UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

UIT-T X.904

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

Enmienda 1
(03/2000)

SERIE X: REDES DE DATOS Y COMUNICACIÓN ENTRE SISTEMAS ABIERTOS

Procesamiento distribuido abierto

Tecnología de la información – Procesamiento distribuido abierto – Modelo de referencia: Semántica arquitectural

Enmienda 1: Formalización computacional

Recomendación UIT-T X.904 – Enmienda 1

(Anteriormente Recomendación del CCITT)

RECOMENDACIONES UIT-T DE LA SERIE X

REDES DE DATOS Y COMUNICACIÓN ENTRE SISTEMAS ABIERTOS

REDES PÚBLICAS DE DATOS	
Servicios y facilidades	X.1–X.19
Interfaces	X.20–X.49
Transmisión, señalización y conmutación	X.50–X.89
Aspectos de redes	X.90–X.149
Mantenimiento	X.150–X.179
Disposiciones administrativas	X.180–X.199
INTERCONEXIÓN DE SISTEMAS ABIERTOS	
Modelo y notación	X.200–X.209
Definiciones de los servicios	X.210–X.219
Especificaciones de los protocolos en modo conexión	X.220–X.229
Especificaciones de los protocolos en modo sin conexión	X.230–X.239
Formularios para declaraciones de conformidad de implementación de protocolo	X.240–X.259
Identificación de protocolos	X.260–X.269
Protocolos de seguridad	X.270–X.279
Objetos gestionados de capa	X.280–X.289
Pruebas de conformidad	X.290–X.299
INTERFUNCIONAMIENTO ENTRE REDES	
Generalidades	X.300–X.349
Sistemas de transmisión de datos por satélite	X.350–X.369
Redes basadas en el protocolo Internet	X.370–X.399
SISTEMAS DE TRATAMIENTO DE MENSAJES	X.400–X.499
DIRECTORIO	X.500–X.599
GESTIÓN DE REDES DE INTERCONEXIÓN DE SISTEMAS ABIERTOS Y ASPECTOS DE SISTEMAS	
Gestión de redes	X.600–X.629
Eficacia	X.630–X.639
Calidad de servicio	X.640–X.649
Denominación, direccionamiento y registro	X.650–X.679
Notación de sintaxis abstracta uno	X.680–X.699
GESTIÓN DE INTERCONEXIÓN DE SISTEMAS ABIERTOS	
Marco y arquitectura de la gestión de sistemas	X.700–X.709
Servicio y protocolo de comunicación de gestión	X.710–X.719
Estructura de la información de gestión	X.720–X.729
Funciones de gestión y funciones de arquitectura de gestión distribuida abierta	X.730–X.799
SEGURIDAD	X.800–X.849
APLICACIONES DE INTERCONEXIÓN DE SISTEMAS ABIERTOS	
Compromiso, concurrencia y recuperación	X.850–X.859
Procesamiento de transacciones	X.860–X.879
Operaciones a distancia	X.880–X.899
PROCESAMIENTO DISTRIBUIDO ABIERTO	**X.900–X.999**

Para más información, véase la Lista de Recomendaciones del UIT-T.

NORMA INTERNACIONAL ISO/CEI 10746-4
RECOMENDACIÓN UIT-T X.904

TECNOLOGÍA DE LA INFORMACIÓN – PROCESAMIENTO DISTRIBUIDO ABIERTO – MODELO DE REFERENCIA: SEMÁNTICA ARQUITECTURAL

ENMIENDA 1

Formalización computacional

Resumen

La enmienda 1 a la Rec. UIT-T X.904 | ISO/CEI 10746-4 refina y amplía la semántica arquitectural del procesamiento distribuido abierto (ODP) con una formulación del lenguaje computacional del modelo de referencia para ODP (RM-ODP). El lenguaje computacional del RM-ODP proporciona una descripción de sistemas ODP como colecciones de objetos que interactúan. Esta enmienda formaliza los conceptos y reglas del lenguaje computacional ODP utilizando diferentes técnicas de descripción formal (LOTOS, SDL, Z y Estelle).

Orígenes

La enmienda 1 a la Recomendación UIT-T X.904, preparada por la Comisión de Estudio 7 (1997-2000) del UIT-T, fue aprobada el 31 de marzo de 2000. Se publica también un texto idéntico como Norma Internacional ISO/CEI 10746-4, enmienda 1.

PREFACIO

La UIT (Unión Internacional de Telecomunicaciones) es el organismo especializado de las Naciones Unidas en el campo de las telecomunicaciones. El UIT-T (Sector de Normalización de las Telecomunicaciones de la UIT) es un órgano permanente de la UIT. Este órgano estudia los aspectos técnicos, de explotación y tarifarios y publica Recomendaciones sobre los mismos, con miras a la normalización de las telecomunicaciones en el plano mundial.

La Conferencia Mundial de Normalización de las Telecomunicaciones (CMNT), que se celebra cada cuatro años, establece los temas que han de estudiar las Comisiones de Estudio del UIT-T, que a su vez producen Recomendaciones sobre dichos temas.

La aprobación de Recomendaciones por los Miembros del UIT-T es el objeto del procedimiento establecido en la Resolución 1 de la CMNT.

En ciertos sectores de la tecnología de la información que corresponden a la esfera de competencia del UIT-T, se preparan las normas necesarias en colaboración con la ISO y la CEI.

NOTA

En esta Recomendación, la expresión "Administración" se utiliza para designar, en forma abreviada, tanto una administración de telecomunicaciones como una empresa de explotación reconocida de telecomunicaciones.

PROPIEDAD INTELECTUAL

La UIT señala a la atención la posibilidad de que la utilización o aplicación de la presente Recomendación suponga el empleo de un derecho de propiedad intelectual reivindicado. La UIT no adopta ninguna posición en cuanto a la demostración, validez o aplicabilidad de los derechos de propiedad intelectual reivindicados, ya sea por los miembros de la UIT o por terceros ajenos al proceso de elaboración de Recomendaciones.

En la fecha de aprobación de la presente Recomendación, la UIT no ha recibido notificación de propiedad intelectual, protegida por patente, que puede ser necesaria para aplicar esta Recomendación. Sin embargo, debe señalarse a los usuarios que puede que esta información no se encuentre totalmente actualizada al respecto, por lo que se les insta encarecidamente a consultar la base de datos sobre patentes de la TSB.

ÍNDICE

NORMA INTERNACIONAL

RECOMENDACIÓN UIT-T

TECNOLOGÍA DE LA INFORMACIÓN – PROCESAMIENTO DISTRIBUIDO ABIERTO – MODELO DE REFERENCIA: SEMÁNTICA ARQUITECTURAL

ENMIENDA 1

Formalización computacional

1) Prefacio

Reemplácese el primer párrafo del prefacio

La presente Recomendación | Norma Internacional forma parte integrante del modelo de referencia del procesamiento distribuido abierto (ODP). Contiene una formalización de los conceptos de modelado del ODP definidos en las cláusulas 8 y 9 de la Rec. UIT-T X.902 | ISO/CEI 10746-2. La formalización se consigue interpretando cada concepto en base a las construcciones de las diferentes técnicas de descripción formal normalizada.

por

La presente Recomendación | Norma Internacional forma parte integrante del modelo de referencia del procesamiento distribuido abierto (ODP). Contiene una formalización de los conceptos de modelado del ODP definidos en la Rec. UIT-T X.902 | ISO/CEI 10746-2, cláusulas 8 y 9 y en la Rec. UIT-T X.903 | ISO/CEI 10746-3, cláusula 7 (Lenguaje computacional). La formalización se consigue interpretando cada concepto en base a las construcciones de las diferentes técnicas de descripción formal normalizada.

2) Cláusula 0 – Introducción

Reemplácese el cuarto apartado de El modelo de referencia de procesamiento distribuido abierto está constituido por:

– La Rec. UIT-T X.904 | ISO/CEI 10746-4: **Semántica arquitectural**: Contiene una formalización de los conceptos de modelado ODP definidos en las cláusulas 8 y 9 de la Rec. UIT-T X.902 | ISO/CEI 10746-2 y una formalización de los lenguajes de punto de vista de la Rec. UIT-T X.903 | ISO/CEI 10746-3. La formalización se consigue interpretando cada concepto en términos de las construcciones de las diferentes técnicas de descripción formal normalizada. Este texto es normativo.

por

– La Rec. UIT-T X.904 | ISO/CEI 10746-4: **Semántica arquitectural**: Contiene una formalización de los conceptos de modelado ODP definidos en las cláusulas 8 y 9 de la Rec. UIT-T X.902 | ISO/CEI 10746-2 y una formalización de los lenguajes de punto de vista computacional de la Rec. UIT-T X.903 | ISO/CEI 10746-3. La formalización se consigue interpretando cada concepto en términos de las construcciones de las diferentes técnicas de descripción formal normalizada. Este texto es normativo.

Reemplácese el cuarto párrafo

La presente Recomendación | Norma Internacional tiene por objeto proporcionar una semántica arquitectural para el ODP, lo cual viene a ser en definitiva una interpretación de los conceptos básicos de modelado y de especificación de la Rec. UIT-T X.902 | ISO/CEI 10746-2 y los lenguajes de punto de vista de la Rec. UIT-T X.903 | ISO/CEI 10746-3, utilizando las diversas características de los diferentes lenguajes de especificación formal. Se elabora una semántica arquitectural en cuatro lenguajes de especificación formal diferentes: LOTOS, ESTELLE, SDL y Z. El resultado es una formalización de la arquitectura del ODP. Siguiendo un proceso de desarrollo iterativo y realimentación, se ha mejorado la coherencia de la Rec. UIT-T X.902 | ISO/CEI 10746-2 y la Rec. UIT-T X.903 | ISO/CEI 10746-3.

por

La presente Recomendación | Norma Internacional tiene por objeto proporcionar una semántica arquitectural para el ODP, lo cual viene a ser en definitiva una interpretación de los conceptos básicos de modelado y de especificación de la Rec. UIT-T X.902 | ISO/CEI 10746-2 y los lenguajes de punto de vista computacional de la Rec. UIT-T X.903 | ISO/CEI 10746-3, utilizando las diversas características de los diferentes lenguajes de especificación formal. Se elabora una semántica arquitectural en cuatro lenguajes de especificación formal diferentes: LOTOS, ESTELLE, SDL y Z. El resultado es una formalización de la arquitectura del ODP. Siguiendo un proceso de desarrollo iterativo y realimentación, se ha mejorado la coherencia de la Rec. UIT-T X.902 | ISO/CEI 10746-2 y la Rec. UIT-T X.903 | ISO/CEI 10746-3.

3) Cláusula 1 – Alcance

Añádase el siguiente párrafo al final del Alcance:

El anexo A muestra una manera que permite representar el lenguaje de punto de vista computacional de la Rec. UIT-T X.903 | ISO/CEI 10746-3 en los lenguajes formales LOTOS, SDL, Z y Estelle. En esta Recomendación | Norma Internacional se utilizan también los conceptos definidos en la Rec. UIT-T X.902 | ISO/CEI 10746-2.

4) Cláusula 2 – Referencias normativas

Cámbiese la fecha de publicación de la Recomendación UIT-T Z.100 de (1993) por (1999).

ISO/CEI 13568:

Añádase la siguiente referencia:

Z Notation, ISO/CEI JTC 1 SC 22 GT 19, Advanced Working Draft 2.C, 13 de julio de 1999.

5) Subcláusula 3.2 – Definiciones de la Recomendación UIT-T Z.100

Reemplácese la lista con los siguientes términos:

active, adding, all, alternative, and, any, as, atleast, axioms, block, call, channel, comment, connect, connection, constant, constants, create, dcl, decision, default, else, endalternative, endblock, endchannel, endconnection, enddecision, endgenerator, endnewtype, endoperator, endpackage, endprocedure, endprocess, endrefinement, endselect, endservice, endstate, endsubstructure, endsyntype, endsystem, env, error, export, exported, external, fi, finalized, for, fpar, from, gate, generator, if, import, imported, in, inherits, input, interface, join, literal, literals, map, mod, nameclass, newtype, nextstate, nodelay, noequality, none, not, now, offspring, operator, operators, or, ordering, out, output, package, parent, priority, procedure, process, provided, redefined, referenced, refinement, rem, remote, reset, return, returns, revealed, reverse, save, select, self, sender, service, set, signal, signallist, signalroute, signalset, spelling, start, state, stop, struct, substructure, synonym, syntype, system, task, then, this, timer, to, type, use, via, view, viewed, virtual, with, xor.

6) Subcláusula 3.3 – Definiciones de la norma de base Z

Cámbiese el título por:

3.3 – Definiciones de la notación Z

Reemplácese la lista con los siguientes términos:

axiomatic description, data refinement, hiding, operation refinement, overriding, schema (operation, state, framing), schema calculus, schema composition, sequence, type.

7) Anexo A

Añádase un nuevo anexo A como sigue:

Anexo A

Formalización computacional

A.1 Formalización del lenguaje de punto de vista computacional en LOTOS

A.1.1 Conceptos

Para la formalización del lenguaje computacional en LOTOS se utilizan los conceptos definidos en la formalización de las reglas básicas de modelado y estructuración indicadas en las cláusulas 8 y 9 de la Rec. UIT-T X.902 | ISO/CEI 10746-2.

Estructuras elementales asociadas con interfaces de operaciones y de señales

Para formalizar el lenguaje computacional en LOTOS es necesario introducir ciertas estructuras elementales. Éstas incluyen parámetros que podrían asociarse con ciertas interfaces computacionales, y un modelo básico de información que se podría utilizaren un flujo de trenes de datos.

Para formalizar parámetros es necesario introducir dos conceptos: nombres para cosas y tipos para cosas. Los nombres son simplemente etiquetas. Como se verá, el punto de vista computacional requiere que estas etiquetas se sometan a comprobaciones, por ejemplo para determinar la igualdad, cuando se construyan interfaces. En general, es posible representar nombres por:

```
type Name is Boolean
        sorts Name
        opns    newName: -> Name
                anotherName: Name -> Name
                _eq_,_ne_: Name, Name -> Bool
endtype (* Name *)
```

En aras de la brevedad se han omitido las ecuaciones que, como cabe esperar, son obvias. Es posible ser más prescriptivo en este lugar, por ejemplo, mediante el empleo de cadenas de caracteres de la biblioteca LOTOS. Lo único que interesa en relación con los nombres es que se pueda determinar su igualdad o desigualdad.

Como se indica en la presente Recomendación | Norma Internacional, un tipo en el sentido del ODP no puede ser interpretado directamente en la parte álgebra de proceso de LOTOS. Sin embargo, es posible modelar tipos mediante la parte Act One de LOTOS. Desafortunadamente, si bien Act One se diseñó específicamente para representar tipos, está limitada en cuanto a las formas de comprobar los tipos y las relaciones de tipos. Por ejemplo, no es posible comprobar la subtipificación o equivalencia hasta el isomorfismo entre tipos, ya que la igualdad de tipos en Act One se basa en la equivalencia de nombres de "sorts". Como una base para el razonamiento se introduce aquí una noción elemental de tipos que permite efectuar pruebas para determinar la igualdad, desigualdad y subtipificación.

```
type AnyType is  Boolean
    sorts AnyType
    opns newType: -> AnyType
        anotherType: AnyType -> AnyType
        _eq_,_isSubtype_: AnyType, AnyType -> Bool
endtype (* AnyType *)
```

Un parámetro es una relación entre un nombre y su representación de tipo subyacente. Por tanto, un parámetro se puede representar por:

```
type Param is Name, AnyType
        sorts Param
        opns newParam: Name, AnyType -> Param
            _eq_,_ne_,_isSubtype_: Param, Param -> Bool
endtype (* Param *)
```

Como se ha mencionado antes, es preciso efectuar comprobaciones para determinar la igualdad o desigualdad de parámetros, y si un parámetro es un subtipo de otro. Dos parámetros están en una relación de subtipo cuando sus tipos están en una relación de subtipo. También es útil introducir secuencias de estos parámetros:

```
type PList is String actualizedby Param
        using sortnames PList for String Param for Element Bool for FBool
        opns _isSubtype_: PList, PList -> Bool
endtype (* PList *)
```

Aquí se utiliza el tipo *String* de la biblioteca LOTOS actualizada con el tipo *Param* definido anteriormente. Se incluye también una operación *isSubtype* que puede comprobar si una secuencia de parámetros es un subtipo de otra. Una lista de parámetros es un subtipo de otra cuando todos los parámetros contenidos en la segunda lista son subtipos de los contenidos en la primera. Además, los parámetros deben ocupar la misma posición en sus listas respectivas. Cabe señalar que estos parámetros podrían contener referencias a interfaces utilizadas para restringir las interacciones que pueden producirse. Si bien es posible modelar una interfaz en el álgebra de proceso, no es posible modelar una referencia a esa interfaz en el álgebra de proceso que, hablando en términos generales, capta la funcionalidad de esa interfaz. Para resolver esta situación, se modelan las referencias de interfaz en Act One. Puesto que una referencia de interfaz capta, entre otras cosas, la firma de la interfaz, se proporciona un modelo Act One de firmas para operaciones. Las operaciones consisten en un nombre, una secuencia de entradas y posiblemente una secuencia de salidas. Para simplificar la exposición, no se considera aquí si para la operación se utiliza la notación de infijo, prefijo o sufijo. Esto se puede representar por el siguiente fragmento:

```
type Op is Name, PList
        sorts Op
        opns makeOp: Name, PList -> Op
             makeOp: Name, PList, PList -> Op
             getName: Op -> Name
             getInps: Op -> PList
             getOuts: Op -> PList
             _eq_: Op, Op -> Bool
        eqns forall op1,op2: Op, n: Name; pl1, pl2: PList
ofsort Name     getName(makeOp(n,pl1,pl2)) = n;
ofsort PList    getInps(makeOp(n,pl1)) = pl1;
                        getInps(makeOp(n,pl1,pl2)) = pl1;
                        getOuts(makeOp(n,pl1)) = <>;
                        getOuts(makeOp(n,pl1,pl2)) = pl2;
ofsort Bool     op1 eq op2 = ((getName(op1) eq getName(op2)) and
                                        (getInps(op1) isSubtype getInps(op2)) and
                                        (getOuts(op2) isSubtype getOuts(op1)));
endtype (* Op *)
```

El hecho de tener un método para determinar si dos operaciones son idénticas simplifica el problema de la subtipificación entre tipos de datos abstractos reduciéndolo a una comparación de conjuntos, donde los elementos de los conjuntos son las operaciones creadas. Así, un servidor es un subtipo de un segundo servidor si soporta todas las operaciones del segundo servidor. Se observa aquí que se modelan dos formas de operaciones: operaciones que no esperan resultados y operaciones que sí esperan resultados. Se introducen también conjuntos de esas operaciones:

```
type OpSet is Set actualizedby Op
        using sortnames OpSet for Set Op for Element Bool for FBool
endtype (* OpSet *)
```

Ahora, una interfaz puede ser representada por el siguiente fragmento LOTOS:

```
type IRef is OpSet
        sorts IRef
        opns makeIRef     :  OpSet -> IRef
             NULL         : -> IRef
             getOps       : IRef -> OpSet
             _eq_         : IRef, IRef -> Bool
        eqns forall o: OpSet; ir1, ir2: IRef
             ofsort OpSet   getOps(makeIRef(o)) = o;
             ofsort Bool    ir1 eq ir2 = getOps(ir1) eq getOps(ir2) ;
endtype (* IRef *)
```

Se observa aquí que la igualdad de las referencias de interfaz se basa solamente en las operaciones contenidas en esa referencia. Podría muy bien ampliarse para que abarcara otros aspectos, por ejemplo, la ubicación de la interfaz o constricciones impuestas a su utilización. Se presentan también conjuntos de estas referencias de interfaz:

```
type IRefSet is Set actualizedby IRef
        using sortnames IRefSet for Set IRef for Element Bool for FBool
endtype (* IRefSet *)
```

Estructuras elementales asociadas con interfaces de trenes

El punto de vista computacional de la Rec. UIT-T X.903 | ISO/CEI 10746-3 considera también interfaces que trabajan con flujos continuos de datos, por ejemplo multimedios. Estas interfaces se denominan interfaces de trenes. Las interfaces de trenes contienen conjuntos finitos de flujos. Estos flujos pueden provenir de la interfaz (flujos producidos) o estar dirigidos a la interfaz (flujos consumidos). Cada flujo se modela mediante una plantilla de acción. Cada plantilla de acción contiene el nombre del flujo, el tipo del flujo, y una indicación de causalidad relativa al flujo.

El punto de vista computacional representa una abstracción a partir del propio contenido del flujo de información. Se considera aquí una idea genérica de flujo de información en la que el flujo de información se representa por una secuencia de elementos de flujo. Un elemento de flujo puede verse como un determinado ítem en el flujo de información. Cabe observar que en el punto de vista computacional los flujos se consideran acciones continuas. En este modelo se representan los trenes como secuencias de eventos discretos temporizados. Por una parte, esto permite tratar las cuestiones de temporización de los flujos de información, pero esto se consigue a expensas de perder la naturaleza continua de los flujos.

Cada elemento de flujo en un flujo de información puede ser considerado como una unidad constituida por datos (estos datos pueden estar comprimidos) que se representa mediante *Data*. Este modelo podría incluir cómo la información, fue comprimida, qué información fue comprimida, etc. Estos aspectos no son aquí objeto de consideraciones ulteriores. Los elementos de flujo contienen también una indicación de tiempo que se utiliza para modelar el instante en que un determinado elemento de flujo fue enviado o recibido. Sucede a menudo en los flujos multimedios que determinados elementos de flujo requieren sincronización, por ejemplo la sincronización de audio con vídeo. En consecuencia, se asocia un determinado *Name* con cada elemento de flujo. Esto puede ser utilizado para seleccionar un determinado elemento del flujo, según se requiera. Por consiguiente, es posible modelar un elemento de flujo como:

```
type FlowElement is Name, NaturalNumber, Data, Param
sorts FlowElement
opns makeFlowElement: Data, Nat, Name -> FlowElement
     nullFlowElement  : -> FlowElement
     getData    : FlowElement -> Data
     getTime    : FlowElement -> Nat
     getName    : FlowElement -> Name
     toParam    : FlowElement -> Param
     setTime    : Nat, FlowElement -> FlowElement
eqns forall d: Data, s,t: Nat, n: Name
          ofsort Data       getData(makeFlowElement(d,t,n)) = d;
          ofsort Nat        getTime(makeFlowElement(d,t,n)) = t;
          ofsort Name       getName(makeFlowElement(d,t,n)) = n;
          ofsort FlowElement        setTime(s,makeFlowElement(d,t,n)) = makeFlowElement(d,s,n);
endtype (* FlowElement *)
```

Cabe señalar que el tiempo es modelado como un número natural, pero que podría darse el caso de que se utilizara tiempo real (denso), o intervalos de tiempo. Sin embargo, para simplificar, este documento se limita al tiempo discreto representado como un número natural. Se introduce también una operación que convierte un elemento de flujo en un parámetro. También para simplificar, se han omitido las ecuaciones asociadas. Se introducen también secuencias de estos elementos de flujo:

```
type FlowElementSeq is FlowElement
          sorts FlowElementSeq
          opns makeFlowElementSeq: -> FlowElementSeq
               addFlowElement: FlowElement, FlowElementSeq -> FlowElementSeq
               remFlowElement: FlowElement, FlowElementSeq -> FlowElementSeq
               getFlowElement: Name, FlowElementSeq -> FlowElement
               timeDiff: FlowElement, FlowElement -> Nat
          eqns forall f1, f2: FlowElement, fs: FlowElementSeq, n1,n2: Name
          ofsort FlowElementSeq
          getTime(f1) le getTime(f2) =>
                    addFlowElement(f1,addFlowElement(f2,makeFlowElementSeq)) =
                                                  addFlowElement(f2,makeFlowElementSeq);
          ofsort FlowElement
                    getFlowElement(n1,makeFlowElementSeq) = nullFlowElement;
                    n1 ne n2 =>
                              getFlowElement(n1,addFlowElement(makeFlowElement(d,t,n2),fs)) =
                                        getFlowElement(n1,fs);
                    n1 eq n2 =>
                              getFlowElement(n1,addFlowElement(makeFlowElement(d,t,n2),fs)) =
                                        makeFlowElement(d,t,n2);
endtype (* FlowElementSeq *)
```

De nuevo para abreviar, no se dan todas las ecuaciones. Los elementos de flujo se añaden a la secuencia si tienen indicaciones de tiempo crecientes. Se prevé una operación para atravesar una secuencia de elementos de flujo hasta encontrar un elemento de flujo provisto de un nombre. Se presenta también una operación que da la diferencia de tiempo entre las indicaciones de tiempo de dos elementos de flujo. Mediante esta operación es posible especificar, por ejemplo, que todos los elementos de flujo en una secuencia estén separados por indicaciones de tiempo iguales. En este caso se trata de un flujo isócrono. Se introducen también conjuntos de estas secuencias de elementos de flujo:

```
type FlowElementSeqSet is Set actualizedby FlowElementSeq
        using sortnames FlowElementSeqSet for Set FlowElementSeq for Element Bool for FBool
endtype (* FlowElementSeqSet *)
```

A.1.1.1 Señal

No existe una característica inherente de LOTOS que se pueda utilizar para distinguir entre una señal, un flujo de trenes y una operación. Es posible, no obstante, utilizar un estilo LOTOS para distinguir entre señales, trenes y operaciones. Por ejemplo, todas las señales podrían tener formatos similares para sus ofertas de eventos. Un ejemplo de un posible formato del lado servidor de una señal se muestra en el siguiente fragmento LOTOS:

```
<g> ?<sigName: Name> !<myRef> ?<inArgs: PList>;
```

Aquí y en el resto de A.1 se ha adoptado una notación en la que <*X*> representa un guardador de puesto para una *X*, por ejemplo *g*, *sigName*, *myRef* e *inArgs* representan guardadores de puesto para la puerta, el nombre de señal, la referencia de interfaz asociada con el servidor que ofrece esta señal y los parámetros asociados con la señal, respectivamente.

Un ejemplo de un posible formato del lado cliente se muestra en el siguiente fragmento LOTOS:

```
<g> !<sigName> !<SomeIRef> !<inArgs>;
```

Aquí el lado cliente de la señal contiene una puerta (*g*), una etiqueta para el nombre de señal (*sigName*), una referencia al objeto a que se va a enviar la señal (*SomeIRef*) y los parámetros asociados con la señal (*inArgs*). En A.1.1.11 se ve cómo estas ofertas de eventos pueden ser utilizadas para construir firmas de interfaces de señales.

A.1.1.2 Operación

Aparición de una interrogación o de un anuncio.

A.1.1.3 Anuncio

Interacción constituida por una sola invocación. Por las razones indicadas en la cláusula A.1.1.1, para modelar anuncios sólo se puede utilizar un convenio de modelado informal. Un ejemplo de esto para el lado cliente de un anuncio se podría representar mediante:

```
<g> !<invName> !<SomeIRef> !<inArgs>;
```

El lado servidor de un anuncio se podría representar mediante:

```
<g> ?<invName: Name> !<myRef>  ?<inArgs: PList>;
```

Las estructuras de datos aquí son similares a las de la cláusula A.1.1.1. En la cláusula A.1.1.12 se ve cómo estas ofertas de eventos se pueden utilizar para construir partes de firmas de interfaces de operaciones.

A.1.1.4 Interrogación

Invocación de un cliente a un servidor seguida de una de las posibles terminaciones de ese servidor o servidores a ese cliente. Sin embargo, por las razones indicadas en la cláusula A.1.1.1, para modelar interrogaciones sólo se puede utilizar un convenio de modelado informal. Un ejemplo de esto para el lado cliente de una interrogación se podría representar mediante:

```
<g> !<invName> !<SomeIRef> !<inArgs> !<outArgs>;
( <g> ?<termName:Name> !<myRef> ?<outArgs: PList>;  (* ... other behaviour *)
  [ ]  (* ... other terminations *))
```

Aquí *termName* representa los nombres de terminación y *outArgs* representa los parámetros de salida. El lado servidor de una interrogación se podría representar mediante:

```
<g> ?<invName: Name> !<myRef>  ?<inArgs: PList> ?<outArgs: PList>;
( <g> !<termName> !<SomeIRef>  !<outArgs>;  (* ... other behaviour *)
  [ ] (* ... other terminations *))
```

Las otras estructuras de datos son similares a las de la cláusula A.1.1.1. En la cláusula A.1.1.12 se ve cómo estas ofertas de eventos pueden ser utilizadas para construir partes de firmas de interfaz de operaciones.

A.1.1.5 Flujo

Abstracción de una secuencia de interacciones entre un objeto productor y un objeto consumidor, que tiene por resultado el transporte de información. Por las razones indicadas en A.1.1.1, los flujos sólo pueden representarse en LOTOS mediante convenios de modelado informales. Suele suceder que los flujos están sometidos a estrictos requisitos temporales. Un ejemplo de la forma en que esto podría obtenerse para la producción de flujos es un proceso parametrizado por una secuencia de estructuras de datos que se han de enviar, por ejemplo, elementos de flujo a los que se pueda poner un indicación de tiempo cuando se envían. Un sencillo ejemplo de la forma de modelar esto en LOTOS es:

```
process ProduceAction[ g, ...](... toSend: FlowElementSeq, tnow: Nat, rate: Nat ...):noexit:=
        g !<flowName> !<SomeIRef> !<SetTime(tnow+rate,head(toSend))>;
        (*... other behaviour and recurse with FlowElement removed from toSend *)
endproc (* ProduceAction *)
```

Aquí los elementos de flujo son enviados junto con la hora (local) actual y con la indicación de velocidad a la que deben producirse los elementos de flujo.

Por lo general, se imponen diferentes requisitos al consumo de elementos de flujos. La necesidad de supervisar continuamente las indicaciones de tiempo del flujo de información entrante es de particular importancia. Una simple representación del consumo de un flujo de información puede ser de la forma siguiente:

```
process ConsumeAction[ g,...](myRef: IRef, recFlowElements: FlowElementSeq, tnow, rate: Nat...) :noexit:=
        g ?<flowName: Name> !myRef ?<inFlowElement: FlowElement>;
        (* check temporal requirements of inFlowElement are satisfied then *)
        (* display FlowElement and recurse with time incremented *)
        (* or recurse with FlowElement added to received FlowElements and time incremented *)
endproc (* ConsumeAction *)
```

A.1.1.6 Interfaz de señales

Como LOTOS no proporciona un medio directo de distinguir formalmente entre una señal y cualquier otro evento LOTOS, determinar que una interfaz dada es una interfaz de señales sólo es posible informalmente modelando los eventos LOTOS utilizados para representar señales de una manera diferente de la empleada para cualquier otro evento. En A.1.1.11 se presenta un ejemplo de la forma en que se podría modelar en LOTOS una interfaz de señales.

A.1.1.7 Interfaz de operaciones

Como LOTOS no proporciona un medio directo de distinguir formalmente entre una operación y cualquier otro evento LOTOS, determinar que una interfaz dada es una interfaz de operaciones sólo es posible informalmente modelando los eventos LOTOS utilizados para representar operaciones de una manera diferente de la empleada para cualquier otro evento. En A.1.1.12 se presenta un ejemplo de la forma en que se podría modelar en LOTOS una interfaz de operaciones.

A.1.1.8 Interfaz de trenes

Como LOTOS no proporciona un medio directo de distinguir formalmente entre un flujo y cualquier otro evento LOTOS, determinar que una interfaz dada es una interfaz de trenes sólo es posible informalmente modelando los eventos LOTOS utilizados para representar flujos de una manera diferente de la empleada para cualquier otro evento. En A.1.1.13 se presenta un ejemplo de la forma en que se podría modelar en LOTOS una interfaz de trenes.

A.1.1.9 Plantilla de objeto computacional

En LOTOS, una plantilla de objeto computacional se representa por una definición de proceso que lleva asociado un conjunto de plantillas de interfaz computacional que el objeto puede ejemplificar, y una especificación de comportamiento, es decir, una expresión de comportamiento que no está compuesta de eventos modelados como firmas de señales, firmas de flujos, o firmas de operaciones. Debe haber también alguna forma de contrato de entorno modelado como parte de la definición de proceso; sin embargo, LOTOS no tiene todas las características necesarias para modelar completamente contratos de entorno. Puede ser posible modelar algunas características en un contrato de entorno mediante un tipo de datos Act One, que se debe dar como un parámetro formal en la lista de parámetros de valor de la definición de proceso.

A.1.1.10 Plantilla de interfaz computacional

Plantilla de interfaz de señales, plantilla de interfaz de trenes, o plantilla de interfaz de operaciones.

A.1.1.11 Firma de interfaz de señales

Una firma de interfaz de señales se representa en LOTOS por una definición de proceso en la cual todas las ofertas de eventos que, para producirse, requieran sincronización con el entorno, estén modeladas como firmas de señales. La aparición de estas ofertas de eventos tiene por consecuencia una comunicación unidireccional de un objeto iniciador a un objeto respondedor. En lo tocante a la estructura, una firma de señal es similar a una invocación de un anuncio (o a una terminación asociada con una interrogación), es decir, consta de un nombre (de la señal), una secuencia de parámetros asociados con la señal y una indicación de causalidad. Como todos los eventos en LOTOS son atómicos, no hay distinción inherente entre eventos modelados como anuncios y eventos modelados como señales.

Las firmas de interfaces de señales se diferencian de las firmas de interfaces de operaciones en que no requieren que a la interfaz, en su totalidad, se le dé una causalidad. En cambio, las firmas de interfaces de señales pueden contener señales con causalidad iniciadora o causalidad respondedora. A partir de esto, se puede modelar una firma de interfaz de señales en LOTOS mediante:

```
process SignalIntSig[ g... ](myRef: IRef, known: IRefs...):noexit:=
  g !<sigName> !<SomeIRef> !<pl>;                    ...(* other behaviour *)
    [ ]... (* other initiating actions *)
  [ ]
  g ?<sigName: Name> !myRef ?<inArgs: PList>;
            ([ not(makeOp(sigName,inArgs) IsIn getOps(myRef))] -> ...(* unsuccessful behaviour *)
            [ ]
            [ makeOp(sigName,inArgs) IsIn getOps(myRef) ] ->     ...(* successful behaviour *) )
  [ ]... (* other responding actions *)
endproc (* SignalIntSig *)
```

Aquí se indica que una interfaz de señales consta de una colección de ofertas de eventos. Estas ofertas de eventos pueden modelar señales salientes, es decir, aquellas ofertas de eventos en las que el nombre de la señal y la lista de parámetros van precedidos del prefijo **!**, o señales entrantes, es decir, aquellas ofertas de eventos en las que el nombre de la señal y la lista de parámetros van precedidos del prefijo **?** En el caso de señales entrantes, es posible comprobar que la señal entrante es una de las señales esperadas, esto es, que la señal pertenece al conjunto de señales autorizadas asociadas con esa referencia de interfaz.

NOTA – Este fragmento de especificación requiere que el proceso sea creado al menos con una puerta que corresponda al punto de interacción en el que existe la interfaz. El proceso debe también ser ejemplificado con un conjunto de referencias de interfaz y su propia referencia de interfaz. Cabe observar que no es posible escribir predicados en la señal enviada. Para ello sería necesario un nivel de prescriptividad que no tenemos, por ejemplo sería necesario asegurar que SomeIRef es una referencia de interfaz que existe en el conjunto de referencias de interfaz conocidas asociadas con el proceso. Sin embargo, es posible realizar comprobaciones en las señales que llegan; esto es, la señal que llega debe ser una de las señales asociadas con esa referencia de interfaz. Se señala también que se ha utilizado aquí el operador choice (elección) para modelar la composición de señales individuales. Es posible utilizar aquí varios otros operadores de composición, por ejemplo, el entrelazado. Si se utiliza la composición por entrelazado, es posible recibir múltiples señales que lleguen antes de que se envíe cualquier señal respondedora. Dado que las interfaces generalmente tienen alguna forma de existencia, es decir, ofrecen operaciones que pueden ser invocadas más de una vez, los comentarios que representan otros comportamientos probablemente contengan casos de procesos recursivos. Mediante el empleo del operador choice se obtiene una forma de bloqueo de señales, es decir, si llega una señal, deberá ser respondida antes de que poder aceptar cualquier otra señal. Son válidos argumentos similares para todos los demás procesos que representan firmas de interfaces computacionales.

A.1.1.12 Firma de interfaz de operaciones

Una firma de interfaz de operaciones se representa en LOTOS por una definición de proceso, de modo que todas las ofertas de eventos que requieren sincronización con el entorno para producirse, están modeladas como partes de firmas de operaciones. Es decir, todas representan partes de anuncios, o de interrogaciones. Se puede modelar una firma de interfaz de operaciones para un cliente mediante la siguiente definición de proceso:

```
process OpIntSigClient[ g... ](myRef: IRef, known: IRefs, ...):noexit:=
  g !<invName> !<SomeIRef> !<inArgs>;                ...(* other behaviour *)
    [ ]... (* other announcements *)
  [ ]
  g !<invName> !<SomeIRef> !<inArgs> !<outArgs>;     ...(* other behaviour *)
            (g ?<termName: Name> !myRef ?<outArgs: PList>;
              [ not(makeOp(termName,outArgs) IsIn getOps(myRef))] -> ...(* return error message *)
              [ ]
              [ makeOp(termName,outArgs) IsIn getOps(myRef)] ->     ...(* other behaviour *)
            [ ] ... (* other terminations *))
   [ ] ... (* other interrogations *)
endproc (* OpIntSigClient *)
```

Aquí se indica que una firma de interfaz de cliente consta de una colección de ofertas de eventos. Estas ofertas de eventos pueden modelar, o bien invocaciones (anuncio o interrogación) salientes, es decir, aquellas ofertas de eventos en las que el nombre de la invocación y la lista de parámetros van precedidos del prefijo **!**, o bien terminaciones entrantes, es decir, aquellas ofertas de eventos en las que el nombre de la terminación y la lista de parámetros van precedidos del prefijo **?** En el caso de terminaciones entrantes, es posible comprobar que la terminación entrante pertenece al conjunto de las terminaciones esperadas asociadas con esa referencia de interfaz.

La nota de A.1.1.11 es también aplicable a las firmas de interfaces de operaciones, con las modificaciones pertinentes, por ejemplo la sustitución de señal entrante por invocación.

Las firmas de interfaces de operaciones para servidores se pueden representar en LOTOS mediante:

```
process OpIntSigServer[ g... ](myRef: IRef, known: IRefs, ...):noexit:=
  g ?<invName: Name> !myRef ?<inArgs: PList>;
   ([ not(makeOp(invName,inArgs) IsIn getOps(myRef))] ->        ...(* ignore/other behaviour *)
    [ ]
    [ makeOp(invName,inArgs) IsIn getOps(myRef) ] ->           ...(* other behaviour *)
    [ ]... (* other announcements *))
 [ ]
  g ?<invName: Name> !myRef ?<inArgs:PList> ?<outArgs:PList>;   ...(* other behaviour *)
   ([ not(makeOp(invName,inArgs,outArgs) IsIn getOps(myRef))] -> ...(* return error message *)
    [ ]
    [ makeOp(invName,inArgs,outArgs) IsIn getOps(myRef) ] ->   ...(* other behaviour *)
     g !<termName> !<SomeIref> !resList ;                      ...(* other behaviour *)
     [ ] ... (* other terminations *))
  [ ] ... (* other interrogations *))
endproc (* OpIntSigServer *)
```

Al igual que las firmas de interfaces de clientes, una firma de interfaz de servidor tiene un conjunto de referencias de interfaz conocidas y una referencia para sí misma. Esta última referencia de interfaz se utiliza para asegurar que las invocaciones de anuncio o interrogación que recibe el servidor son las que se esperaban, es decir, que pertenecían al conjunto de operaciones asociadas con esa referencia de interfaz. Si estas invocaciones no fueren aceptables, por ejemplo sus parámetros no fueren correctos, o la operación solicitada no estuviere disponible, se adoptan comportamientos de tratamiento de error. En el caso de anuncios, esto podría tener por consecuencia una llamada recursiva en la cual la lista de parámetros formales se mantiene sin modificación. También es posible utilizar aquí una guarda para evitar que el evento se produzca en primer lugar. No se procede así porque ello podría producir atascos no deseados en la especificación. En el caso de interrogaciones, el resultado podría ser la devolución de alguna forma de mensaje de error.

Al igual que en el caso de las interfaces de operaciones de cliente, es posible requerir que los mensajes recibidos sean los que se esperaban. Sin embargo, no es posible imponer prescripciones sobre los mensajes que se envían. Cabría considerar que esta limitación no es necesariamente mala ya que, siempre que cada proceso trate de la misma forma los mensajes recibidos, los mensajes enviados no deben causar atascos porque, por ejemplo, no se comprenda su formato.

A.1.1.13 Firma de interfaz de trenes

Una firma de interfaz de trenes se representa en LOTOS por una definición de proceso, de modo que todas las ofertas de eventos que requieren sincronización con el entorno para producirse, están modeladas como flujos productores o como flujos de consumidores. Esto se podría representar en LOTOS como:

```
process StreamIntSig[ g... ](myRef: IRef, known: IRefSet, fss: FlowElementSeqSet...):noexit:=
            ConsumeAction[ g...](myRef, known, recFlowElements...) [ ]... (* other consume actions *)
             [ ]
             ProduceAction[ g...](myRef, known, FlowElementstoSend, ...) [ ]... (* other produce actions *)
endproc (* StreamIntSig *)
```

Al igual que en las interfaces de señales, la noción de causalidad se aplica a plantillas de acción individuales en la firma de interfaz de trenes. Una firma de interfaz de trenes contiene un conjunto de flujos que consumen o producen acciones. Cada firma de flujo se representa por un proceso. Estos procesos contienen la referencia a la interfaz de trenes con la que están asociados. un conjunto de referencias de interfaz que representan las referencias de interfaz conocidas y una secuencia de elementos de flujo que habrán de enviarse (en el caso de flujos productores) o que habrán de recibirse (en el caso de flujos consumidores). En aras de la brevedad, no se especifica cómo el conjunto de secuencias de elementos de flujo que se pasa a una firma de interfaz de trenes se asigna a los flujos productores en esa interfaz. Una vez creados, todos los flujos consumidores están, desde luego, vacíos.

La nota de A.1.1.11 se aplica también a las firmas de interfaces de trenes, con las modificaciones pertinentes, por ejemplo, la sustitución de señal que llega por flujo consumidor.

A.1.1.14 Objeto vinculador

Objeto que soporta una vinculación entre un conjunto de otros objetos computacionales. En el siguiente fragmento en LOTOS se presenta un ejemplo de cómo modelar este objeto.

```
process ServerInterface[ g ...](myRef: IRef, known: IRefSet, ...):noexit:=
    g ?bind: Name !myRef ?pl: PList;
            ([ getIRef(pl) IsIn known ] ->    ....(* already bound to server *)
              ServerInterface[ g ...](myRef,known...)
            []
            [ not(getIRef(pl) IsIn known) and not(getOps(getIRef(pl)) IsSubsetOf getOps(myRef)) ]->
                    ...(* operations requested by client not supported by server *)
            ServerInterface[ g ...](myRef,known...)
            [ not(getIRef(pl) IsIn known) and (getOps(getIRef(pl)) IsSubsetOf getOps(myRef)) ] ->
                    ...(* successful behaviour *)
            ServerInterface[ g ...](myRef,Insert(getIRef(pl),known)...)
    []
    ...(* other behaviours restricted to clients in known *)
endproc (* ServerInterface *)
```

Aquí, si el cliente ya está vinculado al servidor, se rechaza la petición de vinculación y se efectúa una llamada recursiva. Debe señalarse que esta situación no ocurre necesariamente, es decir, el mismo cliente podría estar vinculado al mismo servidor varias veces, en forma concurrente. Cada una de estas vinculaciones podría tener asociadas propiedades diferentes, por ejemplo diferentes conjuntos de operaciones solicitados con diferentes constricciones. Esto requeriría que el objeto servidor devolviera diferentes referencias de interfaz para cada vinculación exitosa. Por ejemplo, en vez de insertar la referencia de interfaz de cliente en el conjunto *conocido* en el caso de peticiones de vinculación exitosas, el servidor podría generar una referencia de interfaz que se envía al cliente y se añade al conjunto *conocido*.

Si el cliente todavía no está vinculado al servidor, es decir, no está en el conjunto de referencias de interfaz conocidas, se comprueban las operaciones asociadas con la petición del cliente. Si las operaciones solicitadas no están disponibles en el servidor, se adopta algún comportamiento de error y se efectúa una llamada recursiva. Para simplificar la exposición, se ha prescindido del tratamiento de las cuestiones relacionadas con la comprobación del tipo de los parámetros de las operaciones de cliente y de servidor. Por las mismas razones, no se han tratado los contratos de entorno. Para abreviar, no se proporcionan las operaciones Act One para acceder a las referencias de interfaz contenidas en las listas de parámetros (*getIRef*). En vez de esto, se indica simplemente que las operaciones solicitadas por los clientes deben pertenecer al conjunto de operaciones proporcionadas por el servidor.

Por último, si el cliente solicita operaciones lícitas, es decir, operaciones pertenecientes al conjunto de operaciones soportadas por la referencia de interfaz de servidor, se produce algún comportamiento satisfactorio, que podría consistir en el envío de una respuesta positiva al cliente. Después de esto, la referencia de interfaz de cliente se añade al conjunto de interfaces conocidas. La pertenencia a este conjunto permite acceder al otro comportamiento (no especificado aquí) disponible en esta interfaz. Este otro comportamiento debe realizar el conjunto de operaciones y constricciones indicado por la referencia de interfaz *myRef*.

A.1.2 Reglas de estructuración

A.1.2.1 Reglas de denominación

Las reglas de denominación contenidas en el lenguaje computacional de la Rec. UIT-T X.903 | ISO/CEI 10746-3 pueden ser soportadas en LOTOS únicamente si se siguen estrictas prácticas de modelado. Por ejemplo, cuando se crean referencias de interfaces de operaciones, por ejemplo *myRef*, se debe asegurar que el nombre de invocación y el nombre de terminación sean únicos, así como también los nombres de parámetros asociados con estas operaciones. La aplicación de estas reglas puede garantizarse mediante guardas para detectar la legalidad de las estructuras de datos recibidas (mediante paso de valores) en esa interfaz.

A.1.2.2 Reglas de interacción

A.1.2.2.1 Reglas de interacción de señales

Sucede siempre en LOTOS que un objeto que ofrece una interfaz de señales (o de trenes o de operaciones) sólo puede iniciar señales (y responder a señales) (o iniciar y responder a flujos u operaciones) que sean casos de la firma de la señal (o flujo u operación) asociada en su tipo de interfaz de señales (o de trenes o de operaciones). Esto está predefinido en las reglas de sincronización de LOTOS, esto es, sólo ofertas de eventos con indicaciones de acción concordantes (o superpuestas) pueden ser sincronizadas. Como se indica en esta Recomendación | Norma Internacional, no hay una noción real de causalidad en LOTOS, y los eventos suceden conjunta e instantáneamente o no suceden en lo absoluto. Por tanto, no se trata del caso en que se envía una invocación y luego se recibe. El envío y la recepción de una invocación se representan en LOTOS por la aparición de un solo evento.

A.1.2.2.2 Reglas de interacción de trenes

Véase A.1.2.2.1.

A.1.2.2.3 Reglas de interacción de operaciones

Véase A.1.2.2.1.

A.1.2.2.4 Reglas de parámetros

Es posible en LOTOS utilizar sorts de Act One como identificadores de interfaces computacionales. Estos identificadores pueden ser pasados entonces (mediante paso de valores) en las interacciones de una especificación dada. Después de estas interacciones, estos identificadores (sorts) pueden ser utilizados en futuras ofertas de eventos del objeto emisor y del objeto receptor, con lo que se permite que se produzcan interacciones entre el emisor y el receptor del identificador. Por tanto, los identificadores pueden considerarse como identificadores de interfaces computacionales.

Es posible tener diferentes representaciones de un identificador de interfaz computacional dado modelado en Act One. Esto se puede conseguir teniendo cierta forma de igualdad mediante una reescritura en Act One.

En LOTOS es posible asegurar que los identificadores de interfaces computacionales identifiquen la misma interfaz computacional.

A.1.2.2.5 Flujos, operaciones y señales

Si se requiere la representación de flujos y operaciones en términos de señales es necesario adoptar convenios de modelado adecuados en LOTOS, por ejemplo efectuando comprobaciones (guardas) de los nombres de las señales y las operaciones o flujos asociados.

A.1.2.3 Reglas de vinculación

A.1.2.3.1 Vinculación implícita para interfaces de operaciones de servidor

En A.1.1.14 se presenta un ejemplo del lado servidor de una vinculación implícita. Un ejemplo del lado cliente de una vinculación implícita con el propósito de invocar una operación *SomeOp* ofrecida por un servidor referenciado por *SomeIRef* y que termina cuando es completada, se podría representar en LOTOS como:

```
let myRef: IRef = makeIRef(insert(someOp,{})
in ClientInterface[g,...](myRef,Insert(SomeIRef,{},... )
        process ClientInterface[ g, ...](myRef: IRef, known: IRefSet, ...):noexit:=
            g !bind !SomeIRef !pl;                  (*  pl contains myRef, SomeIRef references server *)
            ( g ?reply: Name !myRef ?pl: PList;     (* receive successful bind response *)
              g !someOp !SomeIRef !pl2; stop)       (* invoke someOp (contained in myRef) and terminate *)
    endprocess (* ClientInterface *)
```

A.1.2.3.2 Reglas de vinculación primitiva

La vinculación primitiva se puede obtener adoptando convenios de modelado apropiados en LOTOS. Un ejemplo de esto se presenta en A.1.1.14 y A.1.2.3.1. Se señala que el lado cliente de la vinculación primitiva no debe tener que crear dinámicamente la interfaz asociada (y por consiguiente la referencia de interfaz) como se describe en A.1.2.3.1, sino que la interfaz debe existir ya.

A.1.2.3.3 Reglas de vinculación compuesta

La vinculación compuesta se puede representar en LOTOS adoptando convenios de modelado adecuados. Éstos requieren que se especifique un proceso (que represente un objeto vinculador) que acepta (mediante paso de valores) colecciones de referencias de interfaz que representan los objetos que desean ser vinculados juntos. Una vez recibidas todas las referencias, el objeto vinculador envía una petición para crear casos de todas estas interfaces referenciadas, al emisor de las respectivas peticiones de vinculación originales. Estas interfaces creadas son vinculadas seguidamente entre sí (mediante vinculación simple como se indica en A.1.1.14 y A.1.2.3.1). Una vez efectuada la vinculación, en el objeto vinculador se crean procesos que pueden ser utilizados ulteriormente para controlar las interacciones de los objetos que intervienen en la vinculación compuesta.

A.1.2.4 Reglas de tipos

A.1.2.4.1 Reglas de subtipificación para interfaces de señales

Las reglas de subtipificación para interfaces de señales se pueden obtener en LOTOS asegurando que se sigan ciertos convenios de modelado. En A.1.1.11 y A.1.1.14 se presentan ejemplos de estos convenios.

A.1.2.4.2 Reglas de subtipificación para interfaces de trenes

Las reglas de subtipificación para interfaces de trenes se pueden obtener en LOTOS asegurando que se sigan ciertos convenios de modelado. En A.1.1.13 y A.1.1.14 se presentan ejemplos de estos convenios.

A.1.2.4.3 Reglas de subtipificación para interfaces de operaciones

Las reglas de subtipificación para interfaces de operaciones se pueden obtener en LOTOS asegurando que se sigan ciertos convenios de modelado. En A.1.1.12 y 4.1.14 se presentan ejemplos de estos convenios.

A.1.2.5 Reglas de plantillas

A.1.2.5.1 Reglas de plantillas de objetos computacionales

Un objeto computacional puede:

- iniciar o responder a señales haciendo que las ofertas de eventos en su expresión de comportamiento asociada se modelen como firmas de señales con la causalidad adecuada;
- producir o consumir flujos modelando firmas de flujos en su expresión de comportamiento asociada;
- invocar o terminar operaciones haciendo que las ofertas de eventos modelen firmas de interrogaciones y de anuncios en su expresión de comportamiento asociada;
- ejemplificar plantillas de interfaces o de objetos teniendo plantillas de interfaces y de objetos como parte de su expresión de comportamiento;
- vincular interfaces teniendo una expresión de comportamiento que permita que se produzcan vinculaciones entre interfaces;
- acceder a su estado y modificarlo mediante la aparición eventos que formen parte de su expresión de comportamiento;
- detenerse (suprimirse) a sí mismo previendo una terminación LOTOS adecuada como parte de su expresión de comportamiento, por ejemplo stop, exit o [>;
- ramificar bifurcar y unir actividades utilizando combinaciones de diferentes operadores de composición LOTOS en su expresión de comportamiento asociada, por ejemplo, elección, entrelazado y composición paralela;
- vincularse a una función de comercio mediante ofertas de eventos que pueden sincronizarse con una función de comercio como parte de su expresión de comportamiento. Esto implica que la especificación LOTOS que contiene el objeto, también contiene una especificación de comerciante. Debe señalarse que el evento LOTOS que puede representar una acción de comercio puede no ser diferente de cualquier otro evento LOTOS. O sea, una acción de comercio se diferencia de cualquier otra acción, simplemente, por el hecho de que el evento tiene lugar entre el comerciante y el objeto. En consecuencia, una acción de comercio sólo podrá diferenciarse de cualquier otra acción por sus parámetros de indicación de acción en LOTOS. Por esta razón, los sorts de Act One utilizados como identificadores deben ser tratados cuidadosamente para que permitan distinguir entre las diferentes acciones.

A.1.2.5.2 Creación de plantilla de interfaz computacional

Si bien es cierto que la creación de una plantilla de interfaz computacional crea una nueva interfaz computacional en LOTOS, no lo es que se establezcan también identificadores computacionales para estas interfaces creadas. Es posible, no obstante, conseguir esto a condición de que se adopten ciertos convenios de modelado. Un ejemplo sería diseñar referencias de interfaz mediante estructuras de datos Act One que se crean cuando se crean interfaces (procesos LOTOS) asociadas.

A.1.2.5.3 Creación de plantilla de objeto computacional

La creación de una plantilla de objeto computacional en LOTOS se consigue creando la definición de proceso asociada que representa la plantilla del objeto. Las referencias a las interfaces creadas en el proceso de creación de objeto se examinan en A.1.2.5.2.

A.1.2.6 Reglas de fallos

El modelado de fallos en LOTOS se puede obtener hasta cierto punto dando todos los comportamientos posibles para un determinado sistema, es decir, los comportamientos exitosos y los comportamientos fallidos. Esto presupone, sin embargo, que todos los posibles comportamientos sean conocidos de antemano, lo que no siempre es posible porque ello

depende del tipo de fallo. Por tanto, este método de modelar fallos está limitado por el que de los fallos son previsibles en este caso, pero no en el caso general.

Los fallos de la infraestructura que se pueden producir durante la interacción (es decir, fallos relativos a la vinculación, seguridad, comunicación, recurso) pueden ser modelados hasta cierto punto en LOTOS dando todos los posibles comportamientos del sistema.

A.1.2.7 Reglas de portabilidad

LOTOS soporta todas las reglas de portabilidad de la Rec. UIT-T X.903 | ISO/CEI 10746-3, pero la comprobación del subtipo de firma y la vinculación a interfaces de comercio sólo es posible si se adoptan convenios de modelado que permitan que la firma de interfaces en Act One se represente y utilice como una base para la comprobación de tipo y ulteriores operaciones de vinculación.

Se señala que no existe una noción real de que una determinada acción en LOTOS sea una acción de bifurcación o de unión. Sólo al considerar el comportamiento de la especificación podrán identificarse las acciones de ramificación y de unión.

A.1.2.8 Puntos de conformidad y puntos de referencia

Una especificación LOTOS consiste en un sistema de comportamientos posibles. Por tanto, no es posible identificar directamente puntos de referencia que puedan pasar a ser puntos de conformidad programáticos, o perceptuales, o de intercambio, o de interfuncionamiento. Todos los puntos de referencia contenidos en una especificación están predefinidos e incumbe al implementador de la especificación identificar, etiquetar y probar los puntos de referencia. Así, un proceso de prueba que actúa en una especificación LOTOS puede estar limitado a cierta parte de la especificación, es decir, a un determinado objeto o interfaz. La identificación de este objeto o interfaz como un punto de conformidad forma parte, sin embargo, del proceso de prueba, y no del proceso de especificación.

Hay muchos posibles tipos de conformidad diferentes en LOTOS. Todos ellos relacionan el comportamiento de la especificación con alguna forma de comportamiento esperado. Así, se dice que una determinada especificación es conforme si presenta el comportamiento correcto, o sea, el comportamiento esperado, en el proceso de prueba.

A.2 Formalización del lenguaje del punto de vista computacional en SDL

Un sistema ODP (aplicaciones, funciones ODP) se describe en SDL, desde el punto de vista computacional, como una configuración de objetos computacionales. Estos objetos computacionales son casos de tipos de bloque y de tipos de proceso. Estos tipos (SDL) tienen que ser derivados de los tipos de proceso y de los tipos de bloque definidos en esta contribución. Los objetos computacionales interactúan utilizando una infraestructura que ofrece las funciones ODP. El modelo computacional de la infraestructura es modelado por un bloque SDL. Esto hace que todas las funciones ODP estén visibles en el punto de vista computacional como procedimientos distantes SDL.

Esta subcláusula muestra, utilizando la técnica de descripción formal SDL, cómo pueden expresarse los conceptos y reglas del lenguaje computacional. Para conceptos que no puedan ser totalmente abarcados se propone la utilización de texto informal. Se utiliza la escritura en *cursiva* para distinguir entre términos ODP y SDL.

Un sistema ODP se describe en SDL, desde el punto de vista computacional, como una configuración de objetos computacionales. Estos objetos computacionales son casos de tipos de bloque y de tipos de proceso.

A.2.1 Conceptos

Los conceptos del lenguaje computacional se expresan en SDL-92 de acuerdo con la Recomendación UIT-T Z.100 y la Recomendación UIT-T Z.105, y de conformidad con las definiciones, reglas y directrices genéricas de la presente Recomendación | Norma Internacional.

A.2.1.1 Señal

Una señal puede ser representada por la aparición de una acción *OUTPUT* <*sdl-signal*> y la recepción de esa <*sdl-signal*> (o una <*sdl-signal*> conexa que contiene la información de la <*sdl-signal*> original) por el *inputport* del *PROCESS* receptor.

NOTA – Una señal es una acción atómica. Aunque se modela en SDL por una serie de acciones (SDL), la atomicidad queda garantizada porque la transmisión por un canal y la recepción a través del puerto de entrada de los receptores están implícitas. Una salida puede ser instantánea cuando el iniciador y el respondedor están conectados por canales sin retardo o por rutas de señales.

A.2.1.2 Operación

Una operación es la aparición de una interrogación o de un anuncio.

A.2.1.3 Anuncio

Un anuncio es una secuencia de acciones, modelada por la aparición de una de las siguientes acciones:

- *OUTPUT* de una <*sdl-signal*> por un *PROCESS* interfaz de un objeto cliente;
- INPUT de esa <*sdl-signal*> o de una <*sdl-signal*> conexa que contiene la información de la <*sdl-signal*> original, por un *PROCESS* interfaz del objeto servidor (**Invocación**).

NOTA – El comienzo de la función que habrá de ser realizada por el servidor se modela mediante la transición provocada por *INPUT*.

La estructura de un anuncio se indica en el cuadro A.1.

Cuadro A.1 – Anuncio (lado cliente y lado servidor)

```
PROCESS TYPE Client
INHERITS OperationInterface;
...
OUTPUT servicel(p1,p2,p3) TO Serverl

...
ENPROCESS TYPE Client
```

```
PROCESS TYPE Server
INHERITS OperationInterface;
ADDING GATE InterfaceSignature
        ADDING IN WITH servicel;

...
STATE bound;
        INPUT service1(a1,a2,a3);
        TASK 'perform required function';
        NEXTSTATE-;

...
ENDPROCESS TYPE Server
```

A.2.1.4 Interrogación

Interacción entre un objeto cliente y un objeto servidor que consiste en:

- *OUTPUT* de una <*sdl-signal*> por un *PROCESS* interfaz de un objeto cliente;
- *INPUT* de esa <*sdl-signal*> (o una <*sdl-signal*> conexa que contiene la información de la <*sdl-signal*>) original por una interfaz *PROCESS* del objeto servidor (**Invocación**) seguida de;
- una posible ejecución de la función solicitada;
- *OUTPUT* de una <*sdl-signal*> por un *PROCESS* interfaz del objeto servidor; e
- *INPUT* de esa <*sdl-signal*> (o de una <*sdl-signal*> conexa que contiene la información de la <*sdl-signal*>) original por un *PROCESS* interfaz del objeto (**Terminación**).

La estructura de una interrogación se indica en el cuadro A.2.

Cuadro A.2 – Interrogación mediante el empleo de señales SDL

```
PROCESS TYPE Client
INHERITS OperationInterface;
...
OUTPUT servicel(p1,p2,p3) TO Serverl;
NEXTSTATE waitService1;
STATE waitService1
        INPUT service1Term1;
                ...
        INPUT service1Term2;

                ...
        SAVE*;
...
ENPROCESS TYPE Client
```

```
PROCESS TYPE Server
INHERITS OperationInterface;
        ADDING GATE InterfaceSignature
        ADDING IN WITH servicel;
...
STATE bound;
        INPUT service1(a1,a2,a3);
        TASK 'perform required function';
        OUTPUT service1term1 TO SENDER;
                NEXTSTATE-;

...
ENDPROCESS TYPE Server
```

Las interrogaciones síncronas pueden ser modeladas por *REMOTE PROCEDURES* que retornan valores. El cliente llama un *REMOTE PROCEDURE* del *PROCESS* interfaz del objeto servidor. El modelo de sustitución SDL para *REMOTE PROCEDURES* asegura el cumplimiento de los requisitos de secuencialidad de la interrogación.

La estructura de una interrogación síncrona se indica en el cuadro A.3.

Cuadro A.3 – Interrogación mediante el empleo de procedimientos distantes SDL

```
PROCESS TYPE Client
INHERITS OperationInterface;
IMPORTED PROCEDURE service1;
...
CALL servicel(par1,par2,par3) TO Serverl;
...
ENPROCESS TYPE Client
```

```
PROCESS TYPE Server
INHERITS OperationInterface;
ADDING
EXPORTED PROCEDURE service1 REFERENCED;
...
STATE bound;
        INPUT PROCEDURE service1(p1,p2,p3);
                NEXTSTATE-;

...
ENDPROCESS TYPE Server
```

A.2.1.5 Flujo

Un flujo es una abstracción de un conjunto de interacciones. Se modela en SDL en el lado productor por una *señal continua* como se indica en el cuadro A.4.

Cuadro A.4 – Flujo (basado en señales)

```
/* Producer*/

...
PROVIDED Available(Data);
        OUTPUT Frame(GetFrame(Data)) TO BoundTo;
        NEXTSTATE-;
...
```

```
/* Consumer*/

...
STATE Receive;
        PRIORITY INPUT Frame(Data);
                ...;
        NEXTSTATE-;
        INPUT NONE;

                ...;
        NEXTSTATE-;
```

Una ulterior abstracción puede basarse en el concepto de valores *IMPORTED* y *EXPORTED*.

A.2.1.6 Interfaz de señales

Un ejemplar de *PROCESS* que comunica solamente con *SIGNALS* (de SDL) a través de *CHANNELS* sin retardo. El *PROCESS* es una ejemplificación de un subtipo de una plantilla SignalInterfaceTemplate.

A.2.1.7 Interfaz de operaciones

Interfaz en la que todas las interacciones son operaciones. El *PROCESS* es una ejemplificación de un subtipo de una plantilla OperationalInterfaceTemplate.

A.2.1.8 Interfaz de trenes

Ejemplar de *PROCESS* que comunica solamente con *SIGNALS* (SDL) a través de canales *CHANNELS* o a través de variables *EXPORTED* o *IMPORTED*. El *PROCESS* es una ejemplificación de un subtipo de una plantilla StreamInterfaceTemplate.

A.2.1.9 Plantilla de objeto computacional

Plantilla de objeto para un objeto computacional. Se representa en SDL como una definición *BLOCK* basada en tipo, donde el *BLOCK TYPE* es una especialización del *BLOCK TYPE* ComputationalObjectTemplate. El concepto de contrato de entorno no está soportado en SDL; por tanto, hay que utilizar texto informal.

Todas las puertas de entrada y de salida de una ComputationalObjectTemplate tienen que estar conectadas a subtipos de los procesos InterfaceTemplate.

La plantilla de un objeto computacional *BLOCK TYPE* se indica en el cuadro A.5.

Cuadro A.5 – BLOCK TYPE ComputationalObjectTemplate

```
BLOCK TYPE ComputationalObjectTemplate;
VIRTUAL PROCESS TYPE SignalInterfaceTemplate REFERENCED;
VIRTUAL PROCESS TYPE StreamInterfaceTemplate REFERENCED;
VIRTUAL PROCESS TYPE OperationInterfaceTemplate REFERENCED;
VIRTUAL PROCESS TYPE BehaviourTemplate REFERENCED;
PROCESS LocalBehaviour(1,) : BehaviourTemplate;
/*
        GATE Definitions
        SIGNALROUTE Definitions
        PROCESS Definitions
        have to be added in specializations of this TYPE
/*
ENDBLOCK TYPE;
```

A.2.1.10 Plantilla de interfaz computacional

Plantilla de interfaz de señales, plantilla de interfaz de trenes o plantilla de interfaz de operaciones. El concepto de contrato de entorno no está soportado en SDL; por tanto, hay que utilizar texto informal.

Una SignalInterfaceTemplate se representa por una definición de *PROCESS* basada en tipo, donde el *PROCESS TYPE* es al menos una especialización de la plantilla de interfaz de señales *PROCESS TYPE*, como se indica en el cuadro A.6. El *PROCESS TYPE* tiene que tener una sola *GATE* conectada al exterior del *BLOCK* circundante. Esta *GATE* representa la firma de interfaz de señales. Todas las comunicaciones con el exterior del *BLOCK* tienen que basarse en el intercambio de *SIGNAL* a través de esta *GATE*. No puede existir ningún *OUTPUT* al exterior del *BLOCK* circundante en el *STATE* desvinculado. El comportamiento se especifica mediante el gráfico de proceso del cuadro A.6.

Cuadro A.6 – PROCESS TYPE SignalInterfaceTemplate

```
PROCESS TYPE SignalInterfaceTemplate;
GATE InterfaceSignature IN FROM ATLEAST SignalInterfaceTemplate
        OUT TO ATLEAST SignalInterfaceTemplate;
DCL BoundTo PId;
                START VIRTUAL; NEXTSTATE unbound;
                STATE unbound;  INPUT VIRTUAL Bind(BoundTo); NEXTSTATE bound;
                                INPUT VIRTUAL *;   NEXTSTATE -;
                STATE bound;    INPUT VIRTUAL UnBind;   NEXTSTATE unbound;
                                INPUT VIRTUAL *;  NEXTSTATE -;
                STATE*;         INPUT VIRTUAL Delete;  STOP;
ENDPROCESS TYPE;
```

Una plantilla de interfaz de trenes se representa por una definición de *PROCESS* basada en tipo, en la que el *PROCESS TYPE* es al menos una especialización del *PROCESS TYPE* StreamInterfaceTemplate, como se indica en el cuadro A.7. El *PROCESS TYPE* tiene que tener una sola *GATE* conectada al exterior del *BLOCK* circundante. Esta *GATE* representa la firma de interfaz de trenes. Todas las comunicaciones al exterior del BLOCK tienen que basarse en el intercambio de *SIGNAL* a través de esta *GATE* por *señales continuas* (flujos) en el *STATE* vinculado.

El comportamiento se especifica mediante el gráfico de proceso del cuadro A.7.

Cuadro A.7 – PROCESS TYPE StreamInterfaceTemplate

```
PROCESS TYPE StreamInterfaceTemplate;
        INHERITS SignalInterfaceTemplate
        GATE InterfaceSignature ADDING IN FROM ATLEAST BinderStreamInterfaceTemplate
                OUT TO ATLEAST BinderStreamInterfaceTemplate;
ENDPROCESS TYPE;
```

Una plantilla de interfaz de operaciones se representa mediante una definición de *PROCESS* basada en tipo, en la que el *PROCESS TYPE* es al menos una especialización del *PROCESS TYPE* OperationInterfaceTemplate, como se indica en el cuadro A.8. El *PROCESS TYPE* tiene que tener una sola *GATE* conectada al exterior del *BLOCK* circundante. Esta *GATE* representa la firma de interfaz de operaciones. Todas las comunicaciones al exterior del BLOCK tienen que basarse en el intercambio de *SIGNAL* a través de esta *GATE* por *REMOTE PROCEDURES*. El comportamiento se especifica por el cuerpo de *PROCESS* en el cuadro A.8.

Cuadro A.8 – PROCESS TYPE OperationInterfaceTemplate

```
PROCESS TYPE OperationInterfaceTemplate;
        INHERITS SignalInterfaceTemplate
        GATE InterfaceSignature ADDING IN FROM ATLEAST OperationInterfaceTemplate
                OUT TO ATLEAST OperationInterfaceTemplate;
ENDPROCESS TYPE;
```

A.2.1.11 Firma de interfaz de señales

La firma de una interfaz de señales es dada por una definición *GATE* del correspondiente *PROCESS*. Contiene el conjunto de nombre de todas las *SIGNALS* enviadas/recibidas por el *PROCESS* interfaz y da una indicación de su causalidad (*IN WITH* o *OUT WITH* respectivamente). Una firma de señal es dada por una definición de *SIGNAL*, que comprende:

- un nombre de la señal;
- el número y los tipos de los parámetros de la señal.

SDL no permite la denominación de parámetros, pero los parámetros se identifican por su posición.

Además, el *SIGNALSET* del *PROCESS* contiene el conjunto de nombres de todas las *SIGNALS* que el *PROCESS* puede recibir.

A.2.1.12 Firma de interfaz de operaciones

Una firma de interfaz para una interfaz de operaciones comprende la firma de cada operación en la interfaz. Se da por definiciones *GATE* de acuerdo con A.2.1.10 o por definiciones *REMOTE* e *IMPORTED* o *EXPORTED PROCEDURE* (en caso de *REMOTE PROCEDURES*).

La firma de anuncio, en caso de REMOTE PROCEDURE, es reflejada por la firma de *PROCEDURE*, y en otro caso viene dada por una definición de *SIGNAL*.

La firma de terminación, en caso de *REMOTE PROCEDURES*, es reflejada por la firma del tipo de datos del valor devuelto por *PROCEDURE*, y en otro caso viene dada por una definición de *SIGNAL*.

A.2.1.13 Firma de interfaz de trenes

La firma de una interfaz de trenes es dada por una definición de *GATE*, que contiene el conjunto de nombres de todas las *SIGNALS* enviadas/recibidas por el *PROCESS* interfaz, es decir, los flujos, y una indicación de causalidad como se indica en A.2.1.10.

Además, el *SIGNALSET* del *PROCESS* contiene el conjunto de todas las *SIGNALS* que el *PROCESS* puede recibir.

En el caso de procedimientos distantes, éstos tienen que especificarse en el *PROCESS* interfaz como *EXPORTED* o *IMPORTED*. La cláusula *REMOTE* restringe la visibilidad de *EXPORTED PROCEDURES*.

NOTA – Las firmas de interfaz complementarias se indican por definiciones de *GATE* con listas de señales idénticas pero cláusulas de sentido de transmisión complementarias (*IN WITH* y *OUT WITH* respectivamente).

A.2.1.14 Objeto vinculador

Caso de un *CHANNEL*. La estructura interna de una objeto vinculador se especifica mediante una *CHANNEL SUBSTRUCTURE*.

Los objetos vinculadores más complejos se deben representar mediante una configuración de dos o más *CHANNELS* y uno o más casos de *BLOCK*.

NOTA – Algunos *CHANNELS* en SDL se dan implícitamente, por ejemplo *CHANNELS* que soportan la comunicación mediante *EXPORTED PROCEDURES* y valores *EXPORTED*. Una configuración constituida por un *BLOCK* y dos o más *CHANNELS* permite efectuar vinculaciones entre más de dos interfaces, así como vinculaciones con una o más interfaces de control.

A.2.2 Reglas de estructuración

Una especificación computacional en SDL describe la descomposición funcional de un sistema ODP en términos transparentes a la distribución, como:

- una configuración de *BLOCK*S y *CHANNELS*;
- las acciones internas son modeladas por *PROCESSES* locales que no comunican al exterior del *BLOCK*;
- las interacciones son modeladas por *PROCESSES* interfaz.

Una especificación computacional en SDL está constreñida por las reglas del lenguaje computacional y por la semántica de SDL.

El conjunto inicial de objetos computacionales viene dado por un conjunto de *BLOCK*S y los *PROCESSES* en ellos contenidos. Se puede especificar un número inicial para cada diferente modalidad de *PROCESS*. Los cambios de la configuración se expresan en la especificación de comportamiento.

A.2.2.1 Reglas de denominación

La semántica estática del SDL asegura la aplicación de las reglas siguientes:

- los nombres de señales son distintos en cualquier firma de interfaz de señales, porque una *SIGNAL* sólo puede aparecer una vez en una definición de *GATE*;
- los nombres de operaciones son distintos en cualquier firma de interfaz de operaciones, porque un *PROCESS* no puede exportar ni importar dos *PROCEDURES* diferentes con el mismo nombre;
- los parámetros se identifican inequívocamente por su posición. No hay denominación de parámetros de *SIGNAL*;
- los identificadores de interfaz computacional corresponden con *PIds* (identificadores de *PROCESS*) por lo que son únicos en toda la especificación.

A.2.2.2 Reglas de interacción

Para cumplir la constricción de que las interacciones en una interfaz no vinculada causen un fallo de la infraestructura, todas las acciones *OUTPUT* tienen que ser calificadas por cláusulas *VIA* y/o *TO*. Todas las *SIGNALS* enviadas fuera de un objeto computacional *BLOCK* tienen que ser enviadas por un *PROCESS* interfaz y encaminadas a través de una *GATE* de ese *PROCESS*.

Todas las *llamadas a procedimiento distante* tienen que se calificadas por TO. Todos los fallos tienen que ser especificados explícitamente, pues SDL no proporciona un mecanismo para tratar los fallos. Después de producirse un error, el comportamiento es indeterminado.

A.2.2.2.1 Reglas de interacción de señales

El cumplimiento de las reglas de interacción de señales está garantizado por la semántica de SDL. Un *PROCESS* interfaz de señales sólo puede enviar/recibir aquellas *SIGNALS* que hayan sido especificadas para ese *PROCESS*.

A.2.2.2.2 Reglas de interacción de trenes

Las interfaces de trenes se basan en el intercambio de SIGNAL, por lo que el cumplimiento de las reglas de interacción de trenes está garantizado por la semántica de SDL. Un *PROCESS* interfaz de trenes sólo puede producir/consumir aquellos flujos que hayan sido especificados para ese *PROCESS*.

A.2.2.2.3 Reglas de interacción de operaciones

En el caso de PROCEDURES distantes, el cumplimiento de las reglas de interacción de operaciones está garantizado por la semántica de SDL. En el caso de SIGNALS, el especificador tiene que aplicar las siguientes reglas:

- para todas las *SIGNALS* que modelan las invocaciones de operaciones tiene que haber, en todos los estados del *PROCESS*, transiciones para tratar estas *SIGNALS* (¡no puede haber descarte implícito!);
- la transición o secuencia de transiciones provocadas por la invocación tienen que terminar con exactamente una *OUTPUT* <*sdl-signal*>, donde <*sdl-signal*> es una *SIGNAL* que modela una de las terminaciones.

El orden de aparición de las señales de entrega de invocación/terminación de invocaciones/terminaciones concurrentes no tiene necesariamente que ser el mismo que el orden de aparición de las correspondientes señales de depósito de invocación/terminación. En SDL, las *SIGNALS* concurrentes aparecen en un orden arbitrario (no-determinístico). No hay un medio que permita describir directamente la duración de una operación; la duración es arbitraria.

A.2.2.2.4 Reglas de parámetros

Los identificadores de interfaces computacionales se representan como *PIds*. Pueden ser parámetros de argumento y parámetros de resultado, en interacciones tanto de señales como de operaciones. Los identificadores pueden pasarse como parámetros en ulteriores interacciones.

El receptor de un identificador de interfaz computacional puede utilizar el identificador para intervenir en interacciones con el objeto que soporta la interfaz, a condición de que se pueda establecer una vinculación entre las interfaces.

Los identificadores computacionales son unívocos dentro de la especificación *SYSTEM*. Se pueden utilizar *SYNONYMS* para dar más de un identificador a una interfaz.

Siempre es posible determinar si dos identificadores de interfaz computacional identifican la misma interfaz computacional; sin embargo, no hay un medio para calificar un PId con un tipo de firma de interfaz, ni de detectar el tipo de la interfaz a la que el *PId* hace referencia.

A.2.2.2.5 Flujos, operaciones y señales

El modelo de sustitución de operaciones y trenes por señales está garantizado en SDL. En el caso de *REMOTE PROCEDURES* y *EXPORTED/IMPORTED*, la norma SDL proporciona un modelo de sustitución similar.

A.2.2.3 Reglas de vinculación

La interacción entre objetos computacionales sólo es posible cuando sus interfaces están vinculadas al mismo objeto vinculador. Esto lo asegura la semántica de SDL, ya que cada comunicación entre BLOCKS en SDL requiere que los *BLOCK*S estén conectados por un *CHANNEL*. Cada *OUTPUT* utilizará la cláusula *TO* <*PId*> o *VIA* <*channel*> o *VIA* <*gate*>.

A.2.2.3.1 Vinculación implícita para interfaces de operaciones de servidor

La vinculación implícita es posible en SDL para *BLOCKS* objeto computacional que están conectados directamente por un *CHANNEL*. La vinculación implícita siempre tiene lugar con operaciones modeladas con *REMOTE PROCEDURES*. El modelo de sustitución SDL garantiza el cumplimiento de las reglas de vinculación implícita para interacciones de operaciones de servidor. Sin embargo, la interfaz de operaciones de cliente tiene que ser creada explícitamente. El alcance de un *EXPORTED PROCEDURE* se restringe por una cláusula *REMOTE*.

A.2.2.3.2 Reglas de vinculación primitiva

La vinculación primitiva requiere que los *PROCESSES* interfaz de los dos objetos computacionales estén conectados directamente por *CHANNELS* como se indica en el A.9.

Cuadro A.9 – Acción vinculadora

```
/*Initiater*/

...
STATE unbound;
...
TASK
Destination:=CALL Bind(SELF,TDesc) TO Dest;
DECISON Destination=Dest;
FALSE: /*Binding Failure*/
  NEXTSTATE -;
TRUE: TASK BoundTo:= add(BoundTo,Dest)
  NEXTSTATE Bound;
ENDDECISION;
STATE Bound;
...
```

```
/*Destination*/
VIRTUAL EXPORTED PROCEDURE Bind
NEWTYPE PIDSet ATLEAST
        OPERATORS noequality;
                add: PIDSet,PId->PIDSet;
                remove: PIDSet,PId->PIDSet;
ENDNEWTYPE;
DCL BoundTo PIDSet,ThisType TypeDescrType, Failure Boolean>>
FPAR Source PId, Typ TypeDescrType;
RETURNS Dest PId;
START;
        /*type checking*

                NULL: TASK Failure:=true; RETURN NULL;
                ELSE: TASK BoundTo:=add(BoundTo,Source),
                        Failure:= false; RETURN SELF;
        ENDDECISION;
RETURN NULL
ENDPROCEDURE;
STATE unbound;
        INPUT Bind;
                DECISION failure;
                        false: NEXTSTATE Bound;
                ENDDECISION;
                NEXTSTATE -;
```

La acción vinculadora es modelada por un *EXPORTED PROCEDURE* de acuerdo con el cuadro A.9. Hay que proporcionar un *PROCEDURE* complementario para suprimir la vinculación.

A.2.2.3.3 Vinculación compuesta

Para utilizar la vinculación compuesta hay que especificar un objeto vinculador (*BLOCK TYPE*). Este *BLOCK TYPE* tiene que contener al menos dos *PROCESSES* interfaz y un *PROCESS* comportamiento local. Este objeto vinculador tiene que ser conectado a los objetos que intervienen en la vinculación por *CHANNELS*. Se pueden utilizar cláusulas *ATLEAST* para asegurar el cumplimiento de las condiciones previas de vinculación.

La vinculación compuesta comprende:

- ejemplificación del objeto vinculador;
- ejemplificación de las plantillas de interfaz dentro del objeto vinculador, las cuales se asocian con un cometido formal en la plantilla de objeto vinculador (esto puede formar parte de la ejemplificación del objeto);
- vinculación primitiva de las interfaces que intervienen en la vinculación con las correspondientes interfaces del objeto vinculador;
- ejemplificación de las interfaces de control que se necesiten.

Las interfaces de control pueden ser especificadas y ejemplificadas de acuerdo con las secciones A.2.1.10 y A.2.2.5.2.

A.2.2.4 Reglas de tipos

SDL no tiene medios para describir formalmente el concepto de tipo ODP. Hay que utilizar las reglas de la Rec. UIT-T X.903 | ISO/IEC 10746-3 sobre los tipos como una guía de estilos para el proceso de especificación. Las necesarias relaciones de subtipificación para la vinculación de interfaces tienen que expresarse como el comportamiento de la acción vinculadora (*PROCESS* vinculador).

Las reglas de tipos de plantillas se pueden indicar utilizando cláusulas *ATLEAST*.

NOTA – Para modelar el concepto de tipo se pueden utilizar tipos de datos ASN.1 o Act One; no obstante, la relación entre un objeto y el tipo de objeto no se puede verificar formalmente.

A.2.2.5 Reglas de plantillas

A.2.2.5.1 Reglas de plantillas de objetos computacionales

Un objeto computacional puede:

- iniciar señales o responder a señales (*INPUT/OUTPUT*);
- producir/consumir flujos;
- iniciar invocaciones de operaciones;

- responder a invocaciones de operaciones;
- iniciar terminaciones de operaciones;
- responder a terminaciones de operaciones;
- ejemplificar plantillas de interfaces (*CREATE* <*process*>);
- ejemplificar plantillas de objetos (*OUTPUT* CreateObject(<objectname>) *TO* LocalBehaviour);
- vincular interfaces;
- acceder a su estado y modificarlo (acciones *TASK*, *NEXTSTATE*);
- suprimir una o más de sus interfaces (*OUTPUT* Suprimir *TO* Interfaz)
- suprimirse a sí mismo (*STOP*);
- realizar actividades de ramificación, bifurcación y unión (*PROCESS* Creación, *SERVICE* descomposición);
- obtener un identificador de interfaz computacional para un ejemplar de la función de comercio.

A.2.2.5.2 Ejemplificación de interfaz computacional

La ejemplificación de plantilla de interfaz computacional:

- crea un nuevo PROCESS interfaz;
- produce un identificador de interfaz computacional para la interfaz (*PId*).

La ejemplificación se modela por *CREATE* <*interface-process-type*>. Las variables *SELF* del creador y OFFSPRING del creado contienen el nuevo identificador de interfaz.

A.2.2.5.3 Ejemplificación de plantilla de objeto computacional

La ejemplificación de plantilla de objeto computacional:

- crea un nuevo *PROCESS* LocalBehaviour para el objeto;
- produce un conjunto (no vacío) de identificadores para los *PROCESSES* interfaz iniciales del nuevo objeto.

La ejemplificación se modela mediante *CALL* CreateObject *TO* <*object-type*>. Esto crea un nuevo *PROCESS* LocalBehaviour. La ejemplificación de los *PROCESSES* interfaz del nuevo objeto puede incluirse en el *PROCEDURE* CreateObject o puede formar parte de la transición de comienzo de LocalBehaviour. El *PROCEDURE* CreateObject puede ser refinado por herencia

Cuadro A.10 – CreateOject PROCEDURE

```
EXPORTED PROCEDURE CreateObject ATLEAST CreateObject
RETURNS ObjectID PId;
    START VIRTUAL;  CREATE THIS;
    RETURN OFFSPRING;
ENDPROCEDURE CreateObject;
```

NOTA – El método aquí especificado se basa en el supuesto de que siempre existirá al menos un objeto de ese tipo. Si esto no puede se garantizar, hay que añadir un proceso (gestor) especial al ComputationalObjectTemplate-BLOCK. Su finalidad es la creación de nuevos ejemplares.

A.2.2.6 Reglas de fallos

Todos los fallos computacionales posibles deben ser especificados explícitamente. El SDL no proporciona un medio para tratar fallos en la ejecución de una especificación. Tras la aparición de un error (SDL), el comportamiento ulterior de un sistema no está definido.

A.2.2.7 Reglas de portabilidad

El SDL satisface todos los requisitos de las reglas de portabilidad con las siguientes excepciones:

- garantía del ordenamiento y la entrega de anuncios (la entrega de *SIGNALS* siempre tiene éxito o fracasa);
- prueba del subtipo de firma de interfaz.

El SDL proporciona un modelo de procesamiento basado en eventos; las acciones permitidas se representan directamente como parte del lenguaje (*INPUT,OUTPUT,CREATE,CALL*) y mediante estructuras sintácticas.

A.2.2.8 Punto de conformidad y punto de referencia

Las *GATES* de *PROCESSES* interfaz actúan como la comunicación al exterior del objeto computacional; representan los puntos de referencia.

Es posible utilizar adicionalmente diagramas de secuencias de mensajes (MSC, *message sequence charts*) para especificar el comportamiento requerido en un punto de referencia. Se dispone de metodologías de pruebas para verificar la conformidad entre una especificación SDL y una especificación MSC de la UIT.

Se ha creado un lenguaje que vincula el lenguaje de especificación de interfaz normalizado CORBA-IDL y el SDL; dicho lenguaje puede ser utilizado para probar la conformidad de un objeto en puntos de conformidad programáticos.

A.3 Formalización del lenguaje de punto de vista computacional en lenguaje Z

Estructuras elementales asociadas con interfaces de operaciones y de señales

Para formalizar los conceptos asociados con el punto de vista computacional utilizando el lenguaje Z, es necesario introducir etiquetas [Name] para cosas, por ejemplo, nombres de operaciones y sus tipos ODP. Los tipos ODP que existen en el sistema son indicados por [Type].

Los parámetros que están asociados con interfaces a objetos computacionales consisten en un nombre y un tipo. Siempre debe ser posible determinar el tipo de un parámetro en un contexto dado, por ejemplo, como se indica en 7.2.1 de la Rec. UIT-T X.903 | ISO/CEI 107476-3. Así, Param se introduce como una función parcial de nombres a tipos ODP en dicho contexto.

$$Param : Name \pfun Type$$

Esta función incluye en su dominio todos los nombres de parámetros que existen en un contexto dado. También es conveniente introducir secuencias de estos parámetros para poder considerar los conjuntos de parámetros asociados con señales, invocaciones o terminaciones.

$$PList == \mathbf{seq}\ Param$$

Estructuras elementales asociadas con interfaces de trenes

Al igual que en la formalización del punto de vista computacional mediante el empleo del lenguaje LOTOS, se considera una noción genérica de flujo constituido por elementos de flujo. Cada elemento de flujo en un flujo de información puede ser considerado como una unidad constituida por datos (estos pueden estar comprimidos) que son representadas por [Data]. Este modelo podría incluir cómo se comprimió la información, la información que se comprimió, etc., pero este aspecto no se examina detalladamente aquí. Los elementos de flujo contienen también una indicación de tiempo (ts) que se utiliza para modelar el instante en que el elemento de flujo en cuestión fue enviado o recibido, por lo que se introduce el tipo [Time]. En los flujos multimedios se requieren a menudo determinadas elementos de flujo para sincronización, por ejemplo, sincronización de audio con vídeo. Por esta razón, se asocia un Name particular con cada elemento de flujo. Este nombre se puede utilizar para seleccionar un determinado elemento de flujo de los contenidos en el flujo, según se requiera. En consecuencia, se puede modelar un elemento de flujo como:

```
┌─FlowElement──────────────
│label : Name
│data : Data
│ts : Time
└──────────────────────────
```

A.3.1 Conceptos

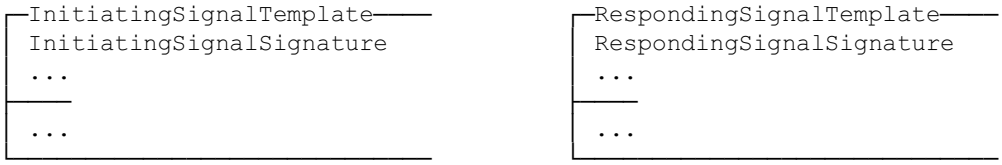
A.3.1.1 Señal

Una señal es una interacción atómica desde un iniciador hasta un respondedor. Dado que Z no posee completamente la característica, orientación al objeto, encapsulado, para el modelado de interacciones de objetos hay que imponer restricciones a los estilos de especificación que deban seguirse. Por ejemplo, se debe asegurar que las señales enviadas por iniciadores a respondedores tengan nombres de variable apropiados (y tipos compatibles) para las correspondientes

etiquetas de salida y de entrada, respectivamente. Para asegurar que las consideraciones relativas a las denominaciones serán satisfechas, se debe redenominar adecuadamente el texto del esquema que representa las firmas de señales, como se indica en A.3.1.11. En el siguiente fragmento en Z se presenta un ejemplo de la forma en que esto se puede conseguir:

```
InitiatingSignalSignature ≙ SignalSignature[pl!/inArgs]
RespondingSignalSignature ≙ SignalSignature[pl?/inArgs]
```

Ahora se puede dar una plantilla para una señal iniciadora y respondedora como:



En este caso los puntos se utilizan para indicar que probablemente habrá más información contenida en la parte de declaración de los esquemas, por ejemplo, en relación con información de estado de los objetos asociados con las señales iniciadora y respondedora, y predicados para indicar los efectos de los esquemas de operaciones que se producen, es decir, el comportamiento.

Cabe señalar que se puede utilizar ocultación de esquema y proyección de esquema para ocultar declaraciones que no deben ser visibles durante la interacción, es decir, pueden ser suprimidas de las declaraciones y cuantificadas existencialmente, en la parte de predicado del esquema. La plantilla de señal para la propia interacción puede ser modelada mediante la composición de las respectivas plantillas de esquemas de señal iniciadora y respondedora.

```
SignalTemplate ≙ InitiatingSignalTemplate >> RespondingSignalTemplate
```

El hecho de que la señal propiamente dicha pueda producirse realmente depende de que se satisfagan los predicados asociados con los esquemas compuestos.

NOTA – La información que es transportada como se requiere en 8.8 de la Rec. UIT-T X.902 | ISO/CEI 10746-2 es dada por los parámetros de salida de la señal iniciadora, es decir, `pl!`.

A.3.1.2 Operación

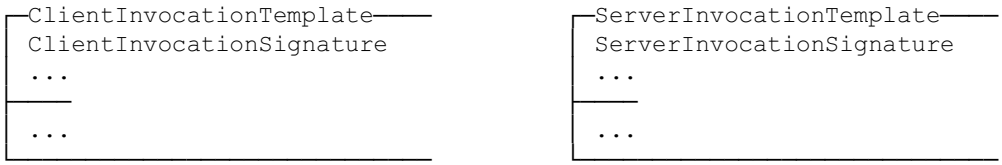
Aparición de un esquema de operación que modela una interrogación o un anuncio. Véase también la nota en A.3.1.4.

A.3.1.3 Anuncio

Aparición de un esquema de operación que modela una invocación de un cliente a un servidor. Como se ha indicado en A.3.1.1, Z no soporta plenamente la orientación a objetos, por lo que es necesario adoptar convenios de modelado para modelar sistemas de objetos que interactúan. Los convenios de denominación se pueden aplicar mediante una redenominación adecuada de la plantilla de invocación indicada en A.3.1.12. Esto se puede representar de la forma siguiente:

```
ClientInvocationSignature ≙ InvocationSignature[pl!/inArgs]
ServerInvocationSignature ≙ InvocationSignature[pl?/inArgs]
```

Ahora se puede dar una plantilla para una invocación de cliente y servidor como:



El propio anuncio puede ser modelado mediante la composición de los respectivos esquemas de invocación de cliente y servidor:

```
InvocationTemplate ≙ ClientInvocationTemplate >> ServerInvocationTemplate
```

El hecho de que el propio anuncio pueda producirse realmente depende de que se satisfagan los predicados asociados con los esquemas compuestos.

NOTA – El texto de A.3.1.1 que explica los puntos en los esquemas para representar el comportamiento no especificado y la utilización de ocultación y proyección de esquemas para proporcionar una forma de encapsulado es también aplicable a los anuncios.

A.3.1.4 Interrogación

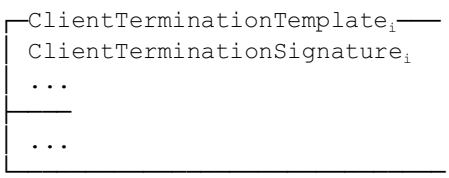
Aparición de un esquema de operación que modela una invocación entre un cliente y un servidor seguida por la aparición de un esquema de operación asociado que modela una terminación entre ese servidor y ese cliente. Como se indica en

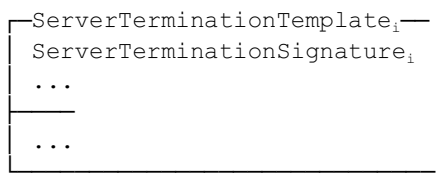
A.3.1.1, el lenguaje Z no soporta plenamente la orientación a objetos, por lo que es necesario adoptar convenios de modelado para modelar sistemas de objetos que interactúan. Los convenios sobre denominación se pueden aplicar mediante una redenominación adecuada de la plantilla de invocación indicada en A.3.1.12. Las invocaciones asociadas con interrogaciones son modeladas de manera similar a la utilizada para modelar invocaciones asociadas con anuncios, como se indica en A.3.1.3.

Como se expone en A.3.1.1, se puede utilizar ocultación de esquema y proyección de esquema para modelar una forma de encapsulado. Los convenios de terminaciones y denominación que habrán de seguirse pueden ser modelados mediante la redenominación de plantillas de terminación (indicadas en A.3.1.12). El lado cliente y el lado servidor de una terminación pueden representarse como:

$$ServerTerminationSignature_i \triangleq TerminationSignature[pl_i!/outArgs]$$
$$ClientTerminationSignature_i \triangleq TerminationSignature[pl_i?/outArgs]$$

En estas expresiones, el subíndice *i* indica que puede haber varias firmas de terminaciones asociadas con una sola invocación. Ahora se puede dar una plantilla para una terminación de cliente y de servidor como:





El subíndice *i* se utiliza para indicar que probablemente habrá varias plantillas de terminaciones, cada una de las cuales tiene asociada una firma. Estas firmas puedan ser diferentes.

Las propias terminaciones pueden ser modeladas mediante la composición de los respectivos esquemas de terminación de servidor y de cliente.

$$TerminationTemplate_1 \triangleq ServerTerminationTemplate_1 >> ClientTerminationTemplate_1$$
$$TerminationTemplate_2 \triangleq ServerTerminationTemplate_2 >> ClientTerminationTemplate_2$$
...

Por tanto, una interrogación al igual que una invocación seguida por una de las posibles terminaciones, se puede representar como:

$$InterrogationTemplate \triangleq InvocationTemplate >> (TerminationTemplate_1 \lor TerminationTemplate_2 \lor \ldots)$$

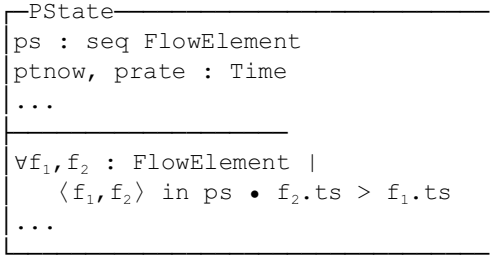
El hecho de que la propia interrogación pueda producirse o no depende de que se satisfagan los predicados asociados con los esquemas compuestos.

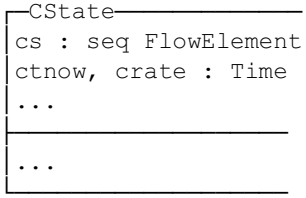
NOTA 1 – El texto de A.3.1.1 que explica el uso de los puntos en los esquemas para representar el comportamiento no especificado y la utilización de ocultación y proyección de esquemas para proporcionar una forma de encapsulado es también aplicable a los anuncios.

NOTA 2 – Este modelo de una interrogación representa un esquema de interrogación individual, es decir, éste no es el caso en que la invocación se produce primero y es seguida de una de las terminaciones. Toda la interrogación representa una sola acción atómica que se produce íntegramente, o no se produce en absoluto, lo que depende de los predicados asociados. Se debe utilizar el comentario informal asociado con la especificación Z para explicar el efecto deseado.

A.3.1.5 Flujo

El modelado de flujos en el lenguaje Z depende considerablemente del nivel de abstracción utilizado cuando se considera la secuencia de interacciones que representan el flujo. Por lo general, los flujos de información tienen asociadas estrictas consideraciones de temporización. Se examina también aquí un modelo basado en un productor de flujo que tiene que enviar una secuencia de ítems de datos (elementos de flujos) a un consumidor de flujo. Estos elementos de flujo son enviados por el productor con la indicación de tiempo, y ésta se utiliza para determinar su validez cuando llegan al consumidor. Los ejemplos del estado de un productor (*PState*) y de un estado del consumidor (*CState*) se pueden representar como:





Se utiliza aquí el modelo de elemento de flujo indicado en A.3.1. Se señala que el estado del productor (y del consumidor) consiste, al menos, en una secuencia de elementos de flujo (*ps/cs*), la hora o tiempo local actual (*ptnow/ctnow*) y la velocidad a la que los elementos de flujo son enviados (*prate*) o aceptados (*crate*). Se señala también que todos los elementos de flujo en la secuencia de elementos de flujo asociada con un productor llevan indicaciones de tiempo crecientes. Con este modelo del estado de productor, se puede modelar el envío de un elemento de flujo como:

```
┌─PSendFlowElement──────────────────────────────────────────
│ΔPState
│f!: FlowElement
│...
├──────────────────
│ps ≠ < > ∧ prate' = prate ∧
│ps' = tail ps ∧ ptnow' = ptnow + 1 / prate ∧
│f! == (μ FlowElement | data = (head ps).data ∧
│             ts = ptnow' ∧ label = (head ps).label) ...
└──────────────────────────────────────────────────────────
```

Con relación a esto hay que precisar varios puntos. El envío de un elemento de flujo, suprime dicho elemento de la secuencia de elementos de flujo que se han de enviar. La velocidad actual asociada al flujo no se modifica. La hora real en que se envió el elemento de flujo depende de la velocidad actual y de la hora (tiempo) actual.

El elemento de flujo real enviado es el elemento de flujo que encabeza la secuencia de elementos de flujo que se han de enviar. A este elemento de flujo se le pone una indicación de tiempo con el valor del tiempo calculado anteriormente. Se observa aquí que la utilización de la descripción definitiva requiere que se cumpla una obligación de prueba con el fin de asegurar que el elemento de flujo enviada es único. Esta obligación se satisface mediante el modelado de todos los elementos de flujo en la secuencia que tienen indicaciones de tiempo crecientes (es decir, no iguales). Puesto que en la secuencia no existen dos elementos de flujo que tengan el mismo sello de tiempo, el elemento de flujo enviado con la indicación de tiempo actual es único. También se requiere como una condición previa que la secuencia de elementos de flujo no esté vacía.

Un consumidor puede recibir correctamente un elemento de flujo siempre que se satisfagan las constricciones para su aceptación.

```
┌─CGetFlowElementOk─────────────────────────────────────────
│ΔCState
│f?: FlowElement
│...
├──────────────────
│cs' = cs^<f?> ∧ crate' = crate ∧ ctnow' = ctnow + 1 / crate ∧ ...
└──────────────────────────────────────────────────────────
```

Para mayor brevedad, no se consideran detalladamente las constricciones que se imponen a la aceptación. Éstas podrían entrañar que se permitieran variaciones en las horas en que el elemento de flujo es aceptable, por ejemplo, fluctuación de fase. El modelo real de un flujo se puede representar ahora como:

```
Flow ≙ PSendFlowElement >> CGetFlowElementOk
```

Se señala que este modelo de flujo requiere que el elemento de flujo enviado y el recibido tengan el mismo nombre de base y que las otras variables locales del estado del productor y del estado del consumidor tengan etiquetas diferentes. Para abreviar, no se consideran los casos de error relacionados con el envío y la recepción de elementos de flujo en un flujo de información.

A.3.1.6 Interfaz de señales

Interfaz en la cual todos los *esquemas de operaciones* son modelados como señales. En A.3.1.1 se presenta un ejemplo del formato de un *esquema de operación* que representa una firma de señal.

NOTA – La especificación de comportamiento y el contrato de entorno asociados con una interfaz dada deben ser representados por estructuras de datos Z adicionales, por ejemplo, esquemas que representan el estado de los objetos que intervienen en las interacciones en la interfaz, o tipos de datos que representan las constricciones impuestas a la interfaz. La creación de una determinada plantilla de interfaz debe satisfacer todos los predicados asociados con la plantilla de interfaz.

A.3.1.7 Interfaz de operaciones

Interfaz en la cual todos los *esquemas de operaciones* son modelados como operaciones. En A.3.1.3 y A.3.1.4, se presenta un ejemplo del formato de los *esquemas de operaciones* que representan partes de una firma de operación. Véase también la nota en A.3.1.6.

A.3.1.8 Interfaz de trenes

Interfaz en la cual todos los *esquemas de operaciones* son modelados como flujos. En A.3.1.5 se presenta un ejemplo del formato de los *esquemas de operaciones* que representan una firma de flujo. Véase también la nota en A.3.1.6.

A.3.1.9 Plantilla de objeto computacional

Plantilla de objeto (véase 4.4.2.11) que comprende un conjunto de plantillas de interfaz que el objeto puede ejemplificar, una especificación de comportamiento y un contrato de entorno. Debe señalarse que Z es esencialmente un lenguaje basado en una notación uniforme por lo que no soporta el modelado de objetos directamente como una característica del lenguaje. En cambio, el comentario en lenguaje ordinario que se debe asociar a cada especificación Z se deberá utilizar para indicar el texto Z, por ejemplo, los *esquemas de operaciones*, que comprenden la interfaz o interfaces de objetos y la relación entre ellos.

A.3.1.10 Plantilla de interfaz computacional

Plantilla de interfaz para una interfaz de señales, una interfaz de trenes o una interfaz de operaciones. Véase también la nota en A.3.1.6.

A.3.1.11 Firma de interfaz de señales

Firma de interfaz para una interfaz de señales. Una firma de interfaz de señales está constituida por un conjunto finito de plantillas de acción, una para cada tipo de señal en la interfaz. Cada plantilla de acción comprende el nombre de la señal, el número, nombre y tipos de sus parámetros y una indicación de causalidad (iniciadora o respondedora) con respecto al objeto que crea la planilla. Una firma de señal se puede representar por:

```
┌─SignalSignature──────────────
│inArgs: PList
└──────────────────────────────
```

El nombre de esquema (`SignalSignature`) se utiliza aquí para representar el nombre de señal, e *inArgs* se utiliza para representar el número, nombre y tipo de los parámetros asociados con la señal. La utilización de este esquema para crear firmas de señales iniciadoras y respondedoras, es decir, con una indicación de causalidad, se explica en A.3.1.1. Véase también la nota en A.3.1.6.

A.3.1.12 Firma de interfaz de operaciones

Firma de interfaz para una interfaz de operaciones. Una firma de interfaz de operaciones contiene un conjunto finito de anuncios e interrogaciones, según proceda, uno para cada operación en la interfaz, junto con una indicación de causalidad (cliente o servidor) para la interfaz en su conjunto con respecto al objeto que crea la plantilla. Los anuncios constan de invocaciones solamente. Las interrogaciones constan de una invocación seguida por una de las posibles terminaciones. invocaciones. Una firma de invocación se puede representar como:

```
┌─InvocationSignature──────────
│inArgs: PList
└──────────────────────────────
```

El nombre de esquema se utiliza aquí para representar el nombre de invocación e *inArgs* para representar el número, nombre y tipo de los parámetros asociados con la invocación. La utilización de este esquema para crear firmas de invocaciones de cliente o de servidor, esto es, con la causalidad asociada, se indica en A.3.1.3 y A.3.1.4. El predicado asociado con este esquema se utiliza para asegurar el cumplimiento de las reglas de denominación de parámetros, es decir, que los nombres de parámetros sean únicos en el contexto de una plantilla de invocación. Véase A.3.2.1.

Una firma de terminación se puede representar como:

```
┌─TerminationSignature─────────
│outArgs: PList
└──────────────────────────────
```

El nombre de esquema se utiliza aquí para representar el nombre de terminación y *outArgs* se utiliza para representar el número, nombre y tipo de los parámetros asociados con esta terminación. La utilización de este esquema para crear firmas de terminación de cliente o de servidor, es decir, con la causalidad asociada, se describe en A.3.1.3 y A.3.1.4. Es probable que haya predicados asociados con este esquema, por ejemplo, reglas de denominación, etc. como se indica en A.3.2.1. Estos predicados son similares a los anteriormente indicados para plantillas de invocación (con cambios de cuantificación apropiados, por ejemplo, la substitución de *inArgs* por *outArgs*). Véase también la nota en A.3.1.6.

A.3.1.13 Firma de interfaz de trenes

Firma de interfaz para una interfaz de trenes. Una firma de interfaz de trenes comprende un conjunto finito de plantillas de acción, una para cada tipo de flujo en la interfaz. Cada plantilla de acción para un flujo contiene el nombre del flujo, el tipo de información del flujo y una indicación de causalidad para el flujo (productor o consumidor) con respecto al objeto que ejemplifica la plantilla. Un ejemplo de una firma de flujo particular se presenta en A.3.1.5. La identificación de las firmas de flujo como parte de una determinada interfaz de trenes se puede efectuar mediante el texto informal de cada especificación Z. Véase la nota en A.3.1.6.

A.3.1.14 Objeto vinculador

Objeto que soporta una vinculación de un conjunto de otros objetos computacionales. Dado que Z no soporta el modelado de objetos y sus interfaces asociadas como una característica del lenguaje, el modelado de objetos vinculadores en general está limitado. Sin embargo, utilizando el cálculo de esquema y proporcionando descripciones textuales informales es posible modelar complejos escenarios de interacciones en los que puede considerarse que existe una forma de vinculación. En el caso de objetos vinculadores entre un objeto cliente y un objeto servidor, por ejemplo, esto se lograr mediante el modelado de esquemas de operaciones adicionales (que representen partes de la interfaz con el objeto vinculador) que se componen de invocaciones de cliente y su ulterior entrega a servidores. Esto se podría representar como:

```
InvocationViaBind ≙ (ClientInvocation >> BindInvocation) >> ServerInvocation
```

El esquema *BindInvocation* debe tener tipos de datos compatibles para las variables que representan la información que se pasa entre el cliente y el servidor. Un ejemplo del esquema *BindInvocation* para la invocación de cliente y de servidor que figura en A.3.1.3 es el siguiente:

```
┌─BindInvocation──────
│inArgs!,inArgs?: PList
│...
├─────────────────────
│...
└─────────────────────
```

Aquí, el esquema debe tener el mismo nombre de base para las estructuras de datos que se pasan desde el cliente (*inArgs?*) y que se pasan al servidor (*inArgs!*). Se ha de señalar que no existe una noción del esquema *InvocationViaBind* que falle parcialmente. Es decir, no se da el caso de que la invocación de cliente (*ClientInvocation*) y su aceptación por el objeto vinculador (*BindInvocation*) puedan pasar, y que la entrega de la invocación desde el objeto vinculador al servidor (*ServerInvocation*) pueda fallar. Esto se puede representar entonces como:

```
InvocationViaBindFail ≙ ClientInvocation >> BindInvocationFail
```

En esta situación, el esquema *BindInvocationFail* podría ser modelado como:

```
┌─BindInvocationFail──────
│inArgs?: PList
│...
├─────────────────────
│...
└─────────────────────
```

Es decir, el objeto vinculador acepta los datos procedentes del cliente (*inArgs?*) pero no los que entrega al servidor por alguna razón no especificada. Las diferentes posibilidades de operaciones exitosas y no exitosas que podrían tener lugar a través de un objeto vinculador pueden representarse mediante el cálculo de esquema. Típicamente, se utiliza la disyunción lógica para representar las elecciones que son posibles, es decir, los casos de fallo.

NOTA – El comportamiento asociado con el esquema *BindInvocation* podría imponer constricciones a los datos que recibe y subsiguientemente envía, o sea, es posible escribir predicados en los valores de las variables que acepta como entradas y que presenta como salidas.

A.3.2 Reglas de estructuración del punto de vista computacional

A.3.2.1 Reglas de denominación

La reglas de denominación del leguaje de punto de vista computacional pueden ser soportadas mediante la escritura de predicados, como se muestra en A.3.1.12 para las reglas de denominación asociadas con parámetros, o dando un alcance global a los nombres de esquemas. Por tanto, no es posible declarar dos esquemas con los mismos nombres; esto es, todas las acciones están identificadas inequívocamente en una especificación Z semánticamente correcta.

A.3.2.2 Reglas de interacción

No suele darse el caso en Z de que se agrupen *esquemas de operaciones* para formar una nueva construcción Z, por ejemplo un esquema que represente la interfaz de un objeto. Hacer esto en el caso general conduciría a un esquema que no tiene la misma estructuración modular y con predicados potencialmente conflictivos que representaran el comportamiento de los esquemas individuales. Por consiguiente, las reglas de interacción de la Rec. UIT-T X.903 | ISO/CEI 10746-3 en general no son soportadas en el lenguaje Z.

A.3.2.2.1 Reglas de interacción de señales

La noción de causalidad no existe en Z, por lo que no tiene sentido expresar que las interfaces inicien señales si tienen causalidad iniciadora o que respondan a señales si tienen causalidad respondedora. La etiqueta de causalidad que puede aplicarse a una interfaz dada tiene carácter informal. Sin embargo, podría darse el caso de que nociones de causalidad pudieran tratarse en el comentario informal asociado a cada especificación Z, en compañía de combinaciones de esquemas apropiadas, por ejemplo, mediante >>.

A.3.2.2.2 Reglas de interacción de trenes

Véanse A.3.2.2 y A.3.2.2.1.

A.3.2.2.3 Reglas de interacción de operaciones

Véanse A.3.2.2 y A.3.2.2.1. Debe señalarse también que no se da frecuentemente el caso de que Z modele la secuenciación u ordenamiento de acciones. Esto se efectúa generalmente cuando se refina una especificación. Así, dado que un cliente envía una invocación que es recibida por un servidor, no existe una construcción propia del lenguaje Z que requiera que un servidor envíe ulteriormente una terminación adecuada. En cambio, el envío y la recepción de la invocación, respectivamente por el cliente al servidor, y los subsiguientes envío y recepción de la terminación, respectivamente por el servidor al cliente, son modeladas usualmente por un solo esquema como se muestra en el ejemplo de A.3.1.4. Como otra posibilidad, las acciones de envío y recepción de una invocación y de envío y recepción de una terminación pueden ser modeladas como esquemas distintos en los que se utiliza texto informal acompañante para explicar su relación.

A.3.2.2.4 Reglas de parámetros

No suele darse el caso en el lenguaje Z de que se agrupen *esquemas de operaciones* para formar la interfaz de un objeto que pueda ser etiquetada y subsiguientemente utilizada para interactuar con la interfaz a que hace referencia. El lenguaje Z, como tal, no soporta directamente el modelado de referencias de interfaces computacionales como parámetros.

A.3.2.2.5 Flujos, operaciones y señales

No hay distinción inherente, en el lenguaje Z, entre un flujo, una operación y una señal. Todos ellos se representan utilizando esquemas de operaciones que pueden ser compuestos, unos con otros, de muchas formas diferentes, por ejemplo, mediante el cálculo de esquema, según el comportamiento del sistema que se especifica. En consecuencia, el modelado de flujos u operaciones por medio de señales se puede efectuar asegurando que los esquemas que representan las señales tienen las etiquetas y los tipos de datos adecuados asociados con el esquema correspondiente que representa el flujo o la operación, respectivamente.

A.3.2.3 Reglas de vinculación

No suele darse el caso en el lenguaje Z de que se agrupen *esquemas de operaciones* para formar la interfaz de un objeto que pueda ser etiquetada y subsiguientemente utilizada para interactuar con la interfaz a que hace referencia. Por tanto, una especificación Z no soporta las reglas de vinculación de la Rec. UIT-T X.903 | ISO/CEI 10746-3. En cambio es más frecuente que Z soporte un forma de vinculación basada en *esquemas de operaciones* individuales (que representan señales, flujos, invocaciones o terminaciones), compuestos uno con otro. En A.3.1.14 se presenta un ejemplo. De esta manera, es posible asegurar que se cumplan ciertas reglas de vinculación, por ejemplo, mediante la escritura de predicados para comprobar los tipos de los parámetros que se pasan. Sin embargo, en el lenguaje Z no hay una característica que restrinja la forma en que los *esquemas de operaciones* puedan ser compuestos de manera general, por ejemplo, asegurando que sólo *esquemas de operaciones* que tengan cierto nombre y declaraciones similares sean compuestos uno con otro. La composición de esquemas que son incompatibles, por ejemplo la combinación, por medio de $\hat{\wp}$, de esquemas que tengan declaraciones de variables con nombres de base similares pero tipos diferentes, produce una especificación semánticamente incorrecta.

A.3.2.3.1 Reglas de vinculación implícita para interfaces de operaciones de servidor

Véanse A.3.2.2, A.3.2.3 y A.3.2.2.4.

A.3.2.3.2 Reglas de vinculación primitiva

Véanse A.3.2.2, A.3.2.3 y A.3.2.2.4.

A.3.2.3.3 Reglas de vinculación compuesta

Véanse A.3.2.2, A.3.2.3 y A.3.2.2.4.

A.3.2.4 Reglas de tipos

Las reglas de tipos contenidas en la Rec. UIT-T X.903 | ISO/CEI 10746-3 en general no son soportadas en el lenguaje Z por las razones indicadas en A.3.2.2, A.3.2.3 y A.3.2.2.4.

A.3.2.4.1 Reglas de subtipificación de firmas para interfaces de señales

Véase A.3.2.4.

A.3.2.4.2 Reglas de subtipificación de firmas para interfaces de trenes

Véase A.3.2.4.

A.3.2.4.3 Reglas de subtipificación de firmas para interfaces de operaciones

Véase A.3.2.4.

A.3.2.5 Reglas de plantillas

A.3.2.5.1 Reglas de plantillas de objetos computacionales

El lenguaje Z soporta todas las acciones asociadas con las reglas de plantillas de objetos computacionales (a condición que se proporcione un texto Z apropiado, por ejemplo esquemas para modelar señales iniciadoras o respondedoras) con las excepciones siguientes:

- la vinculación de interfaces no está plenamente soportada, por la razones indicadas en A.3.2.3 y A.3.2.2.4;
- la ramificación, bifurcación y unión de actividades no son características del lenguaje Z; sin embargo, es posible modelar una representación abstracta de esas actividades;
- la obtención de una referencia a una función de comercio no está plenamente soportada, por las razones indicadas en A.3.2.3 y A.3.2.2.4;
- Z no soporta la comprobación del subtipo de la firma de interfaz computacional, por las razones indicadas en A.3.2.3 y A.3.2.2.4.

A.3.2.5.2 Ejemplificación de plantilla de interfaz computacional

Se puede ejemplificar una plantilla de interfaz computacional proporcionando una vinculación válida para las variables en el texto Z identificado como representativo de la plantilla de interfaz. No suele darse el caso en Z de que se genere un identificador que pueda utilizarse para interactuar con la interfaz creada, por la razones indicadas en A.3.2.3.

A.3.2.5.3 Ejemplificación de plantilla de objeto computacional

Se puede ejemplificar una plantilla de objeto computacional proporcionando una vinculación válida para las variables en el texto Z identificado como representativo de la plantilla de objeto.

A.3.2.6 Reglas de fallos

Los modos de fallo visibles para un objeto se determinan por su especificación de comportamiento y su contrato de entorno. Cualquiera de las acciones indicadas en A.3.2.5 puede fallar.

En el lenguaje Z, es posible modelar los fallos de varias formas. Quizás la más sencilla sea mediante disyunción lógica de los esquemas. Es decir, se especifica un esquema correcto junto con posibles esquemas de error. Los esquemas de error se combinan entonces con el esquema correcto mediante disyunción lógica. La visibilidad de estos esquemas de error depende de la posesión de entradas (**?**) y salidas (**!**). Dado que los fallos de las señales son visibles e idénticos para todas las partes que intervienen en la interacción, un fallo de señal puede especificarse como un esquema con salidas que se envían a todas las partes que intervienen en la interacción.

De acuerdo con los requisitos, el flujo, los fallos de esquemas de operaciones y los fallos asociados con la ejemplificación de interfaces y de objetos pueden ser especificados de modo que: sean visibles para todas las partes al mismo tiempo; sean visibles para una sola parte en un determinado tiempo; y retornen diferentes parámetros a las diferentes partes. Todo esto se puede lograr mediante cálculo de esquema en Z.

A.3.2.7 Reglas de portabilidad

El lenguaje Z soporta las acciones requeridas para aplicar una norma de portabilidad básica o ampliada, a condición de que se proporcione texto Z para representar la acción pertinente, con las excepciones indicadas en A.3.2.5.1.

A.3.2.8 Puntos de conformidad y puntos de referencia

En toda interfaz de todo objeto computacional existe un punto de referencia. Los implementadores que alegan conformidad deben indicar los puntos de referencia de ingeniería que corresponden al punto de referencia computacional e indicar las transparencias y estructuras de ingeniería que son aplicables a los mismos. Al hacer esto, los puntos de referencia computacionales se convierten en puntos de conformidad.

Así, dada una especificación del punto de vista de computacional escrita en Z, la conformidad puede verificarse cerciorándose de que la implementación satisface los esquemas de operaciones asociados con aquellas interfaces que han sido identificadas por el implementador, en la especificación, como puntos de conformidad. En este caso, la satisfacción debe asegurar que las invariantes de la especificación no son violadas y que todas las condiciones previas y posteriores satisfechas por la especificación son satisfechas también por la implementación.

A.4 Formalización del lenguaje de punto de vista computacional en ESTELLE

Esta subcláusula muestra cómo la técnica de descripción formal Estelle se puede utilizar para interpretar los conceptos y reglas del lenguaje computacional de la Rec. UIT-T X.903 | ISO/CEI 10746-3. En el texto que sigue se utiliza la escritura en *cursiva* para indicar conceptos de Estelle definidos en el documento de la norma. La interpretación se explica por medio de ejemplos. Dado que no todos los conceptos del ODP pueden interpretarse formalmente en Estelle, a veces se utiliza un estilo que implica el concepto apropiado.

A.4.1 Conceptos

A.4.1.1 Señal

Una señal se interpreta como el depósito de una *interacción* Estelle a través de un *punto de interacción* de un *caso de módulo* y su subsiguiente consumo por el receptor.

A.4.1.2 Operación

Aparición de un anuncio o de una interrogación.

A.4.1.3 Anuncio

Un anuncio se interpreta como el depósito de una *interacción* Estelle a través de un *punto de interacción* de un *ejemplar de módulo* que representa al cliente y su consiguiente utilización por un *ejemplar de módulo* que representa al servidor.

A.4.1.4 Interrogación

Una interrogación se interpreta como el depósito de *interacciones* Estelle. Una *interacción*, la **invocación**, es depositada desde un *ejemplar de módulo* que representa al cliente. La otra *interacción*, la **terminación**, es depositada desde un *ejemplar de módulo* que representa al servidor. La gestión de las relaciones entre invocaciones y terminaciones se efectúa en los *ejemplares de módulos*.

A.4.1.5 Flujo

Hay dos formas de interpretar este concepto en Estelle. En la primera, un flujo se modela mediante una secuencia de *interacciones* enviadas desde un objeto productor a un objeto consumidor. Esto puede realizarse ligando un *módulo vástago* al *punto de interacción*, el cual genera la secuencia de *interacciones*. En la segunda, un flujo se modela mediante una *interacción* continua entre ambos módulos, y los datos transportados se pueden utilizar para representar un flujo de información.

A.4.1.6 Interfaz de señales

Una interfaz de señales se interpreta como un *punto de interacción* en Estelle. El *punto de interacción* tiene un conjunto definido de *interacciones* que sólo pueden ser señales. La definición del *punto de interacción* se basa en una *definición de canal* apropiada.

A.4.1.7 Interfaz de operaciones

Una interfaz de operaciones se interpreta como un *punto de interacción* en Estelle. El *punto de interacción* tiene un conjunto definido de *interacciones* que sólo pueden ser invocaciones y sus respectivas terminaciones y anuncios. La definición del *punto de interacción* se basa en la respectiva *definición de canal*.

A.4.1.8 Interfaz de trenes

Una interfaz de trenes se interpreta como un *punto de interacción* en Estelle. El *punto de interacción* tiene un conjunto definido de *interacciones* que sólo pueden ser flujos. La definición de los *puntos de interacción* se basa en la *definición de canal* apropiada. Si el flujo se modela mediante una secuencia de *interacciones*, se liga un *ejemplar de módulo vástago* al *punto de interacción* que realiza la secuencia de *interacciones*.

A.4.1.9 Plantilla de objeto computacional

Una plantilla de objeto computacional puede ser modelada utilizando una *definición de encabezamiento de módulo* que contiene los *puntos de interacción*, y una *definición de cuerpo de módulo* que contiene el comportamiento interno y el comportamiento de las plantillas de interfaz. El lenguaje Estelle no proporciona un medio de especificar explícitamente constricciones del entorno.

A.4.1.10 Plantilla de interfaz computacional

Una plantilla de interfaz computacional se modela utilizando una *definición de canal* que contiene las firmas de interfaz, utilizando una *definición de encabezamiento de módulo* que contiene los *puntos de interacción* apropiados, y utilizando una *definición de cuerpo de módulo* que contiene la especificación de comportamiento de la interfaz. Las constricciones del entorno no pueden modelarse explícitamente en Estelle. Una plantilla de interfaz computacional en Estelle consiste en un *punto de interacción* de un *ejemplar de módulo progenitor* que está ligada a un *ejemplar de módulo vástago* con el mismo *punto de interacción* del mismo canal. Este método proporciona la recepción transparente de diversas *interacciones*, porque el *ejemplar de módulo vástago* siempre está listo para recepción.

A.4.1.11 Firma de interfaz de señales

Una firma de interfaz de señales se interpreta como una parte de la *definición de canal*. Esta definición contiene los cometidos asociados con el *canal*, el nombre de las *interacciones* y las informaciones transportadas por ellas, especificados mediante uno o más nombres de parámetro y los respectivos tipos.

A.4.1.12 Firma de interfaz de operaciones

Una firma de interfaz de operaciones se interpreta como una parte de la *definición de canal*. Esta definición contiene los cometidos asociados con el *canal*, el nombre de la *interacción* y las informaciones transportadas por ella, especificados mediante nombres de parámetros y los tipos asociados. Una firma de anuncio contiene solamente una *interacción* que representa la invocación. Una firma de interrogación contiene una *interacción* que representa la invocación y contiene también un conjunto de *interacciones* que representan las posibles terminaciones.

A.4.1.13 Firma de interfaz de trenes

Una firma de interfaz de trenes se interpreta como una parte de la *definición de canal*. Esta definición contiene los cometidos asociados con el *canal*, el nombre de las *interacciones* y las informaciones transportadas por ellas, especificados mediante un nombres de parámetro y el tipo asociado.

A.4.1.14 Objeto vinculador

Un objeto vinculador se modela mediante un *ejemplar de módulo*, junto con los *CHANNELS* a través de los cuales se conecta a objetos computacionales.

A.4.2 Reglas de estructuración

En Estelle, una especificación describe la descomposición funcional de un sistema ODP como una configuración de objetos computacionales y vinculadores, representados, ambos tipos de objetos, como *ejemplares de módulos*. Las acciones internas de un *ejemplar de módulo* son modeladas como una máquina de estado finitos ampliada. Esta máquina se realiza en Estelle utilizando transiciones. Las *interacciones* entre objetos se intercambian a través de *CHANNELS*. Las *interacciones* entre *módulos* son asíncronas. Los puntos extremos de un *canal* son interfaces computacionales representadas en Estelle como *puntos de interacción*. Todos los *ejemplares de módulo* son *ejemplares de módulo vástago*

directos de un *ejemplar de módulo* con el atributo *systemactivity*. Esto significa que la ejecución de todos los *módulos vástagos* es no-determinística. El *ejemplar de módulo progenitor* sólo puede ser subestructurado en *módulos* con el atributo *activity*. El *módulo progenitor* prevé: ejemplificación; destrucción; conexión y desconexión de *ejemplares de módulos* que representan objetos computacionales. Los contratos de entorno de los objetos no son modelados explícitamente en Estelle.

A.4.2.1 Reglas de denominación

La semántica de Estelle garantiza el cumplimiento de todas las reglas de denominación. Los nombres en Estelle tienen asociado el siguiente contexto: *interacción, cometido* y *canal* forman parte de una *definición de canal*. No hay diferencia sintáctica entre las diversas *interacciones*. Las diferencias semánticas provienen de la diferente utilización de las *interacciones*. Se utiliza *identificador de canal* e *identificador de cometido* para la definición de *puntos de interacción* en la *definición de encabezamiento de módulo*. Un nombre de *interacción* se define en el contexto del *ejemplar de módulo* combinándolo con el respectivo nombre de *punto de interacción*. El nombre de parámetro de una *interacción* es un identificador en el contexto de la *definición de interacción*. El nombre de un *punto de interacción* se define en el contexto de todos los *ejemplares de módulo* cuya *definición de encabezamiento de módulo* han definido este punto de interacción.

El nombre de un *ejemplar de módulo* es válido en la *definición de cuerpo de módulo* en que ha sido definido.

A.4.2.2 Reglas de interacción

En Estelle, las *interacciones* sólo pueden ser enviadas y recibidas en *puntos de interacción*. El envío de una *interacción* a través de un *punto de interacción* que no está conectado provoca la pérdida de la *interacción*. El *ejemplar de módulo* emisor no es informado de su error. Si hay que informar un fallo de la infraestructura cuando se produce una *interacción* en una interfaz que no ha sido vinculada, todos los casos de un objeto que no están vinculados pueden ser conectados a *puntos de interacción* internos del entorno. Por tanto, las *interacciones* en cualquiera de estas interfaces pueden ser detectadas e informadas como un fallo de la infraestructura.

A.4.2.2.1 Reglas de interacción de señales

Cuando un *ejemplar de módulo* representa un objeto que ofrece una interfaz de señales de un determinado tipo de interfaz de señales, esta interfaz se representa mediante un *punto de interacción* de una *definición de canal* y *de cometido* apropiada, a la que está ligado un *ejemplar de módulo vástago* que modela el comportamiento en la interfaz. Este *ejemplar de módulo vástago* sólo puede presentar a la salida *interacciones* con el parámetro de causalidad fijado a iniciador. Debe contener una transición con las *cláusulas WHEN* apropiadas para la recepción de cualquiera de las *interacciones* definidas en la plantilla de interfaz de señales.

A.4.2.2.2 Reglas de interacción de trenes

Una interfaz de trenes se modela mediante un *punto de interacción* al que está ligado un *ejemplar de módulo vástago* que modela el comportamiento en la interfaz. Un flujo se puede representar por una *interacción* o por una secuencia de *interacciones*, según las diversas modalidades de un flujo. Si el objeto tiene un cometido de consumidor para esta interfaz, el *ejemplar de módulo vástago* tiene que contener una transición con las *cláusulas WHEN* apropiadas para la recepción de la *interacción* de flujo. Si el objeto tiene un cometido de productor, el *ejemplar de módulo vástago* tiene que contener enunciados de salida para la *interacción* de flujo.

A.4.2.2.3 Reglas de interacción de operaciones

Cuando un *ejemplar de módulo* representa un objeto que ofrece una interfaz de operaciones de un determinado tipo de interfaz de operaciones, esta interfaz se representa mediante un *punto de interacción* de una definición de *canal* y de *cometido* apropiada, al que está ligado un *ejemplar de módulo vástago* que modela el comportamiento en la interfaz. Si el objeto tiene el cometido de cliente para esta interfaz, el *ejemplar de módulo vástago* tiene que contener una transición con la *cláusula WHEN* apropiada para la recepción de terminaciones y los enunciados de salida apropiados para entregar invocaciones. Si el objeto tiene el cometido de servidor para esta interfaz, el *ejemplar de módulo vástago* tiene que contener una transición con la *cláusula WHEN* apropiada para la recepción de invocaciones y los enunciados de salida apropiados para entregar terminaciones. El lenguaje Estelle no proporciona un medio de asignar una duración a una transición o a una secuencia de transiciones.

A.4.2.2.4 Reglas de parámetros

Para un identificador de interfaz computacional hay que definir un tipo de datos (Estelle) específico. Cada interfaz computacional puede ser identificada por una variable de este tipo. Estas variables se pueden pasar como argumentos en interacciones que representen operaciones. El receptor de este identificador de interfaz computacional puede utilizarlo ulteriormente para vincularse a la interfaz a que se hace referencia.

A.4.2.2.5 Flujos, operaciones y señales

Para el intercambio de información entre módulos Estelle siempre se utilizan *interacciones*. Cuando se emplea este método, la modalidad de la información no es visible. Una señal se interpreta únicamente utilizando el modelo de interacción. Como la interpretación de los flujos y operaciones se basa en la noción de interacción, se pueden utilizar señales para explicar sus características especiales. En Estelle, sólo pueden conectarse *puntos de interacción* complementarios del mismo canal. Partiendo del supuesto que todas las *interacciones* en un *punto de interacción* son del mismo tipo, una vinculación compuesta entre diferentes clases de interfaces no es posible en Estelle.

A.4.2.3 Reglas de vinculación

Para permitir la *interacción* entre interfaces de objetos computacionales, los dos *puntos de interacción* que modelan estas interfaces se conectan a *puntos de interacción* del mismo objeto vinculador. El objeto vinculador es ejemplificado por una acción vinculadora. Una acción vinculadora es invocada por el entorno de los *ejemplares de módulos*. Cada objeto computacional posee un *punto de interacción* externo designado a través del cual se conecta con (un *punto de interacción* interno de) su entorno, es decir el *ejemplar de módulo* circundante. En este *punto de interacción* se invoca la acción vinculadora con los parámetros adecuados.

Cabe señalar que este *punto de interacción* puede ser considerado como una modalidad especial de interfaz en el sentido del ODP, pero la acción vinculadora propiamente dicha no es una interacción ODP, pues no se produce entre objetos computacionales. Una interfaz de control del objeto vinculador se conecta a una correspondiente interfaz de control del objeto cliente. El entorno, esto es, el *ejemplar de módulo* circundante, puede verificar si las interfaces que habrán de ser vinculadas están en una relación de tipo apropiada. Dado que (hasta el presente) no son modeladas explícitamente contratos de entorno, no se puede comprobar el cumplimiento de los contratos de entorno. Una vinculación sin una participación explícita del entorno no puede ser interpretada con conceptos Estelle.

A.4.2.3.1 Reglas de vinculación implícita para interfaces de operaciones de servidor

Se requiere vinculación implícita si el objeto cliente y el objeto servidor todavía no han sido vinculados. Estelle no proporciona un medio para detectar la inexistencia de esta vinculación. El cliente tiene que manejar una matriz con *puntos de interacción* y servidores vinculados.

A.4.2.3.2 Reglas de vinculación primitiva

En el caso de la vinculación primitiva no interviene ningún objeto vinculador. Los *puntos de interacción* de *módulos* que modelan objetos computacionales se conectan directamente. Esta conexión puede ser establecida durante la ejemplificación, o puede ser efectuada por el entorno tras la invocación de la función respectiva. En el segundo caso, habrá una notificación si se produce un fallo.

A.4.2.3.3 Reglas de vinculación compuesta

Las acciones de vinculación compuesta permiten vincular un conjunto de interfaces, por medio de un objeto vinculador para soportar la vinculación. Una plantilla de objeto vinculador puede ser interpretar en Estelle de la misma manera que una plantilla de objeto computacional. Las condiciones previas para la vinculación compuesta se pueden formular con las siguientes expresiones Estelle:

- Los parámetros de interfaz correspondientes deben ser de una misma clase. Esto se puede realizar en Estelle utilizando una *definición de canal* válida.
- Los parámetros de interfaz correspondientes deben ser causalidad complementaria. Esto se puede realizar en Estelle utilizando una *definición de canal* válida y el *cometido* respectivo.
- El correspondiente parámetro de interfaz debe ser un subtipo del tipo de firma. Aunque Estelle no soporta la subtipificación, ésta se puede realizar utilizando una *definición de canal* válida.

Una acción vinculadora compuesta puede ser interpretada empleando las siguientes construcciones Estelle:

- Un caso de módulo para un objeto vinculador es inicializado por el *módulo progenitor*. El *encabezamiento de módulo*, el *cuerpo de módulo* y una *variable de módulo* han sido previamente definidas.
- Durante la ejemplificación, los *puntos de interacción* son también ejemplificados.
- Utilizando el *enunciado de conexión* se efectuará una vinculación primtiva entre el objeto vinculador y los otros objetos computacionales.
- Se han ejemplificado las interfaces de control. Su identificador puede ser devuelto.
- Las funciones necesarias en la interfaz de control pueden ser proporcionadas por el *ejemplar de módulo*.

A.4.2.4 Reglas de tipos

En Estelle, los *puntos de interacción* sólo puede ser conectados si están definidos con la misma *definición de canal*. No hay relación de subtipificación entre *puntos de interacción*. Para vincular, por medio de un objeto vinculador, interfaces que guardan una relación de subtipificación, dicho objeto vinculador debe tener diferentes tipos de interfaz para sus interfaces con cometido de cliente (consumidor) y con cometido de servidor (productor). Dado que Estelle no soporta la definición recursiva de interfaces, sólo pueden aplicarse las reglas de subtipificación simplificada. Los tipos de datos Estelle se basan en los tipos de datos de la versión del lenguaje PASCAL adoptada por la ISO (ISO PASCAL). Por tanto, la única subtipificación que se soporta son los tipos subgama. Los tipos subgama pueden definirse para tipos de datos ordinales, a saber, el tipo de datos entero y los tipos enumerados definidos por el usuario.

A.4.2.5 Reglas de plantillas

A.4.2.5.1 Reglas de plantilla de objeto computacional

Un objeto computacional puede:

- iniciar una señal ejecutando los correspondientes *enunciados de salida* para la *interacción* que representa la señal, o responder a señales ejecutando las correspondientes *cláusulas WHEN* en un *punto de interacción* que representa una interfaz de señales. Los *enunciados de salida* y las *cláusulas* WHEN se realizan en *módulos vástagos* ligados al *punto de interacción* del *módulo progenitor*;
- producir flujos ejecutando los correspondientes *enunciados de salida* para la *interacción* que representa el flujo, o consumiendo flujos ejecutado las correspondientes *cláusulas WHEN* en un *punto de interacción* que representa una interfaz de trenes. Los *enunciados de salida* y las *cláusulas WHEN* se realizan en *módulos hijo* ligados al *punto de interacción* del *módulo padre*. Dentro del *módulo hijo* puede existir un bucle para iniciar la secuencia de *interacciones* que modelan un flujo;
- invocar operaciones ejecutando el correspondiente *enunciado de salida* para la interacción que representa la invocación de operación, o respondiendo a invocaciones de operaciones ejecutando la correspondiente *cláusula WHEN* en una interfaz de operaciones. Los *enunciados de salida* y las *cláusulas WHEN* se realizan en *módulos hijo* ligados al *punto de interacción* del *módulo padre*;
- terminar operaciones ejecutando el correspondiente *enunciado de salida* para una *interacción* que representa una operación de terminación;
- ejemplificar plantillas de interfaces asignando un *punto de interacción* no utilizado a la nueva interfaz, ejemplificando el *ejemplar de módulo vástago* que define el comportamiento en la interfaz y ligándola al *punto de interacción*;
- ejemplificar plantillas de objetos directamente, ejemplificando un nuevo ejemplar de la correspondiente *definición de módulo* (de esta forma sólo se pueden ejemplificar objetos internos al objeto creador), o indirectamente, mediante el intercambio de *interacciones* con el entorno;
- vincular interfaces invocando una acción vinculadora con el entorno;
- acceder a su estado y modificarlo, ejecutando una transición de estado o modificando variables;
- suprimirse a sí mismo indirectamente solicitando a sí mismo ser detenido por el entorno;
- ramificar una actividad creando un *ejemplar de módulo vástago*;
- vincularse a un objeto de función de comercio invocando una acción vinculadora con el entorno.

Es preciso señalar que Estelle no soporta actividades de bifurcación ni de unión. Se ha de señalar también que las interfaces no pueden ser suprimidas; y que sólo se puede desconectar las conexiones entre *puntos de interacción*.

A.4.2.5.2 Ejemplificación de plantilla de interfaz computacional

Un *punto de interacción* no utilizado se asigna a la nueva interfaz, un *ejemplar de módulo vástago* es creado por el *módulo progenitor* y ligado al *punto de interacción*, y se produce un identificador de interfaz interno para la interfaz. La creación dinámica de *puntos de interacción* no es posible en Estelle. Esto implica que cada *ejemplar de módulo* se crea (para cada tipo de interfaz) con una matriz de *puntos de interacción* que representa el número máximo de interfaces de este tipo que existen simultáneamente. En un instante dado, un *punto de interacción* no es utilizado, o representa una interfaz (está asignado a una interfaz). En una estructura de datos adecuada, el *módulo progenitor* tiene que mantener información sobre el estado de cada *punto de interacción*. Las interfaces iniciales del objeto son ejemplificadas en la *transición de inicialización* del *módulo*. Cada *módulo* genera un identificador de interfaz interno para cada una de sus interfaces. El identificador computacional externo completo es creado por el entorno del objeto utilizando el identificador de interfaz interno y un *identificador de módulo*.

A.4.2.5.3 Ejemplificación de plantilla de objeto computacional

La ejemplificación de un objeto computacional sólo puede ser efectuada por el entorno que está modelado como un módulo con el atributo *systemactivity*. Es preciso diferenciar dos fases. En la primera se crea el *ejemplar de módulo*. Esto puede hacerse utilizando dos formas genéricas del *enunciado-inic*. En una forma se pueden pasar parámetros al *ejemplar de módulo* que se crea. Dado que las *variables de módulo* que serán utilizadas en el *enunciado-inic* tienen que ser declaradas previamente, la creación dinámica de *ejemplares de módulo* no es posible en Estelle. En la segunda fase, hay que conectar los *puntos de interacción* adecuados a sus respectivos copartícipes.

A.4.2.6 Reglas de fallos

Puesto que las señales, como el depósito de invocación de operación, son modeladas en Estelle por mensajes enviados a través de un *punto de interacción*, un fallo de la infraestructura no se produce automáticamente si la interfaz no está vinculada. Un mensaje enviado a través de un *punto de interacción* que no está conectado se pierde, simplemente. Sin embargo, en una implementación de la interpretación Estelle, la pérdida de un mensaje por el hecho de que las interfaces no están vinculadas puede ser detectada e informada al *ejemplar de módulo* emisor como un fallo de la infraestructura. La creación dinámica de *puntos de interacción* o de *ejemplares de módulo* no es posible en Estelle. Siempre es necesario definir una matriz con variables del tipo respectivo. La ejemplificación de plantillas fracasará si todos los casos posibles ya existen.

A.4.2.7 Reglas de portabilidad

En el lenguaje Estelle es posible crear plantillas de acción para todas las acciones. Estelle satisface todos los requisitos de las reglas de portabilidad con las siguientes excepciones: Estelle no soporta acciones de bifurcación ni de unión, ni tampoco garantiza el ordenamiento y la entrega de anuncios.

A.4.2.8 Puntos de conformidad y puntos de referencia

Puntos de referencia son *puntos de interacción* externos de *módulos* Estelle. Existen métodos para derivar casos de prueba a partir de especificaciones Estelle. Sin embargo, la derivación de casos de prueba para especificaciones multimódulo es una cuestión que está aún sujeta a debate. No hay vinculación de lenguaje entre un lenguaje de especificación de interfaz normalizado y Estelle. No obstante, existe una vasta metodología para pruebas basadas en interacciones visibles en protocolos de comunicaciones.

SERIES DE RECOMENDACIONES DEL UIT-T

Serie A	Organización del trabajo del UIT-T
Serie B	Medios de expresión: definiciones, símbolos, clasificación
Serie C	Estadísticas generales de telecomunicaciones
Serie D	Principios generales de tarificación
Serie E	Explotación general de la red, servicio telefónico, explotación del servicio y factores humanos
Serie F	Servicios de telecomunicación no telefónicos
Serie G	Sistemas y medios de transmisión, sistemas y redes digitales
Serie H	Sistemas audiovisuales y multimedios
Serie I	Red digital de servicios integrados
Serie J	Redes de cable y transmisión de programas radiofónicos y televisivos, y de otras señales multimedios
Serie K	Protección contra las interferencias
Serie L	Construcción, instalación y protección de los cables y otros elementos de planta exterior
Serie M	RGT y mantenimiento de redes: sistemas de transmisión, circuitos telefónicos, telegrafía, facsímil y circuitos arrendados internacionales
Serie N	Mantenimiento: circuitos internacionales para transmisiones radiofónicas y de televisión
Serie O	Especificaciones de los aparatos de medida
Serie P	Calidad de transmisión telefónica, instalaciones telefónicas y redes locales
Serie Q	Conmutación y señalización
Serie R	Transmisión telegráfica
Serie S	Equipos terminales para servicios de telegrafía
Serie T	Terminales para servicios de telemática
Serie U	Conmutación telegráfica
Serie V	Comunicación de datos por la red telefónica
Serie X	**Redes de datos y comunicación entre sistemas abiertos**
Serie Y	Infraestructura mundial de la información y aspectos del protocolo Internet
Serie Z	Lenguajes y aspectos generales de soporte lógico para sistemas de telecomunicación

Ginebra, 2001