded in the reference. The detailed format of an interface reference may be transformed as it is passed from object to object.

The binding factory combines the information in the interface references it receives with constraints in the binding type or from the initiator of the binding, to steer the negotiation of the binding's properties, and to decide on the level of resources required. This process may involve negotiation with the objects being bound to take into account their current availability of resources and aspects of their environment contracts which were not included in the interface references. A binding can only be created if a satisfactory set of properties can be identified which is consistent with the requested binding action and the environment contracts of all the objects involved in the binding.

6.2.4 Renegotiating the properties of the binding

In many situations, it may be necessary for a binding to evolve during its lifetime, either to change its properties or to modify the set of interfaces being bound. The kind of changes that can occur will depend on the type of the binding, and this will be constrained by the capabilities of the engineering infrastructure available to support it. For example, elaborate facilities for modifying bindings are likely to be required in an environment supporting mobile or nomadic computing platforms.

Changes to the binding configuration, or to the quality of service being either required or offered, will generally involve renegotiation between the participants, and may result in the addition or replacement of some supporting components. For example, the migration of an object into a different kind of environment may require modification, and thus renegotiation, of quality of service, involving the use of different network facilities, different data representations and different protocols.

6.2.5 Quality monitoring and control

In addition to the ability to modify the quality of service using the control interface of the binding object, there is, in general, a need to monitor the quality of service actually achieved. To do this, monitoring mechanisms may need to be attached to specific reference points at each of the bound interfaces.

The maintenance of agreed quality of service may involve the creation of internal feed back processes linking the observation of achieved quality at or between interfaces with modification by some quality management object of the requirements on particular parts of the binding, using the control interface to the binding object.

6.2.6 Destroying a binding

The definition of when a binding ceases to exist is part of its behaviour, and so depends on its type. A binding will generally cease to exist as a result of a request to do so, received at its control interface. It may also cease to exist as a result of an action internal to the binding, such as detection of a failure of communication or of one or more of the objects being bound.

Destruction of a binding does not, in general, imply the destruction of the interfaces being bound, or of the objects providing those interfaces.

7 Enterprise viewpoint

The purpose of the binding function is to bind together interfaces (signal, operational, stream) to enable communication between objects. The binding function selects and names the communication interfaces, checks that these interfaces conform to each other, checks that these interfaces initially satisfy the QoS requirements of each other, and forms a liaison between the interfaces. The binding liaison guarantees that the objects can interact. The binding function also provides for the management of the binding and eventual destruction of the binding.

Binding actions are of two kinds: primitive binding actions in which the objects involved are modelled as interacting directly and compound binding actions in which an intermediate object represents the mechanisms providing the binding.

Transferring operation invocations and implementing binding actions require support of the mechanisms and functions of the RM-ODP infrastructure.

7.1 Communities

The roles involved in the binding community are target interface creator, binding initiator, unbinding initiator, target interface, binding factory, binding liaison, binding controller, and channel.

The binding community has three epochs. In one epoch the initiator is bound to the binding factory. In another epoch the targets are members of a binding liaison. In the third epoch, the binding liaison has been terminated.

In addition, the binding community is supported by a channel, and therefore, the binding community may alternate between the epochs with and without an established channel.

7.2 Roles

7.2.1 Binding initiator

The binding initiator is the role of an object that initiates binding establishment between some targets by activating the binding factory.

7.2.2 Unbinding initiator

The unbinding initiator is the role of an object that initiates binding termination.

7.2.3 Binding controller

The binding controller is the role of an object that modifies the existing channel's properties via the control interface provided by the channel. The binding controller itself may offer an interface for controlling and managing the binding liaison it supports.

7.2.4 Target interface creator

A role of an object that initiates target interface creation. There are two cases, one in which the target interface creator requests a new interface to be created by the infrastructure, and one in which the target interface creator creates a new interface on itself dynamically. In either case, a reference is associated with the interface. This interface reference is past to the binding initiator.

Target objects are those objects that have a need to interact and may assume the role of target interface creator.

7.2.5 Target interface

A target interface is the interface where the initiator wants the activity to take effect. Target interfaces are bound together either directly within a cluster or via a channel.

Interfaces are either operational interfaces or stream interfaces.

NOTE – The expression of quality of service properties may require the refinement of either operational interfaces or stream interfaces in terms of signal interfaces.

7.2.6 Binding factory

The binding factory represents the infrastructure for channel creation. The binding factory may itself be a federated entity that has local representatives in each administrative domain involved in the channel instantiation activity. Federation issues are discussed in clause 10.

7.2.7 Binding liaison

A binding liaison is a community providing a common contractual context where two or more objects agree about the mechanism they use for interaction. The objects therefore share common information. The agreement may include quality of service statements.

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

Binding liaisons can be qualified by QoS assertions that further constrain their correct behaviour (e.g. to constrain end-to-end communication delay or end-to-end delay-jitter at a recipient interface). Where such QoS assertions are made, the points in space and time at which QoS observations are made must also be specified.

7.2.8 Channel

A channel represents a concrete realization of a binding liaison that enables interactions to occur between target objects.

A channel is responsible for maintaining the quality of communication and distribution transparency of communication. The channel includes objects such as stubs, binders, protocol objects and interceptors. These objects support the transport of operation invocations and terminations, information flows, and signals. Their functionality and behaviour is defined in ITU-T Rec. X.930 | ISO/IEC 14753.

7.3 Activities

Activities of the binding community include binding, unbinding, binding management, and event notifications.

7.3.1 Interface creation

Interface creation is a chain of actions that involves the creation of an interface and distribution of their interface references to potential binding initiators.

7.3.2 Binding

Binding is a chain of actions, where the initiator adds a target to the binding liaison. If the binding liaison does not yet exist, it is created. The initiator specifies the model of interaction (signal, stream, operation) and selects the binding type. The binding liaison ensures that the interfaces are identifiable, conformant and that a channel either exists or can be created between the objects.

Direct use of binding actions is called *explicit* binding. Explicit binding can be used for all interface types. In the case of operation interfaces RM-ODP also specifies that binding can be *implicit*, in order to allow the use of notations that do not provide for the expression of bind actions.

There are two kinds of binding actions: primitive binding actions and compound binding actions. A primitive binding action binds two computational objects directly. A compound binding action can be expressed in terms of primitive binding actions linking two or more computational objects via a binding object. The presence of a binding object in a computational binding gives the means to express configuration and quality of service control.

In explicit binding, a control interface is created. This interface is the means by which binding management activities occur.

7.3.3 Unbinding

Unbinding is a chain of actions, where the initiator removes a target object from the binding liaison. If the binding liaison becomes empty, it may be deleted, depending on the defined behaviour of the binding.

The effect of deleting a binding liaison on the components of a channel is determined by the behaviour of the channel.

7.3.4 Binding management

Binding management provides the means to change the binding liaison and thus the internal configuration of the channel.

The control interfaces of a channel provide functions including:

- monitoring the use of the channel;
- monitoring changes to the channel;
- authorizing changes to the channel;
- changing the membership of the binding liaison;
- changing the pattern of communication enabled by the binding liaison;
- controlling and changing the quality of service in the binding liaison;
- deleting the binding liaison as a whole;
- control of notification of errors that disrupt the channel: this would allow specification of an interface at which the object invokes a notification operation if failures disrupt the binding;
- control of a dynamic multicast liaison, allowing the addition of new consumers and removal of existing consumers.

Rebinding may be needed to restore a bound configuration after a failure. Rebinding is an internal management activity of the binding channel.

7.3.5 Event notification

Event notification is an activity of the channel object. It reports contract violations during communication to the communicating partners, and potentially to the controller.

7.4 Policies

The roles of any initiator and target can be combined.

Binding and unbinding need not to be initiated by the same object.

The binding initiators may pass information about target interfaces and control interfaces to other objects.

The initiator determines the initial properties of the binding channel.

The controller role can be fulfilled jointly by multiple objects.

The channel enables interactions between engineering objects by:

- providing conversion of data carried by interactions over the channel;
- applying controls to, and keeping records of, interactions over the channel;
- providing conversion of protocol used in the interactions;
- providing measurement and control of achieved quality of service;
- enabling migration and relocation of interfaces linked by the channel;
- enabling failure, persistence, replication and transaction transparencies.

The channel may be able to reconfigure.

The objects participating in the binding enterprise community must establish policies for:

- distributing the interface references;
- quality and capability of a binding liaison;
- renegotiating and modifying a binding liaison; and
- destroying a binding liaison.

7.5 Rules

The role of initiator for the various activities in 7.3 may be assumed by separate objects.

Only the channel within each binding community creates, manages and deletes resources. Resources are managed with the help of the infrastructure.

All bindings, even local interface bindings, can fail.

The number of simultaneous binding liaisons an interface can participate in is determined by the supporting engineering infrastructure or by the behaviour of the object concerned.

The binding factory is obliged to check that the preconditions of a binding liaison are fulfilled. The preconditions for compound binding are that, for each formal role in the binding object template (i.e. to each place where an object can be bound):

- the corresponding interface parameter must be of the same kind (signal, stream or operation) as the interface template associated with the formal role in the binding object template;
- the corresponding interface parameter must be of complementary causality to the interface template associated with the formal role in the binding object template;
- the corresponding interface parameter must be a subtype of the signature type of the interface template associated with the formal role in the binding object template;
- the interfaces have such quality of service properties available as the peers require; security requirements are part of the quality of service attributes.

In the case of binding of stream interfaces, the binding liaison may abstract from application specific stream composition rules. To determine whether two stream interfaces can be bound, it must be guaranteed that the individual flows are compatible. The liaison may be responsible for the binding of two or more stream interfaces.

Channel must invoke an event notification in case of failure. Failure means inability to behave according to a contract. Failures to be notified include breach of quality of service agreements and breach of behaviour specifications.

NOTE – Examples for stream management are included in Annex C.

8 Information viewpoint

Computationaly, binding process is an activity where the binding factory establishes a binding liaison between two or more target interfaces, based on the information retrievable from the set of interface references and the type of the binding liaison.

In the engineering description, the type of the binding liaison is captured as a binding type. A binding type can be realized by various channel types, i.e., the binding type defines restrictions to the engineering of channels. Depending on the system domain, a channel type can be associated with different channel templates. A binding factory can realize a channel by parametrizing a channel template with the interface references.

NOTE 1 – The practical negotiation protocols are performed on information contained in channel types. However, binding types form a necessary abstraction level for creating a mapping between channel types at separate engineering domains, by representing restrictions on channel types. The actual channel instantiation process requires that the suitable channel templates are located at each involved domain.

The agreements on behaviour within the binding liaison, e.g. QoS contracts and transparency support, are captured as a binding contract. Interface references can only include information about the actual abilities of the objects they represent, i.e. the environment contract of an object constrains the content of the interface reference. Once an object becomes a member of a binding liaison, the binding contract reflecting that liaison becomes part of the environment contract of that object.

The information objects related to binding actions are binding contracts, environment contracts, interface references, binding types, and channel templates.

The information view of the binding system comprises:

- a set of binding types,
- a set of channel templates,
- a set of objects, with environment contracts and interface references associated to them.

NOTE 2 – The set of binding types and the set of channel templates can be evolved dynamically via administrative actions. However, those actions are not within the scope of this Recommendation | International Standard.

Figure 1 gives an overview of the information view of the binding system.

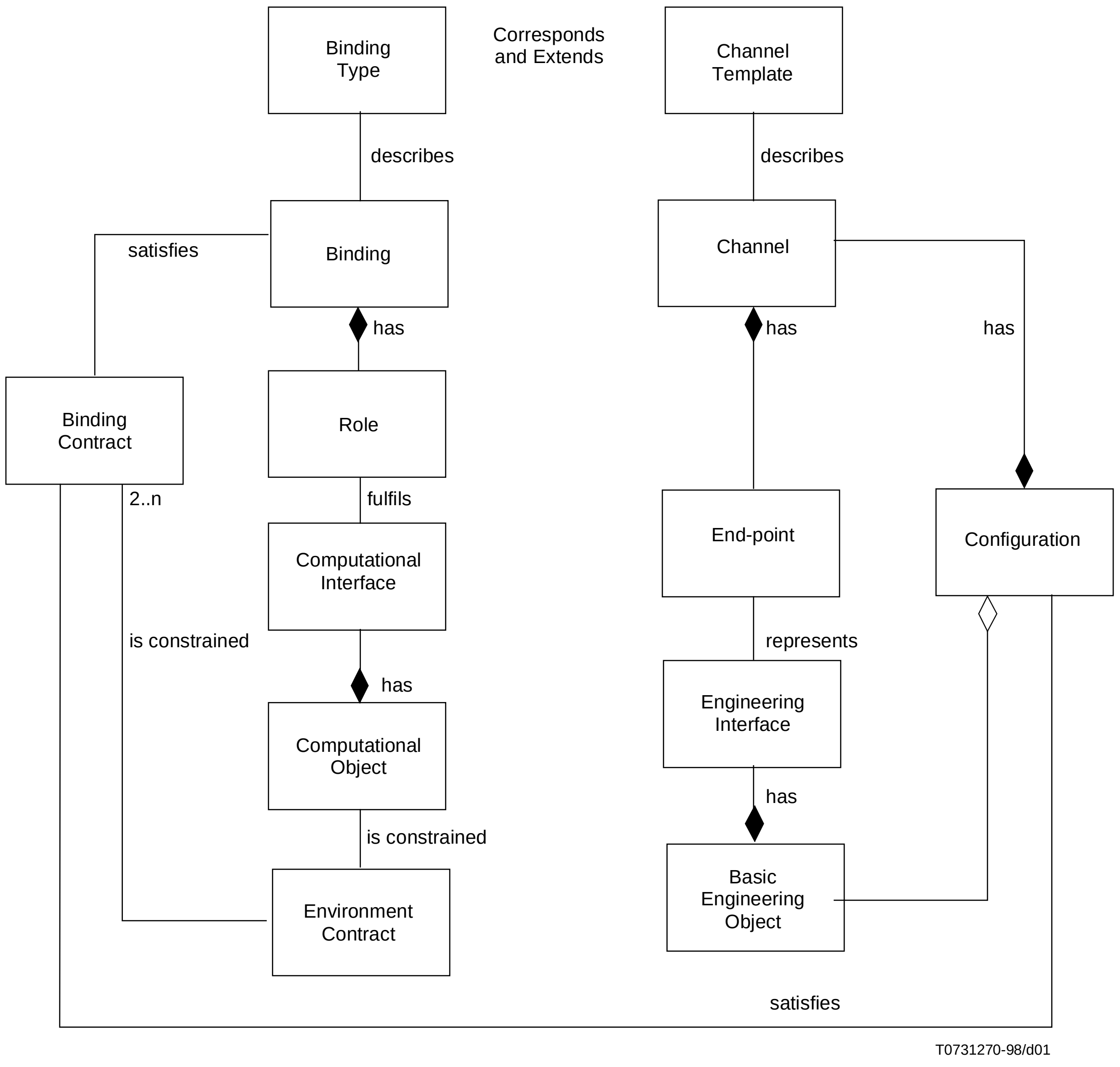


Figure 1 – Information viewpoint overview

8.1 Binding contract

A binding contract is established as a result of negotiation about the properties and capabilities of a particular binding liaison. A binding contract captures the QoS agreement between the interfaces that are bound. The negotiation process can be performed either by the binding factory or by other means outside the scope of this Recommendation | International Standard.

NOTE 1 – The information of a binding contract can be captured either within interface references or referred to from interface references. This Recommendation | International Standard does not specify the structure.

NOTE 2 – Various optimization scenarios presented in 9.4 imply separation of binding contract information (e.g. QoS) from the interface reference.

NOTE 3 – In cases, where federation between computational objects is required, the negotiation process benefits from an external structure that supports binding liaison renegotiation.

8.2 Environment contracts

Objects that wish to communicate must present their properties to the infrastructure so that the infrastructure is able to bind them based on the available information. The advertised information is presented as interface references. Other information about the environment may be needed in later phases of channel creation. The concept of environment contracts is introduced in ITU-T Rec. X.903 | ISO/IEC 10746-3.

Interface references are largely determined and restricted by the environment contract of the involved object.

The way an environment contract is expressed will, in general, be determined as part of the detailed design of the system providing the environment. Environment contract notation will not, therefore, be completely standardized.

8.3 Binding type

Binding type specifies the rules for the binding liaison in regard to:

- roles of target objects required in the liaison;
- interface types of the target objects involved;
- interface types of binding controller interfaces;
- required channel functionality, including selected aspects of distribution transparency, QoS monitoring, and security support;
- required behaviour in cases of failure of channel functionality.

8.4 Channel type

The channel type is corresponding to the binding type and expressed in engineering terms. A channel type specifies the requested behaviour of the binding, expressed in terms of roles:

- the required channel functionality, including selected aspects of distribution transparency, QoS monitoring, and security support;
- required behaviour in cases of failure of channel functionality.

8.5 Channel template

A channel template is a refinement of a channel type and contains sufficient information for channel instantiation. This information specifies the configuration of stubs, binders, protocol objects, and interceptors, created during channel establishment.

NOTE – A channel template can contain alternative configurations to be applied in selected circumstances. For example, if communication paths are insecure, encrypting stubs might be required.

The channel creation process may be federated and heterogeneous, and subject to optimization (see 9.4).

8.6 Interface references

An interface reference is a structured identifier for an interface, containing or implying an interface type and sufficient information to allow a binding to that interface to be established. The interface reference is established when the interface is created and is used to instantiate a compatible channel structure to access the interface.

Interface references enable:

- the identification of an engineering object interface that is available for distributed binding; the identification is based on engineering interface reference domains;
- a binding to be established to the engineering object interface that it identifies;
- the detection and repair of distributed bindings invalidated by engineering object relocation;
- transformation to and from engineering interface references in other engineering interface reference domains;
- support of groups; an interface reference may refer to a group of engineering objects.

Interface reference contents must be considerably flexible, for example because:

- ODP computational objects may be at arbitrary levels of granularity which will effect the binding of interfaces to network addresses. For example in an RPC-based system each interface may have a distinct address. An object oriented database management system might associate many (small) object interfaces with a single network address.
- Different transparencies put different demands on the binding mechanisms and require different kinds of information.
- ODP systems may involve interworking between federated domains which use incompatible representations of addresses; the interceptors which link such domains may need to transform interface references.
- ODP systems may operate over a variety of lower level infrastructures which have already established interface reference contents and binding procedures. ODP interface references must be able to "wrap" such references without loss of information.

ODP objects have the potential to be relocated from one node to another, for example as part of object migration, or as part of object persistence. As a consequence, the network addresses associated with an object's interfaces are liable to change. Information in an interface reference must be sufficient to enable the current network address of an interface to be determined, and for interactions targeted at previous locations of the interface to be correctly redirected, particularly in a federated situation.

QoS related information of interface references include items such as:

- required transparency services;
- contract concerning level of security;
- contract concerning level of guaranteed resources;
- timeliness;
- synchronization;
- failure related behaviour in cases of security, communication or resource failure;
- required level of binding behaviour auditing.

8.6.1 General interpretation

Interface references are unambiguous identifiers for the interfaces they reference. The property of unambiguity arises from the complete collection of information in the interface reference structure, rather than from any particular field or fields within it.

The amount of information embodied in an interface reference is potentially large. Direct encoding of this information as data may be inefficient. An alternative is to define procedures for objects to exchange short names for interface references and call back for further information when it is required (see 8.6.5).

In general, the representation of an interface reference is specific to an engineering domain, in which there is one appropriate naming authority and binding is managed in a uniform way. Such a domain is called an interface reference domain. Within such a domain, well-known interfaces (such as a trader interface or an interpreter for non-interpreted references) can be referenced using a local shorthand, although this must be expanded if the reference leaves the domain.

In general, any of the fields in the interface reference may need to be translated to yield a different detailed representations if the reference is passed across a domain boundary. Direct references may need to be converted to non-interpreted references, or vice versa.

Errors or changes in system configuration which occur after the interface reference has been created may invalidate one or more of the fields in the interface reference. Use of the relocator refreshes out-of-date fields.

8.6.2 Definition of structures

This subclause gives a conceptual description of interface reference contents. The suggested partitions are further explained in 8.6.3. Information items denoted in this abstract description can be considered either as embedded in the interface reference itself or as references for accessing the information.

```
<interf-ref>                     ::= <null> | <direct-reference> | <non-interpreted-reference>
<direct-reference>               ::= <interf-type><causality-info><channel-class><location-info>
                                     <relocation-info><group-info><security-info><interf-QoS-info>
                                     <additional-info>
<non-interpreted-reference>      ::= <interpreter-reference> <opaque-info>
<interf-type>                    ::= <stream-interf-type> | <operational-interf-type>
```

8.6.3 Definition of fields

8.6.3.1 Interface type

The interface type may be represented by a type name specific to the local interface reference domain, or by an interface reference to a type description.

Type description references take the form specified by the ITU-T Rec. X.960 | ISO/IEC 14769 Type Repository Function.

The `interf-type` is thus represented either by:

a) a name; or

b) an interface reference to a type repository interface and an identifier for the interface type.

NOTE – The use of indirect interface type definitions stored in type repository allows the interface type definition structures to be evolved. Thus, the information contents of stream interface references, operational interface references, and signal interface references may vary depending on the type system used. As the information contents associated with interface references may evolve, also new functionality can be introduced to exploit that information (e.g. contract management of explicit bindings).

8.6.3.2 Causality information

Each interface signature contains an indication of causality in respect to the interaction in which it participates. Causality denotes the role that the interface plays in the interaction.

For example, the `causality` for an operational interface is either client or server. For a stream interface, the causality of producer or consumer is expressed separately for each flow. For a signal interface, the causality of initiator or responder is expressed separately for each signal.

8.6.3.3 Channel class

The `channel-class` field may be represented by a name specific to the local interface reference domain, or by an interface reference to a template description.

NOTE – The channel templates can be stored in a distributed repository, where each repository can have a private, platform-specific subtemplate. So the binding factory is able to derive a suitable subtemplate in each domain for the federated instantiation process.

In either case, the `direct-reference` part identifies the channel template for the channel which is constructed when binding to the interface. This information is needed to configure the remote end of the channel using equivalent stub, binder and protocol objects. This information, together with the location of the interfaces to be bound, identifies the supporting objects needed for the channel (e.g. interceptor).

For channels depending on the group function, the channel template must contain sufficient information to identify the corresponding group management policies.

The channel class is thus represented either by:

a) a name; or

b) an interface reference to a type repository interface and a definition identifier for the channel template.

8.6.3.4 Location information

The `location-info` field provides the necessary information for the construction of a binding to the location of the interface at the time it was created. The information includes network and sufficiently unique node-specific addressing. An interface reference may include a set of different pieces of addressing information, corresponding to different access paths. Location information will, in general, be context sensitive, because of this association with possible access paths.

Location information may be abbreviated to a domain specific shorthand, or may be in the form of an interface reference to, and handle for use at, a specific supporting object.

The format of the location information will depend on the channel class in the interface reference.

The location information is thus a set, each member of which is represented either by:

a) a network address and node specific addressing information; or

b) an interface reference to a supporting object and a handle in a form required by that supporting object. The definition of this supporting object and its interface will be the subject of future standardization.

NOTE – The format of the location information is specific to the engineering domain concerned.

8.6.3.5 Relocation information

The `relocation-info` field identifies a relocator object which can be queried if a binding involving the interface reference fails, either during creation or subsequently. The relocation information is kept distinct from the location information, because the supporting objects involved are required to maintain different state information and play different roles in the infrastructure. For example, the relocator is involved directly in object migration and deactivation/reactivation.

The relocation information is thus an interface reference to a supporting object. The definition of this supporting object and its interface will be the subject of future standardization.

8.6.3.6 Group information

NOTE – This Recommendation | International Standard details only peer communication, but provides also a framework for further extensions on the area of group communication. This item `group-info` is reserved for further standardization.

8.6.3.7 Security information

NOTE – The format of the security information and security mechanism will be the subject of future standardization.

8.6.3.8 Interface quality of service information

The interface QoS information captures information about the (potential) quality of service agreement associated with an interface. For the interpretation of the information, also the information structure (type description) need to be included or referred.

The `interf-QoS` information can be expressed either as:

- direct attribute values; or
- QoS conformance statements.

The use of QoS conformance statements may require that the binding liaison behaviour contain capabilities for negotiating the QoS level, monitoring the actually reached QoS and adaptation to changed behaviour of the supporting environment.

NOTE 1 – The format of the quality of service information, together with the negotiation and monitoring mechanisms involved, will be the subject of future standardization.

NOTE 2 – The QoS negotiation, monitoring and binding adaptation processes are being studied in the project on QoS in ODP.

NOTE 3 – The QoS monitoring and (re)negotiation processes may require prescription of some QoS information items that are not part of the interface binding process. For instance, the roles played by the interfaces in the QoS negotiation process may need to be stored. Such aspects are outside the scope of this Recommendation | International Standard.

The appropriate of QoS information type may depend on the interface type.

NOTE 4 – The QoS type information and the QoS attribute type information can be stored to a type repository. The attribute identification scheme can be supported by an appropriate naming facility.

8.6.3.9 Additional information

The `additional-info` is manipulated by functions not directly related to binding, such as the engineering interface reference tracking function.

NOTE – The format of the additional information will be the subject of future standardization as and when necessary. Some kind of tagged structure will be required to support the requirements of a variety of additional functions.

8.6.3.10 Non-interpreted reference

The `non-interpreted-reference` is used in systems crossing different interface reference domains where different naming policies apply (e.g. CORBA object reference versus ANSA interface reference).

An interpreter may return a direct identifier or another reference that needs to be interpreted.

8.6.3.11 Interpreter reference

The `interpreter-reference` field identifies an object able to replace the opaque-info part of a non-interpreted-reference with a new reference. This new reference can either be a direct reference, or a reference needing further interpretation (e.g. a translator object that is able to convert a CORBA object reference to an ANSA interface reference).

This format is provided to allow the federation of different interface reference domains.

The interpreter reference can be represented as a domain-specific shorthand, representing a full reference held by a capsule or node manager.

The interpreter reference is thus an interface reference to an interpreter interface of type Binding_Interpreter::interpreter.

The interpreter interface is further described in Annex B.

8.6.3.12 Opaque information

The format of the `opaque-info` is not standardized, as it is defined by the non-interpreted reference interpreter for the domain it controls. Objects other than the interpreter do not need to analyse this format.

8.6.4 Structuring interface types

Interface types fall in three categories: stream interfaces, operational interfaces and signal interfaces.

This Recommendation | International Standard presents the engineering interface references for the operational and stream interfaces. When either operational or stream interface structures are refined to signal interfaces, the interface references for the resulting signal interface must capture the same information as the operational or stream interface reference, although the technical representation of that information may differ.

The structural descriptions are conceptual. The suggested partitions are for further study.

8.6.4.1 Stream interfaces

Abstract description (based on the RM-ODP suggested partitioning in ITU-T Rec. X.903 | ISO/IEC 10476-3):

<stream-interf-type>	::= <stream-interf-ref-name> {<flow-description>}$^+$
<flow-description>	::= <flow-type> <flow-QoS-info>
<flow-type>	::= <flow-name><flow-behaviour>

The set of `flow-descriptions` contains information about each flow in the stream interface that reflects the computational stream interface specification. It describes the name of the flow and type of the flow, which can be for instance an audio or video protocol, and the associated QoS characteristics of the flow.

The `flow-type` information is needed to determine whether two flows can be bound.

The `flow-QoS-info` captures information about the (potential) quality of service agreement associated with a flow. For the interpretation of the information, also the information structure (type description) need to be included or referred. The flow QoS information can be expressed either as direct attribute values or QoS conformance statements.

The flow QoS information can be used for the provision of QoS contract negotiation and monitoring of flow behaviour. The use of flow QoS conformance statements may require that the channel supporting the flow contains objects for monitoring the actually reached QoS and adaptation to changed behaviour of the supporting environment.

The appropriate of flow QoS information type depend on the `inter-QoS-type` and the `flow-type`.

NOTE 1 – Both the flow-type structure and the flow-QoS-info structure are subjects of further study.

NOTE 2 – The format of the quality of service information for the flows, together with the negotiation and monitoring mechanisms involved, will be the subject of future standardization. The QoS negotiation, monitoring and binding adaptation processes have been described in QoS in ODP.

8.6.4.2 Operational interfaces

Abstract description (based on the RM-ODP suggested partitioning in ITU-T Rec. X.903 | ISO/IEC 10476-3):

<operational-interf-type>	::= <operational-interf-ref-name>{<operation-description>}+
<operation-description>	::= <operation-type><oper-QoS-info>
<operation-type>	::= <operation-name><operation-kind>
<operation-kind>	::= <announcement> | <interrogation>
<announcement>	::= <invocation-type><operation-behaviour>
<interrogation>	::= <invocation-type><operation-behaviour>{<termination-type>}+

The set of `operation-descriptions` contains information about each operation in the interface that reflects the computational operational interface specification. It describes the name of the operation and kind of the operation, which can be announcement or interrogation, and the associated operation QoS characteristics.

The `operation-type` information is needed to determine whether an offered operation can provide the requested operation. The type information can be stored to a type repository and the name information supported by an appropriate naming facility.

The `oper-QoS-info` can be used for the provision of QoS contract negotiation and monitoring activities. The operation-behaviour can include both functional and non-functional aspects, thus supporting monitoring. The QoS type information and the QoS attribute type information can be stored to a type repository. The attribute identification scheme can be supported by an appropriate naming facility.

NOTE 1 – The oper-QoS-info structure is subject of further study.

NOTE 2 – The format of the quality of service information for the operations, together with the negotiation and monitoring mechanisms involved, will be the subject of future standardization.

8.6.5 Reducing the size of the interface reference representation

If the content of an interface reference becomes too large for convenient storage or transmission, then all or part of the content can be held in a repository and replaced in the interface reference by a small key which can be used to access that content from the repository when required. The objects passing such interface references must have some other means of determining the repository at which the content is held. For example, in some systems, this might be some well-known interface.

8.7 Schemata

8.7.1 Invariant schemata

Types and templates are immutable once defined.

The interface role in a binding does not change during the lifetime of the binding.

Interface identity does not change once the interface is created.

Interface type does not change during the lifetime of the interface.

Interface reference is valid only as long as the referenced interface exists. Interface reference references only one interface during its lifetime.

8.7.2 Static schemata

Static schemata are implicit in the various object templates, i.e. the initial state of each object is determined by its template.

8.7.3 Dynamic schemata

Binding contracts can be created during negotiation and establishment of a binding liaison. The termination of a binding liaision causes destruction of the binding contract. Binding contracts can be modified as a result of a binding management activity. As a consequence of changes in binding contracts, the corresponding channel configuration can change.

Interface references are created prior to binding establishment. A binding liaison becomes invalid if a member interface or an interface reference to a member interface becomes unaccessible. However, interface references may be modified during their lifetime as result of interface relocation, or other incidents requiring liaison renegotiation or channel re-establishment.

9 Computational Viewpoint

This clause focuses on a computational viewpoint description of the engineering binding mechanism.

The objective of binding is to establish a channel between engineering objects, possibly in different clusters. The binding process can be distributed and recursive.

The binding process is obliged to select the interfaces. Therefore the binding initiator needs to identify the interface references before initiating the binding protocol. The mechanism of obtaining the references is not prescribed by this standard. Creation of interface references is supported by the node management functions prescribed in ITU-T Rec. X.903 | ISO/IEC 10746-3.

The nucleus is involved in this process, so as to make the reference unambiguous, and sufficient resources are allocated and initialized for the engineering objects in that node to participate in bindings if asked to do so. The process also involves binders, protocol objects, nucleus objects, interceptors and the relocation function. Consequently future standardization requires additional prescription of these components beyond that included in ITU-T Rec. X.903 | ISO/IEC 10746-3.

Since configuration is dynamic in ODP systems an important aspect of interface binding is fault tolerance and detection of inconsistencies including subtype mismatches.

9.1 Computational activities related to binding

Activities related to binding and interface reference management that are within the scope of this specification include:

- binding establishment, including roles and type checking, identifying of locations, construction of channels, instantiation of objects, and primitive bindings of interfaces; and
- channel establishment.

Activities related to binding and interface reference management that are not within the scope of this specification include:

- binding termination;
- channel deletion;
- creation of interface references;
- deletion of interface references;
- transfer of interface references; and
- comparison of interface references.

A channel enables interactions between engineering objects. In doing this it can also:

- provide data conversion carried by interactions over the channel;
- apply controls to, and keep records of, interactions over the channel;
- provide protocol conversion used in the interactions;
- provide measurement and control of achieved quality of service;
- enable migration and relocation of interfaces linked by the channel;
- enable failure, persistence, replication and transaction transparencies.

9.2 Binding establishment

The binding protocol provides an abstract description of the main phases in establishing a channel between interfaces. This protocol specifies a case where the interfaces to be bound are within a single domain. A case where the interfaces reside at different domains is discussed in clause 10.

The binding protocol is parametrized by factory objects that are responsible for instantiating channels of a given type. The protocol assumes the existence, for each interface in the system, of a special primitiveBind operation. A primitiveBind operation "primitively binds" adjacent interfaces, i.e. corresponds to the primitive binding action described in the RM-ODP computational model and essentially provides a (local) reference for a binding object interface.

9.2.1 Notations

We introduce the following notations in order to describe the protocol:

- C is the object that requests the creation of the binding liaison;
- A_j are interfaces to be bound using a binding object of a given type T;
- B is a channel of type T created to support the binding;
- BF is the binding factory responsible for instantiating B;
- B_j is the interface in B adjacent to A_j (i.e. that is to be locally bound to A_j).

The correspondence of these objects to those in other viewpoints are as follows:

- C is an initiator;
- binding object is related to the enterprise roles of a binding liaison, and the information viewpoint concepts of a binding type and a binding contract;
- other relationships are denoted by using the same terms.

9.2.2 Binding protocol

The protocol comprises the following main steps (see Figure 2):

1) C asks BF for the creation of a channel. In its request, C passes to BF the set of A_i interfaces to be bound. The channel structure is derived by the set of A_i references and channel template.

2) BF instantiates a channel B with a type conformant interfaces B_i in the right nodes. This includes creation or allocation of supporting object at the locations identified.

3) Once B has been created, the binding factory primitively binds all pairs (A_i, B_i) of adjacent interfaces, at the locations identified by calling the primitiveBind operation.

4) If the previous steps completed successfully, BF returns to C the references of control interfaces CI on the channel. Otherwise it returns an error termination.

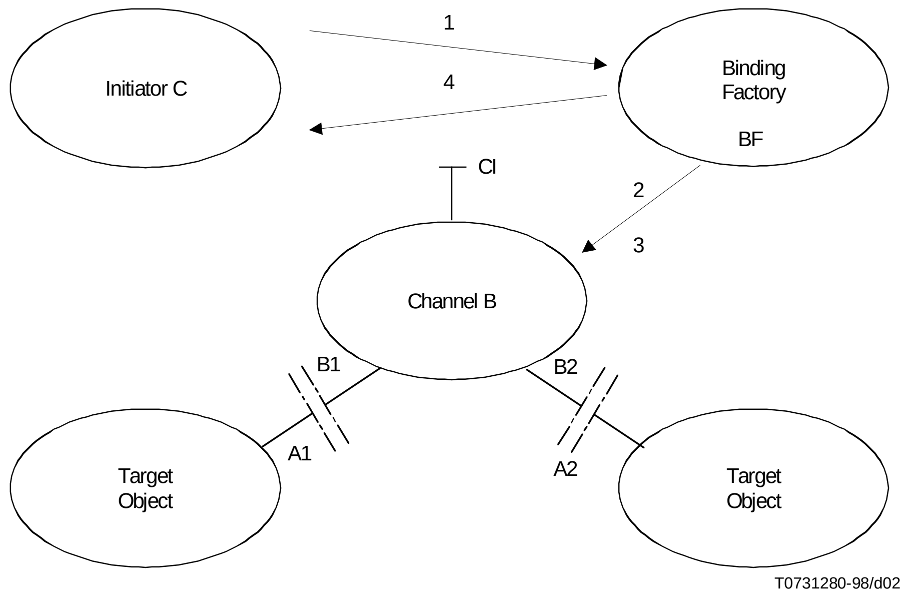


Figure 2 – Binding protocol

Step 1) invokes a "bind" operation on the factory BF. The invocation parameters depend on the binding type. Typically, they include references for the interfaces to bind and quality of service parameters for the requested binding. Invocation parameters may include other specific binding-specific parameters, for example those required to perform access control. This step can only be performed if the types of interfaces to be bound conform to the types declared for each role in the binding type definition.

Step 2) creates a distributed object. A binding factory is typically distributed over several nodes. For a given interface A_j, the fragment of the binding factory BF collocated with A_j is called the local binding factory LBF_j. The creation of B is performed in cooperation of several LBF_j, and includes:

- locating interfaces A_j;
- for each interface B_j, creating a supporting stub, and setting up appropriate communication resources (e.g. transport connections). A stub at least encapsulates the marshalling and unmarshalling of signal (operation, frames) parameters.

In this framework, the instantiation of a channel can be implemented as a distributed application, using e.g. standard operation interfaces and operation invocations. The binding protocol can thus be applied recursively during the instantiation of a channel. In particular, federated systems probably need interceptors to be included into the channel. Binding more than two targets may require renegotiation of channel templates.

Step 3) need not involve the binding factory. Instead, it can be completed later by C, or any client of the binding liaison control interfaces, provided the latter provide appropriate operations to initiate the primitiveBind calls.

NOTE 1 – At Step 2), creation of the supporting objects may involve negotiation between the different locations. At this point further bindings may (recursively) be called for.

NOTE 2 – Step 2) may involve the creation of interceptors.

9.3 Channel establishment

Channel establishment requires the creation of appropriate stubs, binders, protocol objects and interceptors. Channel establishment is parameterized by a channel template and a set of interface references each assigned to a particular role in the channel template. The channel template must be compatible with the channel types nominated by the engineering interface references for the interfaces to be bound. The nucleus for each object to be bound creates a configuration of stubs, binders and protocol objects at its node to support the interfaces of that object being bound. This includes configuration of their control interfaces. The protocol objects that support the channel are connected (possibly via interceptors) at their communication interfaces. The selection and configuration of stubs, binders, protocol objects and interceptors is determined by the channel template and channel types of the interface references involved. Each basic engineering object bound by the channel is assigned a binding endpoint identifier for each interface it has to the channel. Binding endpoint identifiers are used by basic engineering object to nominate at which of their interfaces a distributed interaction is to occur.

NOTE 1 – Any engineering object can establish a channel irrespective of whether or not the object has an interface that is to be bound by the channel.

NOTE 2 – A basic engineering object initiating a distributed binding requires a set of interface references. These may be obtained in any of the following ways:

a) on initialization of the object;

b) by interaction between the initiating object and the nucleus as part of the instantiation of the initiating object's interfaces;

c) through some chain of interactions with the other objects concerned (e.g. by parameter passing or trading).

9.4 Channel optimizations

In the general case, the binding process uses the interface references and binding type to determine the structure of the channel and then to create the channel. However, a number of optimizations can arise.

9.4.1 Pre-allocation of channel resources

In many cases, the nature of the interface permits only one possible choice of local stub, binder, and protocol object. When the interface is created, the nucleus might create these objects immediately, so that binding process can proceed more rapidly. In many systems, the local stub and binder objects are produced during the compile-link process and are always available. Similarly, many systems support only a fixed set of communications protocols and the corresponding protocol objects are always available for use and need only minor initialization/configuration during the binding process.

9.4.2 Re-binding

Since information about an interface is input to the binding process to determine the channel structure, any change to the interfaces (e.g. relocation) invalidates that channel structure, requiring the destruction of the original channel and a complete reconfiguration and re-construction of a new channel. However, in many cases, much of the configuration of the original channel (interfaces, objects, and bindings) will be identical to that required by the new channel.

Therefore, when re-binding, many systems might first attempt to repair the channel rather than the more expensive alternative of destruction and re-construction. For example, a channel with a relocated interface can be repaired by:

- removal of unwanted components (stub, binder, and protocol object at the relocated interface's previous location);
- creation of new components (stub, binder, and protocol object at the relocated interface's new location);
- replacing bindings to unwanted components by bindings to new components;
- minor reconfiguration of ongoing components.

Similar optimizations can often be performed when other aspects of a channel change, e.g. changes to the supporting objects or local bindings.

NOTE – Although channel "repairs" can be locally efficient, the overall channel structure might not be as optimal as a complete re-construction. For example, if the interface was relocated from a node which only provided an inefficient communication protocol to a node which supported an efficient communication protocol (suitable to the other interfaces in the binding), then a repair would probably continue to use the inefficient protocol whereas the re-construction would select the efficient protocol.

9.4.3 Use of recursive binding

When planning of resource allocation for a channel, a set of decisions is needed which balances efficiency with the flexibility which arises from the separation of management responsibility. The use of recursive binding allows the separation of resource management and resource policies into distinct engineering domains. However, this separation may result in a lack of global optimization which might be overcome by collecting all necessary information and centralising the allocation decisions.

For example, a binding may require that there is a limit on the delay between the endpoints. If the binding is performed recursively, arbitrary targets must be given to each segment; if all information is available at one location, a rational latency budget can be designed.

9.4.4 Elimination of unnecessary channel components

Due to the recursive nature of the binding process, it might be possible that the resulting channel structure contains redundant objects and bindings. For example, some objects might merely pass interactions between another pair of objects. Such object could be eliminated by directly binding the other pair of objects.

Similarly, an object might exist to implement a multicast interaction over several separate bindings using a simple peer-to-peer protocol. If there was a multicast protocol, then the object and the individual bindings could be replaced by a single binding using the multicast protocol.

9.5 Reducing amount of interface reference related data

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

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

9.6 Security

The security aspects of binding will be subject for further standardization in ODP security framework standard.

9.7 Failures

This subclause identifies cases where (for implementation specific reasons) the postconditions of dynamic schemata are not reached although the transition has been started from a valid precondition state.

The following failures are identified:

- failure to create a binding object because a binding liaison is not successfully established, due to an inability to match QoS constraints or to satisfy security requirements;
- failure to delete a binding object because a binding liaison is not consistently terminated in all the involved domains;
- failure to create a binding object because a channel cannot be created between objects, due to problems in resource allocation, creation mechanisms, etc.
- failures in comparison of interface references, for example, because of security reasons or failures of interpretation;
- failures to transfer interface references.

9.8 Functions

Interface references and binding activities are supported by several ODP functions:

- Node management functions support creation of interface references and resource allocation for channel instantiation, and participates in the federated binding factory activities.
- The relocation function is involved when a binding fails, when an objects migrates, or when objects are deactivated and reactivated. In such cases, the relocation function offers new interface reference information for old, and support reconfiguration of channels.
- Cluster management functions participates in the binding factory activities.

- Checkpoint and recovery functions support persistency of bindings in failure situations, in particular in recursive binding activities.
- Deactivation and reactivation function support persistency of bindings in situations where the objects are deactivate and reactivated, potentially because of relocation.
- Migration function supports re-binding after object migration.
- Engineering interface reference tracking function monitors the existence and the transfer of interface references.

The control interface of the binding liaison provides the means to manage stubs and binders in different nodes. A channel controller object can be used for dispatching of the control operations. The communication between the channel controller object and the stub and binder objects takes place through channels established for this purpose. Supporting engineering objects may be created (e.g. synchronization objects) to manage and control a set of interrelated channels.

10 Federation

The domains concerned in a federation may be administrative domains (each subject, for example, to particular security or management controls) or technology domains (each subject, for example, to common choices of system hardware or software). Federation involves specification of the objectives for interworking between different domains and of the policies governing that interworking.

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

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

Both the transfer of interface references (during computational interactions) and the binding process must take the above federation issues into account.

Autonomy of administrative domains also allows independent evolution of environment contracts at each domain, reflecting to interface references and binding contracts. Such environments require the use of special binding types that are capable of dynamically maintaining the binding liaison.

10.1 Transfer of interface references

The information about interfaces needed to allow binding may have to be passed across one or more federation boundaries on the path from the object which supports the interface to the object which initiates the binding. In such cases, federation of different domains may require translation between different interface reference formats or different naming environments.

Any necessary translation can be made when a domain boundary is crossed, so that either:

a) a valid direct reference is produced in the destination domain; or

b) a non-interpreted reference is produced, which contains a reference to an interpreter which accesses the functions associated with federation, taking some ’`opaque_info`’ and returning a more appropriate reference; the opaque information may encapsulate the `direct_reference` format of a foreign domain.

In either case, the translation may produce a result which contains all the necessary information needed to perform binding, or it may produce a result which references some piece of persistent state held by, for example, an interceptor.

Combinations of these choices lead to different policies and different associated interception strategies, such as deferred resolution, or a leave-and-forward strategy (see, for example, ANSA APM.1514.01) which associate different semantics with the components of the interface reference's structure. Some of the issues involved are introduced in the next subclause.

NOTE – The trading function (see ITU-T Rec. X.950 | ISO/IEC 13235-1) can be used for transferring interface references across federation boundaries.

10.2 Name resolution and locating the endpoints of the binding

Once the object which is to initiate a binding has collected all the necessary interface references, it submits them to the binding factory as part of the binding action. The factory then has to locate the various interfaces involved (i.e. locate the endpoints of the corresponding communication path). It may locate the interfaces as an initial phase, or it may do so in parallel with the allocation of resources for communication.

The name resolution process is compliant with that described in the ODP Naming Framework, ITU-T Rec. X.910 | ISO/IEC 14771, with:

a) the named entities being of type "ODP interface";

b) the behaviour being binding creation;

c) the information elements which are context-relative names being interface references and such of their components are either themselves interface references or name types or templates; the naming contexts at each step are associated with objects either sending or receiving interface references;

d) the action associated with name resolution prepares a communication path to be used by the binding, either by returning sufficient addressing information to create the path, or by creating and linking together segments of the path so that communication is possible immediately.

If domains limit their involvement with others to federation agreements which simply name the agreement and identify suitable interceptors, then the resolution of a non-interpreted reference involves re-tracing the path by which it was originally obtained. If the domains share knowledge of a broader interdomain topology, larger scale optimization and alternate interdomain routing become possible.

Another trade-off to be considered in deciding on a policy for the management of interface references is whether the foreign form created on entry to a domain should encapsulate the previous local form or whether an identifier for a copy held in a repository should be issued. If the encapsulation is used, the resulting reference may be bulky, but some other interpreter in another domain can interpret the resulting reference. If an identifier for it is issued, the resulting reference can be quite compact, but at the cost of requiring an access to the repository if a direct reference is to be regenerated, and an obligation on the repository to maintain the recorded state, with attendant garbage collection problems. Which option is chosen will depend on the expected traffic patterns and on security policies.

Another factor which will influence the approach chosen is the estimated likelihood of change to the federation agreements, since such changes may invalidate reference which depend on a path between domains. The complexity of recovery mechanisms in such cases will need to be considered.

10.3 Construction of the binding and resource allocation

Given the location of the set of interfaces and information about the communication paths to them, it is possible for the binding factory to determine whether the requested binding is possible and to identify the resources needed to support it. This will, in general, involve negotiation in each of the domains involved in a federation. Since this process requires local knowledge from each domain, it may be performed by subdivision into a number of localized activities.

In order to carry out the negotiations between domains, the binding factories involved must be already linked by a suitable binding. This initial binding is created as part of the agreement to federate the domains. The means to establish such an agreement are outside the scope of this Recommendation | International Standard.

The binding process can therefore involve cooperation between a number of federated binding factories to provide:

a) forwarding of the binding request to the most appropriate factory to develop a high-level communication and interception plan for the binding (a channel-structure plan, or template); this will involve the creation of a plan for the satisfaction of quality of service requirements and identification of necessary points of observation and control to manage quality of service;

b) interaction between this coordinating factory and factories in each of the other domains to determine the availability of resources in each domain to support the binding. This may involve some iteration to achieve a viable binding plan;

c) requests by the coordinating factory to factories in other domains for the creation of a number of subbindings and interceptor activations or creations to achieve the desired binding;

d) creation of control interfaces to each component involved and associated bindings to support the binding control interface whose interface reference will be returned to the binding initiator;

e) initialization of the paths involved in the binding and return of the result to the initiator.

Each of the factories involved in the creation of parts of the binding may itself construct a binding plan and delegate its execution to smaller scale factories in its subdomains. In simple cases, some of the above steps may be trivial. Since, for some network technologies, the availability of resources and expected quality may not be decidable until path creation is attempted, there may be a need for some backtracking and recovery during the binding process.

11 Compliance

This Recommendation | International Standard can be related to other, less abstract, specifications in two ways, described as follows:

a) Existing standards which define interface references or binding templates that have a compatible binding model and the properties necessary for them to take part in federations of the kind defined by this Recommendation | International Standard are said to be "consistent" with the ODP interface references and binding framework, even though they do not themselves reference this Recommendation | International Standard.

b) ODP standards which contain a reference to this Recommendation | International Standard in order to define the properties of their binding templates are said to be "compliant" with the ODP interface references and binding framework. Such standards are expected to define the relationship of their specification to the interface references and binding framework, as documented in the following clauses.

NOTE – When a consistent standard becomes supplemented with compliance statements, it is considered to become compliant as well. All compliant standards are also consistent.

Binding related activities seldom occur in isolation. Subclause 9.2 has described binding process in terms of a series of binding related actions, terminated by a channel instantiation and primitive bindings. In many practical cases, however, the binding actions also have other effects, such as allocation of resources.

A standard complying with this framework shall declare:

a) what types of interface references are defined in the standard;

b) what behaviour defined in the standard requires binding related actions;

c) what are the information elements forming the interface reference;

d) what actions in the standard are performed in the association with binding;

e) what reference points are needed to identify points of observation and control of quality of service;

f) what quality of service control methods and mechanisms are required by the standard.

Annex A

Mapping of interface reference abstract syntax to CORBA IIOP-IOR format

(This annex forms an integral part of this Recommendation | International Standard)

A representation of the abstract information language specification of an interface reference (clause 8) is needed for each engineering domain. This annex gives the representation to be used in the CORBA 2.1 engineering domain for accessing objects with ODP interface references. It can also be used as an example for other engineering domain mappings.

Interface references appear in two forms, direct interface references and non-interpreted interface references (see 8.6).

Direct interface references are represented as OMG CORBA Interoperable Object References (IOR), using both the IIOP-IOR format and a series of additional tags, specific for ITU-T | ISO, as defined in Table A.1. Within IIOP-IOR, the ITU-T | ISO-specific tags are carried in the component part of the profile body in TAG_INTERNET_IIOP profile.

Where suitable fields already exist in the IIOP-IOR format, they are used directly, with additional information included in the corresponding ITU-T | ISO-specific tag if it cannot be represented directly. In cases where an ITU-T | ISO specific tag is present, it is used after the native IIOP-IOR information.

Non-interpreted interface references are captured in a new ISO IOR profile.

NOTE – The ISO IOR profile is subject to further study. It can be based on the IIOP-IOR but redefines the sematical use of the fields.

A CORBA implementation only conforms to this specification if it preserves and forwards, as mandatory, all the tagged profiles and components defined here as representing interface references.

The choice of the format for non-interpreted references and the style of client implementation to be used (see A.3.3 and A.3.4) are determined within each interface reference management domain.

A.1 Direct interface references

Table A.1 lists the tags reserved for ITU-T | ISO use within the IIOP-IOR format, using the TAG Internet IOP profile of the OMG CORBA Interoperable Object Reference (IOR) format. Where not specified here, the format of these components are defined in the ITU-T | ISO standards which specify their use.

A.2 Non-interpreted interface references

The choice of the format for non-interpreted references and the style of client implementation to be used (see A.3.3 and A.3.4) are determined within each interface reference management domain.

The following tags are reserved for ITU-T | ISO use within the ISO IOR format when managed within OMG CORBA 2.1 engineering domain:

– TAG_ISO_REFTYPE

  See TAG_ISO_REFTYPE in A.1. The field has integer values:

  2 NON_INTERPRETED_IN_OPAQUE_INFO

– TAG_ISO_INTERPRETER_REF

  This field contains an OMG CORBA object reference.

  NOTE – In general, this field contains the interpreter interface reference in the native format for the engineering domain in question. See A.3.3 and A.3.4 for binding procedures.

– TAG_ISO_OPAQUE_INFO

  The field is a sequence of octets, whose format is defined locally to the interface reference management domain in which it is used.

Table A.1 – CORBA mapping of interface references

CORBA-IOR TAG	Interpretation of existing IOR fields	ISO interface reference fields
TAG_ISO_REFTYPE	Note 1	
TAG_ISO_INTERFACETYPE	Note 2	Interface type
TAG_ISO_CHANNEL_CLASS	Note 3	Channel class
TAG_ISO_LOCATION_INFO	Host, port, object key (Note 4)	Location information
TAG_ISO_RELOCATION_INFO		Relocation information
TAG_ISO_SECURITY_INFO	Security name, generic security mechanism (Note 5)	Security information
TAG_ISO_ADDITIONAL_INFO		Additional information
TAG_ISO_FLOW_INFO		Flow information
TAG_ISO_GROUP_INFO		Group information
TAG_ISO_CAUSALITY_INFO	Note 6	Causality information
TAG_ISO_QoS_INFO		QoS info

NOTE 1 – The way the interface reference components are interpreted depends on the interface reference type tag. The following encoding is used:

The field TAG_ISO_REFTYPE has integer values

```
0  DIRECT
1  NON_INTERPRETED_IN_OBJECT_KEY
```

If the component is not present, the reference is a direct reference.

The integer is represented as an unsigned long, encoded as a CDR encapsulation.

NOTE 2 – The type information given in the IIOP-IOR format should be used in the first instance; if more precise type information is needed, it should be included in the TAG_ISO_INTERFACETYPE. The format of this component is defined in Annex A of ITU-T Rec. X.960 | ISO/IEC 14769 (Type Repository Function).

NOTE 3 – If the channel class field is not present, the channel template used is that representing CORBA 2.0. The format of this component is defined in Annex A of ITU-T Rec. X.960 | ISO/IEC 14769 (Type Repository Function).

NOTE 4 – The host, port and object key fields are used to identify the object with which to communicate in the first instance; the way the binding process is then progressed depends on the interface type field.

NOTE 5 – The security information given in the IIOP-IOR format should be used for the first invocation, but information in the TAG_ISO_SECURITY_INFO element may be needed to continue the binding.

NOTE 6 – If the causality field is not present, the reference is to an interface with causality corresponding to a server role.

A.3 Binding procedures

NOTE – This annex covers only cases where the client ORB is unaware of the extended behaviour of ODP systems. The extended behaviour is supported either by clients, servers, and some interceptors, or by clients, server side ORBs, servers and some interceptors.

A.3.1 DIRECT

The normal CORBA binding procedure is used, without modification or extension.

NOTE – This case is limited to inter-ORB communication.

A.3.2 NON_INTERPRETED_IN_OBJECT_KEY

A.3.2.1 Using the binding

The client object is not aware of the interpretation process, and attempts to bind the object reference in the normal way. Since the client object is not aware that the referenced interfaces reside outside its local interface management domain, the procedures in A.4 and A.5 will not be applied, and so, unless an interface-type-aware interceptor is used, interworking is limited to interface types which do not involve reference passing.

A.3.2.2 Creating the binding

The client ORB uses the IOR information to bind the client object to an interceptor. The interceptor will continue the binding process, based on information in the object key. In this case, all the necessary IOR information not represented by the native CORBA 2.1 format must be included in the object key. The interceptor may redirect communication to a more specific interceptor using the IIOP location forward facility.

NOTE – The format of the object key in this case is the subject of further study.

A.3.3 NON_INTERPRETED_IN_OPAQUE_INFO with an interpreter which is within the ORB

A.3.3.1 Using the binding

In this case the client object is aware that interpretation is taking place, and will perform the procedures in A.4 and A.5 when using the resulting binding.

A.3.3.2 Creating the binding

The client object does not possess the actual interface reference for the server object, but instead is able to receive the interface reference from an interpreter via location forward facilities of the supporting ORB.

The client object does not invoke the requested operation immediately, but uses the full interface reference to send a LocateRequest message for the interpreter. The host, port and object key in the LocateRequest refer to the interpreter. The interface type refers to the target interface. The interpreter answers with LocateReply message with OBJECT_FORWARD information.

A.3.4 NON_INTERPRETED_IN_OPAQUE_INFO with an interpreter which is a CORBA object

A.3.4.1 Using the binding

In this case the client is aware that interpretation is taking place, and will perform the procedures in A.4 and A.5 when using the resulting binding.

A.3.4.2 Creating the binding

The binding process is supported by an interceptor object that supports an "interpret" operation. The interceptor is engineered as a CORBA object outside the ORBs. However, the interceptor object requires the support of the ORB in manipulating the interface references.

The client object does not invoke the requested operation immediately, but constructs an interface reference using the host, port and object key from in the interface reference to be bound, but modifying the object type to be that of the interpreter (see Annex B). This interface reference is used to invoke the "interpret" operation with the opaque_information component as parameter. The result returned is a direct interface reference which can be used to invoke the target interface (in general, this direct reference will point to a suitable interceptor, which may have been constructed specially to perform this invocation, or which may already have been available).

NOTE – The client uses the normal IIOP communication processes through the client ORB for running the extended binding process itself.

A.4 Marshalling

When a client or server is aware that the binding being used crosses a domain boundary (because the interface reference used to create the binding was a non-interpreted reference) and is about to marshal an interface reference, it should invoke the "marshal" operation of the interpreter interface used to create the binding, in order to convert any object reference to be marshalled into a form suitable for use in the destination domain.

The result returned may be either a direct or a non-interpreted reference, including an opaque_information component.

A.5 Unmarshalling

When a client or server is aware that the binding being used crosses a domain boundary (because the interface reference used to create the binding was a non-interpreted reference) and is about to unmarshal an interface reference, it should invoke the "unmarshal" operation of the interpreter interface used to create the binding, in order to convert the object reference to be unmarshalled into a form suitable for use in the local domain.

The result returned may be either a direct or a non-interpreted reference, including an opaque_information component.

Annex B

Binding interpreter interface

(This annex forms an integral part of this Recommendation | International Standard)

```
// Definition of the interface to the manager of non-interpreted-references.
// This interface is defined as an architectural reference. It may be realized in a CORBA environment (see annex A), or
// otherwise, depending on the interface management domain concerned. The way in which interface reference managers
// in different domains communicate is for bilateral agreement, and is not standardized.

module Binding_Interpreter {

        interface interpreter {
            // generic failure of interpretation process
            exception InvalidReference{};

            Object interpret (in sequence <octet> opaque_information)
                    raises (InvalidReference);

            Object marshal (in Object parameter, in Object remote_object)
                    raises(InvalidReference);

            Object unmarshal (in Object parameter, in Object remote_object)
                    raises(InvalidReference);

        };
};
```

Annex C

Bibliography

(This annex does not form an integral part of this Recommendation | International Standard)

HOFFNER (Y.) and CRAWFORD (B.), Federation and Interoperability. APM.1514.01.

Annex D

Examples

(This annex forms an integral part of this Recommendation | International Standard)

EXAMPLE 1 – STREAM INTERFACE RULES

In the simplest case, the stream channel will represent a single flow from a producer interface to a consumer interface (e.g. from an audio filing system to a speaker).

EXAMPLE 2 – STREAM INTERFACE RULES

The stream channel composition rules may be more complex.

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

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

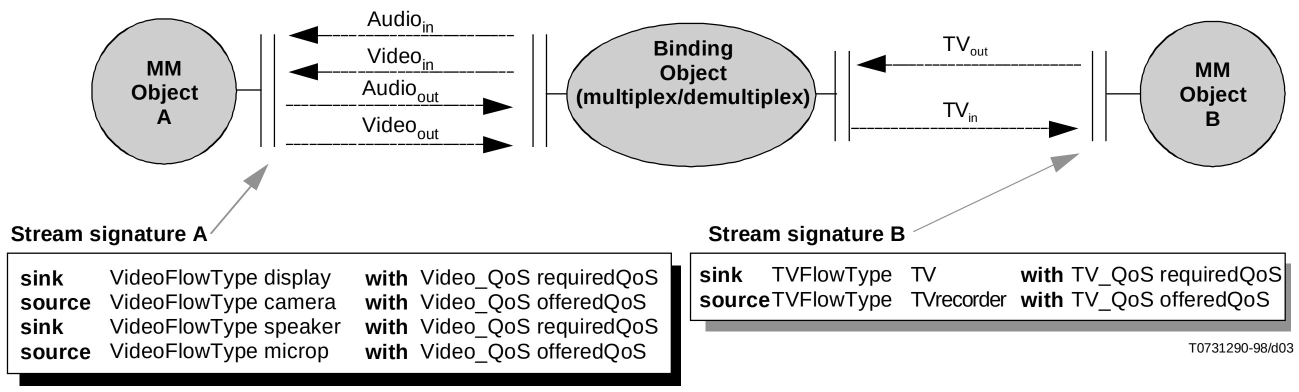
Flows from a number of producers may be combined to provide a composite flow to a single consumer. For example, a video flow from one source and an audio flow from another, might be combined into a single television flow as image and associated commentary. Here the control interface might allow manipulation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

Figure D.1 illustrates a possible binding of two multimedia objects using a channel that will multiplex the audio$_{out}$ and video$_{out}$ flows of multimedia object A onto a TV$_{in}$ flow for multimedia object B, and vice versa. The TV$_{out}$ is demultiplexed into an Audio$_{in}$ and Video$_{in}$ flow.

`TVFlowtype` is a subtype of `AudioFlowtype` and `Videoflowtype`. The following should hold:

- Signature B is a subtype of signature A and the flows that are bound have opposite causality;
- TVQoS requiredQoS < (Audio offeredQoS & Video offeredQoS) AND

  (Audio requiredQoS & Video requiredQoS) < TVQoS offeredQoS;
- CBR for both audio and video (8000 samples/s, 25fps). Similar CBR for TV flow.



Stream signature A

sink	VideoFlowType display	**with**	Video_QoS requiredQoS
source	VideoFlowType camera	**with**	Video_QoS offeredQoS
sink	VideoFlowType speaker	**with**	Video_QoS requiredQoS
source	VideoFlowType microp	**with**	Video_QoS offeredQoS

Stream signature B

sink	TVFlowType	TV	**with**	TV_QoS requiredQoS
source	TVFlowType	TVrecorder	**with**	TV_QoS offeredQoS

Figure D.1 – Example of binding stream interface

Index