INTERNATIONAL STANDARD

ISO/IEC 10746-4

First edition
1998-12-15

Information technology — Open Distributed Processing — Reference Model: Architectural semantics

Technologies de l'information — Traitement distribué ouvert — Modèle de référence: Sémantique architecturale

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 10746-4 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*,

Introduction

This Recommendation | International Standard is an integral part of the ODP Reference Model. It contains a formalisation of the ODP modeling concepts defined in ITU-T Rec. X.902 | ISO/IEC 10746-2, clauses 8 and 9. The formalisation is achieved by interpreting each concept in terms of the constructs of the different standardised formal description techniques.

This Recommendation | International Standard is accompanied by an amendment and a technical report. The associated amendment focuses on the formalisation of the computational viewpoint language contained in ITU-T Rec. X.903 | ISO/IEC 10746-3. The associated technical report contains examples on how the formalisation of the ODP Reference Model can be applied to develop specifications.

INTERNATIONAL STANDARD

ITU-T RECOMMENDATION

INFORMATION TECHNOLOGY – OPEN DISTRIBUTED PROCESSING – REFERENCE MODEL: ARCHITECTURAL SEMANTICS

1 Scope

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

The Basic Reference Model of Open Distributed Processing (RM-ODP), (see ITU-T Recs. X.901 to X.904 | ISO/IEC 10746), is based on precise concepts derived from current distributed processing developments and, as far as possible, on the use of formal description techniques for specification of the architecture.

The RM-ODP consists of:

– ITU-T Rec. X.901 | ISO/IEC 10746-1: **Overview**: Contains a motivational overview of ODP giving scooping, justification and explanation of key concepts, and an outline of ODP architecture. This part is not normative.

– ITU-T Rec. X.902 | ISO/IEC 10746-2: **Foundations**: Contains the definition of the concepts and analytical framework and notation for normalized description of (arbitrary) distributed processing systems. This is only to a level of detail sufficient to support ITU-T Rec. X.903 | ISO/IEC 10746-3 and to establish requirements for new specification techniques. This part is normative.

– ITU-T Rec. X.903 | ISO/IEC 10746-3: **Architecture**: Contains the specification of the required characteristics that qualify distributed processing as open. These are the constraints to which ODP standards must conform. It uses the descriptive techniques from ITU-T Rec. X.902 | ISO/IEC 10746-2. This part is normative.

– ITU-T Rec. X.904 | ISO/IEC 10746-4: **Architectural Semantics**: Contains a formalisation of the ODP modeling concepts defined in ITU-T Rec. X.902 | ISO/IEC 10746-2, clauses 8 and 9, and a formalisation of the viewpoint languages of ITU-T Rec. X.903 | ISO/IEC 10746-3. The formalisation is achieved by interpreting each concept in terms of the constructs of the different standardized formal description techniques. This part is normative.

The purpose of this Recommendation | International Standard is to provide an architectural semantics for ODP. This essentially takes the form of an interpretation of the basic modeling and specification concepts of ITU-T Rec. X.902 | ISO/IEC 10746-2 and viewpoint languages of ITU-T Rec. X.903 | ISO/IEC 10746-3, using the various features of different formal specification languages. An architectural semantics is developed in four different formal specification languages: LOTOS, ESTELLE, SDL and Z. The result is a formalization of ODP's architecture. Through a process of iterative development and feedback, this has improved the consistency of ITU-T Rec. X.902 | ISO/IEC 10746-2 and ITU-T Rec. X.903 | ISO/IEC 10746-3.

An architectural semantics provides the additional benefits of:

– assisting the sound and uniform development of formal descriptions of ODP systems; and

– of permitting uniform and consistent comparison of formal descriptions of the same standard in different formal specification languages.

Rather than provide a mapping from all the concepts of ITU-T Rec. X.902 | ISO/IEC 10746-2, this Recommendation | International Standard focuses on the most basic. A semantics for the higher level architectural concepts is provided indirectly through their definition in terms of the basic ODP concepts.

Examples of the use of some of the formal specification languages in this report can be found in TR 10167 (Guidelines for the Application of ESTELLE, LOTOS and SDL).

In the following clauses, the concepts are numbered in accordance with the scheme used in ITU-T Rec. X.902 | ISO/IEC 10746-2.

This Recommendation | International Standard specifies an architectural semantics for ODP. This is required to:

– provide formalisation of the ODP modelling concepts;

– assist sound and uniform development of formal descriptions of standards for distributed systems;

– act as a bridge between the ODP modelling concepts and the semantic models of the specification languages: LOTOS, SDL, ESTELLE and Z;

– provide a basis for uniform and consistent comparison between formal descriptions of the same standard in specification languages that are used to develop an architectural semantics.

This part is normative.

2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

– ISO/IEC 8807:1989, *Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour.*

– ITU-T Recommendation Z.100 (1993), *CCITT Specification and Description Language (SDL).*

– ISO/IEC TR 10167:1991, *Information technology – Open Systems Interconnection – Guidelines for the application of Estelle, LOTOS and SDL.*

– ISO/IEC 13568[1], *Information technology – Programming Languages their Environments and System Software Interfaces, Z Specification language.*

– The Z Notation, *A Reference Manual*, J.M. Spivey, *International Series in Computer Science, Second Edition, Prentice-Hall International*, 1992.

– ISO/IEC 9074:1997, *Information technology – Open Systems Interconnection – Estelle: A formal description technique based on an extended state transition model.*

3 Definitions

3.1 Definitions from ISO/IEC 8807

This Recommendation | International Standard makes use of the following terms defined in ISO/IEC 8807:

action denotation, actualisation of parameters, behaviour expression, choice, conformance, disabling, enabling, enrichment, equation, event, extension, formal gate list, formal parameter list, gate, gate hiding, guard, instantiation, interleaving, internal observable event, operation, parallel composition, parameterised type definition, process definition, reduction, selection predicate, sort, synchronisation, type definition, value parameter list.

3.2 Definitions from ITU-T Recommendation Z.100

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. Z.100:

action statement, active, atleast, block (type), call, channel, content parameter, continous signal, create, enabling condition, export, exported procedure, exported variable, finalized, gate, import, imported variable, input, nextstate, nodelay, now, output, procedure, process (type), provided, redefined, remote procedure, reset, return, revealed variable, service (type), set, signal, signalroute, stop, system (type), task, time, timer, transition, view, viewed variable, virtual.

[1] Currently at the stage of draft.

3.3 Definitions from the Z-Base Standard

This Recommendation | International Standard makes use of the following terms defined in the Z-Base Standard:

axiomatic description, conjunction, data refinement, invariant, operation refinement, overriding, postcondition, precondition, schema (operation, state, framing), schema calculus, schema composition.

3.4 Definitions from ISO/IEC 9074

This Recommendation | International Standard makes use of the following terms defined in ISO/IEC 9074:

activity, assignment statement, attach, channel, channel definition, connect, control state, DELAY-Clause, detach, disconnect, exported variable, external interaction point, FROM-Clause, function, init, instantiation, interaction, interaction point, module body definition, module header definition, module instance, output, parent instance, primitive procedure, procedure, PROVIDED-Clause, release, role, systemactivity, systemprocess, TO-Clause, transition, transition block, transition clause, WHEN-clause.

4 Interpretation of modelling concepts

4.1 Architectural semantics in LOTOS

LOTOS is a standardized (ISO/IEC 8807) Formal Specification Language (FSL). Tutorial material is available in the standard.

This clause explains how the fundamental modeling concepts can be expressed in LOTOS (see IS0/IEC 8807). It should be pointed out that there exist two main ways in LOTOS to model the concepts contained in ITU-T Rec. X.902 | ISO/IEC 10746-2. These are based upon the process algebra part of the language and the ACT ONE data typing part of the language. Since the ACT ONE formalisation of the concepts is applicable to SDL-92 also, the ACT ONE formalization is given in an independent clause. See 4.2.

To avoid confusion in the ODP and LOTOS terminology, the following clause uses *italics* to denote LOTOS specific terms.

4.1.1 Basic modeling concepts

4.1.1.1 Object

An *instantiation* of a LOTOS *process definition* which can be uniquely referenced.

4.1.1.2 Environment (of an object)

The part of a model which is not part of the object. In LOTOS, the environment of an object within a specification at a given time is given by the environment of the specification and by the other *behaviour expressions* that are composed with that object in the specification at that time.

NOTE – The environment of a specification is empty if the specification is not parameterised.

4.1.1.3 Action

Actions in LOTOS are modeled as either *internal events* or *observable events.* All events in LOTOS are atomic. An internal action may be given explicitly by the *internal event* symbol, **i**, or by an event occurrence whose associated *gate is hidden* from the environment.

An interaction is represented in LOTOS by a *synchronisation* between two or more *behaviour expressions* associated with objects at a common interaction point *(gate).* Interactions may be of the kind:

– pure *synchronisation* on a common *gate* with no offer: No passing of values between objects occurs;

– ! and ! for pure *synchronisation*: No values are exchanged between the objects;

– ! and ? for value passing provided the ? event contains the ! event: Another way of considering this is that the ! event selects a value from a choice of values for the ? event;

– ? and ? for value establishment: Here the effect is an agreement on a value from the intersection of the set of values. If the intersection of the values is the empty set then no *synchronisation* and hence no interaction occurs.

If a non-atomic granularity of actions is required event refinement may be used. This will then enable non-instantaneous and overlapping actions to be modelled. It should be noted that event refinement is a non-trivial problem, especially when behavioural compatibility is to be maintained.

There exists no construct in LOTOS to express cause and effect relationships, although this might sometimes be possible to represent informally.

4.1.1.4 Interface

An abstraction of the behaviour of an object that consists of a subset of the observable actions of that object. As all observable actions of an object in LOTOS require *gates* with which to synchronise with the environment, the subset of observable actions is usually achieved by partitioning the *gates* given in the *process definition* associated with the object. In order to obtain an interface, *hiding* the *gates* not required for the interface under consideration can be achieved. Alternatively, *synchronising* on only a subset of the *gates* associated with an object can be used. In this case, actions occurring at those *gates* in the *process definition* not in the set synchronised with, may be regarded as actions internal to the object as far as the environment synchronising on those *gates* making up the interface is concerned.

It should be noted that this definition requires that the interfaces of an object use different *gate* names, i.e. it is not possible to distinguish between interfaces that use the same *gate.*

4.1.1.5 Activity

An activity is a single-headed directed acyclic graph of actions, where each node in the graph represents a system state and each arc represents an action. For an action to occur it must satisfy the preconditions of the system state.

4.1.1.6 Behaviour (of an object)

The behaviour of an object is defined by the LOTOS *behaviour expression* associated with the *process definition* that constitutes the object template. A *behaviour expression* may consist of a sequence of both externally visible event offers and *internal events.* The actual behaviour of an object as might be recorded in a trace, is dependent upon the *behaviour expression* associated with the object and how this is configured with the environment. The actual behaviour the object exhibits depends upon the *behaviour expression* of the object and how this synchronises with its environment. An object may exhibit non-deterministic behaviour.

4.1.1.7 State (of an object)

The condition of an object that determines the set of all sequences of actions in which the object can take part. This condition is governed by the *behaviour expression* defined in the object template from which the object was created and possibly by the current bindings of any existing local variables.

4.1.1.8 Communication

The conveyance of information (via value passing) between two or more interacting objects. It is not possible to write directly, cause and effect relationships. It should also be pointed out that the *synchronisation* itself may be construed as communication.

4.1.1.9 Location in space

LOTOS abstracts away from the notion of location in space. It is only possible to equate space with the structure of the specification model. The location of an event – the structural location with respect to the specification model – is given by a *gate* for interactions in LOTOS. The notion of location in space at which an *internal event* can occur is abstracted away from in LOTOS. This abstraction is achieved implicitly using the LOTOS *hide ... in* construct which makes *gates* used internally within a process invisible to the environment of the process, or explicitly using the *internal event* symbol, **i**.

It is possible for the same location in space to be used for more than one interaction point. This is made possible in LOTOS by having a single *gate* with different *action denotations.*

The location of an object is given by the union of the locations of the *gates* associated with that object, i.e. the union of all of the locations of the actions in which that object may take part.

4.1.1.10 Location in time

LOTOS abstracts away from the concept of time, only considering temporal order so there is no *absolute location in relative metric time.* Location in time would be possible, however, if an extended form of LOTOS were used with time aspects incorporated.

4.1.1.11 Interaction point

A *gate* with a possibly empty list of associated values.

NOTE – In a specification, changes in location may be reflected by changes in the associated values.

4.1.2 Specification concepts

4.1.2.1 Composition

- **of objects:** A composite object is an object described through the application of one or more LOTOS combination operators. These include:
  - *interleaving operator* (|||);
  - *parallel composition operators* (|| and |*[gate-list]*|);
  - *enabling operator* (>>);
  - *disabling operator* ([>);
  - *choice operator* ([])
- **of behaviours:** The composition of the *behaviour expressions* associated with the component objects in the creation of a composite object through composition. The operators for the composition of behaviours are the same as those for the composition of objects.

4.1.2.2 Composite object

An object described using one or more of the *interleaving, parallel composition, disabling, enabling* and *choice operators* of LOTOS.

4.1.2.3 Decomposition

- **of objects:** The expression of a given object as a *composite object.* There may be more than one way to decompose a composite object, however.
- **of behaviours:** The expression of a given behaviour as a composite behaviour. There may be more than one way to decompose a composite behaviour.

NOTE – It might also be considered that the notion of decomposition of behaviours is inherently supplied by the ACT ONE *operations* and equations associated with a *sort.* That is, these *operations* and *equations* provide all possible combinations of behaviours. Thus for example, sequential composition might be generated through *operations* applied sequentially. Each *operation* application in the sequence must satisfy the necessary equations for occurrence. Whether this is behavioural composition is debatable though, since the *operations* and equations already existed and defined all possible behaviours.

4.1.2.4 Behavioural compatibility

In LOTOS, specific theories have been developed to check for behaviour compatibility. There are no specific LOTOS language syntactical features to construct and ensure behaviour compatibility generally. The LOTOS standard, however, develops the notion of *conformance* which provides a basis for consideration of behaviour compatibility.

In order to determine whether or not two object behaviours are compatible, the notion of *conformance* needs to be introduced. *Conformance is* concerned with assessing the functionality of an implementation against its specification, where here the term implementation may be taken to be a less abstract description of a specification.

If **P** and **Q** are two LOTOS processes, then the statement **Q** conforms to **P** (written as **Q conf P**) signifies that **Q** is a valid implementation of **P**. This means that if **P** can perform some trace σ and then behave like some process **P'**, and if **Q** can also perform trace σ and then behave like **Q'** then the following conditions on **P'** and **Q'** must be met: whenever **Q'** can refuse to perform every event from a given set A of observable actions, then **P'** must also be able to refuse to perform every event of A.

Thus **Q conf P** if and only if, placed in any environment whose traces are limited to those of **P**, **Q** cannot deadlock when **P** cannot deadlock. Another way of defining this is **Q** has the deadlocks of **P** in an environment whose traces are limited to those of **P**.

An object can be made behaviourally compatible with a second object after some modification to its behaviour, which might include *extending* the object's behaviour (adding additional behaviour) or a *reduction* of the object's behaviour (restricting the object's behaviour). This process of modification of an object is known as refinement (see 4.1.2.5).

4.1.2.5 Refinement

Refinement is the process by which an object may be modified, either by extending or reducing its behaviour or a combination of both, so that it conforms to another object. Letting **P** and **Q** be LOTOS processes, an *extension* of **P** by **Q** (written as **Q extends P**) means that **Q** has no less traces than **P**, but in an environment whose traces are limited to those of **P**, then **Q** has the same deadlocks. A *reduction* of **P** by **Q** (written as **Q reduces P**) means that **Q** has no more traces than **P**, but in an environment whose traces are limited to those of **Q**, then **P** has the same deadlocks.

4.1.2.6 Trace

A trace of the behaviour of an object from its initial instantiated state to some other state is a recording of the finite sequence of interactions *(observable events)* between the object and its environment.

4.1.2.7 Type of an <X>

Types that can be written down explicitly in LOTOS for objects and interfaces are template types. There is no explicit construct in LOTOS that will permit the modeling of action types as such. A LOTOS specification consists of a *behaviour expression* which is itself composed of *action denotations* (action templates). These action templates either occur as part of the behaviour of the system, in which case their occurrence may loosely be regarded as the action template instantiation, or they do not occur, in which case the action template remains uninstantiated. The action templates themselves may be given by the *internal event* symbol, **i**, or event offers at *gates* which may or may not have finite sequence of value and/or variable declarations.

LOTOS does not offer facilities to characterise actions directly, however, a limited form of action characterisation is built into the *synchronisation* feature of LOTOS. That is, it might be considered that synchronised *action denotations* (action templates) must satisfy the same action type in order for the action to occur. However, LOTOS does not classify the characterising features of these arbitrary *action denotations* and thus it is not possible to put a formal type to any given action. It might be the case that informally the event offers involved in an interaction are given a cause and effect role, but this is generally not the case. See 4.1.1.8.

The *internal event* symbol may be used to represent an action type, where the common characteristics of this collection of actions are that they have no characteristics.

It should be noted that by stating that the only predicate possible in LOTOS for objects (and interfaces) are that they satisfy their template type, the concepts of type and template type as given in ITU-T Rec. X.902 | ISO/IEC 10746-2 reduce to the same modeling technique in LOTOS. Thus there is no distinction in LOTOS between a type in its broad characterisation sense, and a template type in its more restrictive sense of template instantiation.

4.1.2.8 Class of an <X>

The notion of class is dependent upon the characterising type predicate which the members of the class satisfy. Objects, interfaces and actions can satisfy many arbitrary characterising type predicates. A type that can be written down is a template type. When this is the case, the class of objects, interfaces and actions associated with that type is the template class.

NOTE – It should be noted that by stating that the only classification possible in LOTOS for objects, interfaces and actions is that they satisfy their template type, the concepts of class and template class as given in ITU-T Rec. X.902 | ISO/IEC 10746-2 reduce to the same modeling technique in LOTOS. Thus there is no distinction in LOTOS between a class in its general classification sense, and a template class in its more restrictive sense as the set of instances of a given template type.

4.1.2.9 Subtype/Supertype

As the types that can be written down in LOTOS for objects, interfaces and, to a lesser extent, actions, are template types, a subtype relation in LOTOS is a relation that may exist between template types. In LOTOS, however, there exists no direct feature to write down subtyping relations directly. If subtyping is required then *extension* can be used to give a subtype relation based on substitutability, however, this is not a feature explicitly provided for in LOTOS.

4.1.2.10 Subclass/Superclass

As the types that can be written down in LOTOS for objects, interfaces and, to a lesser extent, actions, are template types, a subclass relation exists between two classes when a subtyping relation exists between their corresponding template types.

4.1.2.11 <X> Template

– **Object Template:** A *process definition* with some means by which it can be uniquely identified once instantiated. If no value parameter list is given, then object identification will not be possible for more than one object instantiated from the object template.

  With regard to combination of object templates in LOTOS there are no existing combination operators except for a limited form of scoping using the LOTOS "*where*" term.

– **Interface Template:** Any behaviour obtained from a *process definition* by considering only the interactions at a subset of the *gates* associated with the *process definition.* This subsetting of the *gates is* achieved by *hiding* the *gates* not required for the interactions under consideration.

  With regard to combination of interface templates in LOTOS there are no existing combination operators except for a limited form of scoping using the LOTOS "*where*" term.

– **Action Template:** An *action denotation* where an *action denotation* may be either an *internal-event* symbol, a gate-identifier or a gate-identifier followed by a finite sequence of value and/or variable declarations.

NOTE – The definition here of *action denotation* is contrived as LOTOS does not really support the concept of an action template. In LOTOS, possible behaviours are specified by giving *action denotations* combined in some form. To relate a template to an *action denotation* is the closest that can be achieved in LOTOS. However, the text of ITU-T Rec. X.902 | ISO/IEC 10746-2 requires an action template to group the characteristics of actions. This is not part of LOTOS as event offers *(action denotations)* exist in isolation and it is not possible to collect them and apply a template to characterise them.

Composition of action templates may loosely be likened to *synchronisation* with value passing or value generation. In this case, two (or more) action templates agree on a common action template for the *synchronisation* to occur, i.e. an action template with the common characteristics of all of the action templates involved in the *synchronisation* (composition).

4.1.2.12 Interface signature

An interface signature as a set of action templates associated with the interactions of an interface is represented in LOTOS by a set of *action denotations.* The members of this set are those *action denotations* that require *synchronisation* with the environment in order to occur.

4.1.2.13 Instantiation (of an < *X* > Template)

– **of an Object Template:** The result of a process which uses an object template to create a new object in its initial state. This process involves the *actualisation* of the *formal gate list* and *formal parameters* of a *process definition* by a one-one relabelling from a specified gate list and list of actual parameters. The features of the object created will be governed by the object template and any parameters used to instantiate it.

– **of an Interface Template:** The result of a process by which an interface is created from an interface template. The interface created can thereafter be used by the object it is associated with to interact with the environment. The features of the interface created will be determined by the interface template and any parameters used to instantiate it.

– **of an Action Template:** This is given as action occurrence in LOTOS. This may involve the rewriting of ACT ONE expressions.

4.1.2.14 Role

A name associated with a *process definition* in the template for a composite object (i.e. LOTOS composition of *behaviour expressions).* As such, roles cannot be used as parameters. However, it is possible to assign data values to each role in a composition in order to distinguish or address them specifically.

4.1.2.15 Creation (of an < *X* >)

– **of an object:** The instantiation of an object template as part of the behaviour of an existing object.

– **of an interface:** As objects and interfaces are modelled the same way in LOTOS (via *process definitions),* creation of objects corresponds to creation of interfaces. Thus the definition for interface creation is given by the creation of objects.

4.1.2.16 Introduction (of an object)

The *instantiation* of the behaviour associated with a LOTOS specification.

4.1.2.17 Deletion (of an < X >)

- **of an object:** The termination of a process *instantiation.* This may be achieved through the use of the LOTOS *disabling operator,* the LOTOS inaction *(stop) behaviour expression* which does not allow for the passing of control, or the successful termination *(exit) behaviour expression* where passing of control is possible via the *enabling operator.*
- **of an interface:** The process by which the future behaviour of an object is limited to that behaviour which did not require the participation of the given interface to be deleted.

4.1.2.18 Instance of a type

- **of an Object Template:** An instance of a given object template is represented in LOTOS by an instantiation of that object template or an acceptable substitution for an instantiation of that object template. Here the acceptable substitute should capture the characteristics that identify this type. Thus an acceptable substitute might be another template that is behaviourally compatible with the first. This might be achieved through *extension* as defined in section 4.1.2.4. Using this relation guarantees that all characteristics of the type under consideration are included. It might be the case, however, that a weaker form of type satisfaction relation can be found which does not require all characteristics associated with a given template to be included, but some subset of the total characteristics.
- **of an Interface Template:** As an interface template is represented the same way as an object template (via a *process definition* in LOTOS), the above text applies equally well (i.e. replace all occurrences of object with interface) for instance of an interface template.
- **of an Action Template:** An instance of an action template *(action denotation)* is represented in LOTOS by another *action denotation* offering an equivalent event offer.

4.1.2.19 Template type (of an < X >)

A predicate expressing that an < X > is an instance of a given template, where an < X > may be an object, an interface or an action.

4.1.2.20 Template class (of an < X >)

The template class of an < X > is the set of all < X >'s that are instances of that < X > template, where an < X > may be an object, an interface or an action.

NOTE – The notion of the template class of an action is limited in its application to LOTOS, as LOTOS does not provide explicitly for action templates, action template instantiations or action template types.

4.1.2.21 Derived class/Base class

If the template of a class is an incremental modification of the template of a second class, then the first class is a derived class of the second class, and the second class is a base class of the first.

LOTOS templates can be incrementally modified by *extending, enriching* and modifying the *data types* or by modifying the behaviour. Problems arise with the behaviour modifications however, specifically:

- subtyping: Non-determinism may be introduced into the system when the initials of the inherited template and its modification are the same, thus subtyping cannot be guaranteed;
- the need for a redirection of self-reference: Any reference to a derived template from a parent template should be redirected to the derived template, which is not always possible.

There is no satisfactory solution to these problems in standard LOTOS.

4.1.2.22 Invariant

In LOTOS, the only invariants which can be written down are *process definitions.* There is no way to attach an invariant to a *process definition* which is not the *process definition* itself.

4.1.2.23 Precondition

A precondition is a predicate that a specification requires to be true in order for an action to occur and may be expressed directly in LOTOS using one or more of:

- sequencing of actions;
- *guards* and *selection predicates.*