UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

# UIT-T Z.500

(05/97)

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

SERIE Z: LENGUAJES DE PROGRAMACIÓN

Métodos para validación y pruebas

---

# Marco de los métodos formales en las pruebas de conformidad

Recomendación UIT-T Z.500

(Anteriormente Recomendación del CCITT)

---

RECOMENDACIONES DE LA SERIE Z DEL UIT-T

**LENGUAJES DE PROGRAMACIÓN**

| | |
|---|---|
| TÉCNICAS DE DESCRIPCIÓN FORMAL | Z.100–Z.199 |
| Lenguaje de especificación y descripción (SDL) | Z.100–Z.109 |
| Aplicación de técnicas de descripción formal | Z.110–Z.119 |
| Gráficos de secuencias de mensajes | Z.120–Z.129 |
| LENGUAJES DE PROGRAMACIÓN | Z.200–Z.299 |
| CHILL: el lenguaje de alto nivel del UIT-T | Z.200–Z.209 |
| LENGUAJE HOMBRE-MÁQUINA | Z.300–Z.499 |
| Principios generales | Z.300–Z.309 |
| Sintaxis básica y procedimientos de diálogo | Z.310–Z.319 |
| LHM ampliado para terminales con pantalla de visualización | Z.320–Z.329 |
| Especificación de la interfaz hombre-máquina | Z.330–Z.399 |
| CALIDAD DE SOPORTES LÓGICOS DE TELECOMUNICACIONES | Z.400–Z.499 |
| **MÉTODOS PARA VALIDACIÓN Y PRUEBAS** | **Z.500–Z.599** |

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

**RECOMENDACIÓN UIT-T Z.500**

# MARCO DE LOS MÉTODOS FORMALES EN LAS PRUEBAS DE CONFORMIDAD

## Orígenes

La Recomendación UIT-T Z.500 ha sido preparada por la Comisión de Estudio 10 (1997-2000) del UIT-T y fue aprobada por el procedimiento de la Resolución N.° 1 de la CMNT el 6 de mayo de 1997.

## 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 N.° 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

## Introducción

Muchas especificaciones de protocolos y servicios se describen actualmente en notaciones formales denominadas técnicas de descripción formal (FDT, *formal description technique*). Son ejemplos de técnicas de descripción formal DL, LOTOS, Estelle y ASN.1. Existe también una notación formal para la especificación de sucesiones de pruebas: TTCN. Las técnicas de descripción formal ofrecen las siguientes ventajas:

– describen los formatos y comportamientos sin ninguna ambigüedad;

– proporcionan una base para una validación rigurosa, incluidas pruebas de conformidad.

La conformidad con una norma de protocolo o servicio de comunicación es un requisito previo para la correcta interoperabilidad de sistemas abiertos. Las pruebas de conformidad, es decir, la determinación, por medio de pruebas, de la conformidad de un producto con su especificación, son importantes para el desarrollo de los productos porque contribuyen a aumentar el nivel de confianza en una interoperabilidad correcta.

Esta Recomendación "Marco para métodos formales de pruebas de conformidad" (FMCT, *framework on formal methods in conformance testing*) define el significado de la conformidad cuando se utilizan métodos formales para la especificación de un protocolo o servicio de comunicación. Está también destinada a servir de guía para la generación de pruebas por medios informáticos.

La presente Recomendación define un marco para el uso de métodos formales en las pruebas de conformidad. Se pretende que los implementadores, probadores y especificadores que intervienen en las pruebas de conformidad la utilicen como una guía para la definición de la conformidad y el proceso de prueba de una implementación, con respecto a una especificación que se expresa como una descripción formal.

**Recomendación Z.500**

# MARCO DE LOS MÉTODOS FORMALES EN LAS PRUEBAS DE CONFORMIDAD

*(Ginebra, 1997)*

## 1 Alcance

Esta Recomendación es aplicable cuando existe una especificación formal de un protocolo o servicio de comunicación, a partir de la cual deberá desarrollarse una serie de pruebas de conformidad. Puede servir de guía tanto para el proceso manual, como para el desarrollo de herramientas destinadas a la generación de sucesiones de pruebas por medios informáticos.

La Recomendación define un marco y no prescribe ningún método particular de generación de casos de prueba, ni tampoco una determinada relación de conformidad entre una especificación formal y una implementación. Es suplementaria a la Recomendación UIT-T/ISO "Metodología y marco para las pruebas de conformidad" (CTMF, *conformance testing methodology and framework*) [ISO/CEI 9646], que es aplicable a una amplia gama de productos y especificaciones, incluidas especificaciones en lenguaje natural. FMCT interpreta conceptos de prueba de conformidad en un contexto formal.

## 2 Referencias normativas

Las siguientes Recomendaciones del UIT-T y otras referencias contienen disposiciones que, mediante su referencia en este texto, constituyen disposiciones de la presente Recomendación. Al efectuar esta publicación, estaban en vigor las ediciones indicadas. Todas las Recomendaciones y otras referencias son objeto de revisiones por lo que se preconiza que los usuarios de esta Recomendación investiguen la posibilidad de aplicar las ediciones más recientes de las Recomendaciones y otras referencias citadas a continuación. Se publica periódicamente una lista de las Recomendaciones UIT-T actualmente vigentes.

### 2.1 Pruebas de conformidad

NOTA – En lo sucesivo, en la presente Recomendación, el siguiente conjunto de referencias se designará por CTMF.

– Recomendación UIT-T X.290 (1995) (equivalente a ISO/CEI 9646-1:1994), *Metodología y marco de las pruebas de conformidad para interconexión de sistemas abiertos de las Recomendaciones sobre los protocolos para aplicaciones del UIT-T – Conceptos generales*.

– Recomendación UIT-T X.291 (1995) (equivalente a ISO/CEI 9646-2:1994), *Metodología y marco de las pruebas de conformidad para interconexión de sistemas abiertos de las Recomendaciones sobre los protocolos para aplicaciones del UIT-T – Especificación de sucesiones de pruebas abstractas*.

– Recomendación UIT-T X.292 del CCITT (1992) (equivalente a ISO/CEI 9646-3:1992), *Metodología y marco de las pruebas de conformidad para interconexión de sistemas abiertos de las Recomendaciones sobre los protocolos para aplicaciones del CCITT – Notación combinada arborescente y tabular*.

– Recomendación UIT-T X.293 (1995) (equivalente a ISO/CEI 9646-4: 1994), *Metodología y marco de las pruebas de conformidad para interconexión de sistemas abiertos de las Recomendaciones sobre los protocolos para aplicaciones del UIT-T – Realización de las pruebas*.

– Recomendación UIT-T X.294 (1995) (equivalente a ISO/CEI 9646-5:1994), *Metodología y marco de las pruebas de conformidad para interconexión de sistemas abiertos de las Recomendaciones sobre los protocolos para aplicaciones del UIT-T – Requisitos que deberán cumplir los laboratorios de pruebas y los clientes en el proceso de evaluación de conformidad*.

– Recomendación UIT-T X.295 (1995) (equivalente a ISO/CEI 9646-6:1994), *Metodología y marco de las pruebas de conformidad para interconexión de sistemas abiertos de las Recomendaciones sobre los protocolos para aplicaciones del UIT-T – Especificación de pruebas de perfil de protocolo*.

– Recomendación UIT-T X.296 (1995) (equivalente a ISO/CEI 9646-7:1995), *Metodología y marco de las pruebas de conformidad para interconexión de sistemas abiertos de las Recomendaciones sobre los protocolos para aplicaciones del UIT-T – Declaraciones de conformidad de implementación*.

## 2.2 Técnicas de descripción formal

– Recomendación UIT-T Z.100 (1993), *Lenguaje de especificación y descripción del CCITT*.

– Recomendación UIT-T Z.120 (1996), *Gráficos de secuencias de mensajes*.

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

– ISO/CEI 9074:1989, *Information processing systems – Open Systems Interconnection – Estelle: A formal description technique based on an extended state transition model*.

# 3 Definiciones

## 3.1 Términos tomados de otras normas conexas

NOTA – Aunque las siguientes definiciones figuran en la Recomendación UIT-T X.290 e ISO/CEI 9646-1, se reproducen en la presente Recomendación porque el significado de estos términos es importante para las interpretaciones formales en la presente Recomendación.

**3.1.1 método de prueba abstracta**: La definición de la manera de probar una implementación sometida a prueba, formulada en un nivel de abstracción apropiado para que la descripción sea independiente de toda realización particular de un medio de prueba, pero lo suficientemente detallada para permitir la especificación de pruebas para este método de prueba.

**3.1.2 sucesión de pruebas de conformidad**: Un conjunto completo de casos de prueba, posiblemente combinados en grupos de pruebas contenidos unos en otros, que se selecciona para realizar pruebas de conformidad dinámica para uno o más protocolos.

**3.1.3 prueba(s) de conformidad**: Prueba de la medida en que una implementación sometida a prueba es una implementación conforme.

**3.1.4 implementación conforme**: Una implementación sometida a prueba que satisface los requisitos de conformidad estática y de conformidad dinámica, de acuerdo con las capacidades enunciadas en el (los) enunciado(s) de conformidad de implementación.

**3.1.5 requisito de conformidad dinámica**: El conjunto de requisitos (y opciones) que determinan el comportamiento observable permitido por la especificación o especificaciones pertinentes en casos de comunicación.

**3.1.6 fracaso (veredicto de)**: Veredicto relativo a la prueba que se obtiene cuando el resultado observado de la prueba, o bien demuestra la no conformidad con respecto a (por lo menos) uno de los requisitos de conformidad en que se basa el propósito del caso de prueba, o bien contiene por lo menos un evento de prueba que no es válido, con respecto a la especificación o especificaciones pertinentes.

**3.1.7 enunciado de conformidad de implementación (ICS, *implementation conformance statement*)**: Un enunciado formulado por el suministrador de una implementación o sistema que pretende que dicha implementación o sistema es conforme con una determinada especificación, y que indica las capacidades que han sido implementadas. El enunciado de conformidad de implementación (ICS) puede adoptar varias formas: ICS de protocolo, ICS de perfil, ICS específico de protocolo, e ICS de objeto de información.

**3.1.8 implementación sometida a prueba (IUT, *implementation under test*)**: Una implementación de uno o más protocolos en una relación de usuario/proveedor adyacentes, que es la parte del sistema abierto real que se estudia mediante la prueba.

**3.1.9 información suplementaria de implementación para pruebas (IXIT, *implementation extra information for testing*)**: Un enunciado formulado por el suministrador o implementador de una IUT que contiene o hace referencia a todas las informaciones (además de la proporcionada en el ICS) relacionadas con la IUT y su entorno de prueba, y que permite al laboratorio de pruebas realizar una sucesión de pruebas apropiadas sobre la IUT. La IXIT puede adoptar varias formas: IXIT de protocolo, IXIT de perfil, IXIT específica de perfil, e IXIT de objeto de información, enunciado de implementación TMP.

**3.1.10 medio de pruebas (MOT, *means of testing*)**: La combinación de equipos y procedimientos que puede efectuar la derivación, selección, parametrización y ejecución de casos de prueba, de acuerdo con una serie de pruebas abstractas normalizada de referencia, y que puede producir un registro cronológico de conformidad.

**3.1.11 sucesión de pruebas abstractas parametrizadas**: Una sucesión de pruebas abstractas seleccionadas en la que a todos los parámetros pertinentes se les ha dado valores de acuerdo con uno o varios ICS e IXIT apropiados.

**3.1.12 éxito (veredicto de)**: Veredicto relativo a la prueba que se obtiene cuando el resultado observado de la prueba demuestra la conformidad con el requisito o requisitos de conformidad en que se basa el propósito del caso de prueba, sin que se haya detectado ningún evento de prueba no válido.

**3.1.13 punto de control y de observación (PCO*, point of control and observation*)**: En un entorno de pruebas, un punto en el que la aparición de eventos de prueba habrá de controlarse y observarse en la forma definida por el método de prueba abstracta.

**3.1.14 sucesión de pruebas abstractas normalizada de referencia (ATS*, reference standardized abstract test suite*)**: La sucesión de pruebas abstractas normalizada, para la que se ha realizado un medio de pruebas.

**3.1.15 requisito de conformidad estática**: Uno de los requisitos que especifican las limitaciones que se imponen a las combinaciones de capacidades implementadas permitidas en un sistema abierto real que se pretende que es conforme con una o varias especificaciones pertinentes.

**3.1.16 propósito de (la) prueba**: Una descripción, escrita en lenguaje ordinario, de un objetivo de prueba bien definido, basada en un solo requisito de conformidad o en un conjunto de requisitos de conformidad conexos, expresados en la especificación apropiada.

**3.1.17 veredicto de la prueba**: Un enunciado de "éxito", "fracaso", o "no concluyente", especificado en un caso de prueba abstracta, referente a la conformidad de una IUT con respecto al caso de prueba que se ejecuta.

## 3.2 Términos definidos en esta Recomendación

**3.2.1 compatibilidad**: véase 6.4.3.

**3.2.2 completa (sucesión de pruebas)**: véase 8.2.

**3.2.3 conformidad**: véase 6.4.

**3.2.4 pruebas de conformidad**: véase 8.2.

**3.2.5 conformidad dinámica**: véanse 6.4.2 y 6.4.3.

**3.2.6 exhaustiva (sucesión de pruebas)**: véase 8.2.

**3.2.7 modelo de fallo**: véase 8.4.1.

**3.2.8 cobertura de fallos**: véase 8.5.

**3.2.9 especificación formal**: véase 6.2.

**3.2.10 implementación**: véase 6.3.

**3.2.11 punto de acceso a la implementación**: véase 7.2.

**3.2.12 punto de acceso a la implementación, (modelo formal de)**: véase 7.3.

**3.2.13 opción de implementación**: véase 6.2.

**3.2.14 relación de implementación**: véase 6.4.2.

**3.2.15 implementación sometida a prueba**: véase 3.1.

**3.2.16 implementación sometida a prueba, (modelo formal de)**: véase 7.3.

**3.2.17 especificación instanciada**: véase 6.2.

**3.2.18 punto de interacción**: véase 7.3.

**3.2.19 especificación menos restrictiva**: véase 8.4.2.

**3.2.20 modelo de implementación**: véase 6.3.

**3.2.21 mutante**: véase 8.4.1.

**3.2.22 observación**: véase 7.4.1.

**3.2.23** **especificación parametrizada**: véase 6.2.

**3.2.24** **punto de control y observación**: véase 3.1.

**3.2.25** **punto de control y observación, (modelo formal de)**: véase 7.3.

**3.2.26** **relación de satisfacción**: véase 6.4.3.

**3.2.27** **sana (sucesión de pruebas)**: véase 8.2.

**3.2.28** **especificación**: véase 6.2.

**3.2.29** **conformidad estática**: véase 6.4.1.

**3.2.30** **arquitectura de prueba**: véase 7.2.

**3.2.31** **caso de prueba (modelo formal de)**: véase 7.3.

**3.2.32** **ejecución de caso de prueba**: véase 7.4.1.

**3.2.33** **contexto de prueba**: véase 7.2.

**3.2.34** **contexto de prueba, (modelo formal de)**: véase 7.3.

**3.2.35** **generación de pruebas**: véase 8.3.

**3.2.36** **propósito de la prueba**: véanse 3.1 y 7.4.2

**3.2.37** **propósito de la prueba, (modelo formal de)**: véase 7.4.2.

**3.2.38** **sucesión de pruebas (coste de la)**: véase 8.6.

**3.2.39** **sucesión de pruebas, (modelo formal de)**: véase 7.3.

**3.2.40** **reducción del tamaño de la serie de pruebas**: véase 8.4.

**3.2.41** **probador**: véase 7.2.

**3.2.42** **probador (modelo formal de)**: véase 7.3.

**3.2.43** **relación de implementación menos rigurosa**: véase 8.4.2.

## 4 Abreviaturas

En esta Recomendación se utilizan las siguientes siglas.

CTMF Metodología y marco para pruebas de conformidad (*conformance testing methodology and framework*)

FDT Técnica de descripción formal (*formal description technique*)

FMCT Métodos formales de pruebas de conformidad (*formal methods in conformance testing*)

IAP Punto de acceso a la implementación (*implementation access point*)

ICS Enunciado de conformidad de implementación (*implementation conformance statement*)

IUT Implementación sometida a prueba (*implementation under test*)

IXIT Información suplementaria de implementación para pruebas (*implementation extra information for testing*)

LOTOS Lenguaje de especificaciones con ordenación temporal (*language of temporal ordering specifications*)

LTS Sistema de transiciones etiquetadas (*labelled transition system*)

MSC Diagrama de secuencia de mensajes (*message sequence chart*)

PCO Punto de control y observación (*point of control and observation*)

SDL Lenguaje de especificación y descripción (*specification and description language*)

TTCN Notación combinada de árbol y tabla (sinónimo: notación combinada arborescente y tabular) (*tree and tabular combined notation*)

# 5 Conceptos matemáticos y convenios de notación

## 5.1 Conjuntos

Un subconjunto de un conjunto se designa por una letra mayúscula (por ejemplo *A*, *B*, *C*) o por varias letras mayúsculas (por ejemplo *SPECS*, *IMPLS*, *TESTS*). Los elementos de un conjunto se designan por letras minúsculas (por ejemplo *a*, *b*, *c*).

En esta Recomendación se utilizan las siguientes notaciones y operaciones sobre conjuntos:

| | |
|---|---|
| $\{a,b,c,...\}$ | El conjunto que contiene los elementos *a*, *b*, *c*, ...; el orden en que aparecen los elementos no es importante. |
| $\varnothing$ | El conjunto vacío, es decir, el conjunto que no tiene ningún elemento. |
| $a \in A$ | *a* es un elemento del conjunto *A*. |
| $\{a \in A \mid P(a)\}$ | El conjunto que contiene todos los elementos de *A* que satisfacen el predicado *P*. A veces se utiliza la expresión simplificada $\{a \mid P(a)\}$ cuando puede deducirse del contexto que se trata del conjunto *A*. |
| $A \subseteq B$ | *A* es un subconjunto de *B*, es decir, todos los elementos de *A* son también elementos de *B*. |
| $A = B$ | El conjunto *A* es igual al conjunto *B*, es decir, *A* es un subconjunto de *B* y *B* es un subconjunto de *A*. |
| $A \subset B$ | *A* es un subconjunto propio de *B*, es decir, *A* es un subconjunto de *B* y *A* no es igual a *B*. |
| $A \cap B$ | Intersección de *A* y *B*, es decir, el conjunto que contiene todos los elementos que pertenecen a *A* y a *B*. |
| $\bigcap_{i \in I} A_i$ | Intersección generalizada, es decir, la intersección de todos los conjuntos $A_i$: $A_1 \cap A_2 \ldots \cap A_n$, donde *n* es un número natural. |
| $A \cup B$ | Unión de *A* y *B*, es decir, el conjunto que contiene todos los elementos que pertenecen a *A*, o a *B*, o a ambos. |
| $\bigcup_{i \in I} A_i$ | Unión generalizada, es decir, la intersección de todos los conjuntos $A_i$: $A_1 \cup A_2 \ldots \cup A_n$, donde *n* es un número natural. |
| $A \times B$ | Producto cartesiano de *A* y *B*, que designa el conjunto de todos los pares ordenados (*a*, *b*) tales que $a \in A$ y $b \in B$. |
| $A_1 \times A_2 \times ... \times A_n$ | Producto cartesiano generalizado de $A_1$, $A_2$, ..., $A_n$, que designa el conjunto de todos los pares ordenados ($a_1$, $a_2$, …, $a_n$) tales que $a_1 \in A_1$, $a_2 \in A_2$, ..., $a_n \in A_n$. |
| $A - B$ | Diferencia de los conjuntos *A* y *B* (o diferencia conjunto *A* menos conjunto *B*), es decir, el conjunto que contiene todos los elementos de *A* que no pertenecen a *B*. |
| *Powerset(A)* | El superconjunto de *A*, es decir, el conjunto que contiene todos los subconjuntos de *A*. |
| $R_{\geq 0}$ | El conjunto de los números reales positivos, incluido el cero. |

El símbolo '/' superpuesto a un operador (de conjuntos) significa la negación del operador, por ejemplo, $a \notin A$ (*a* no es un elemento del conjunto *A*), $A \not\subset B$ (*A* no es un subconjunto propio de *B*), etc.

## 5.2 Notaciones lógicas

En esta Recomendación se utilizan las siguientes notaciones lógicas:

| | |
|---|---|
| $\neg p$ | No *p*, es decir, la negación de *p* |
| $p \wedge q$ | *p* y *q*, la conjunción de *p* y *q* |
| $p \vee q$ | *p* o *q*, la disyunción de *p* y *q* |
| $p \Rightarrow q$ | *p* implica *q*, también se dice no *p* o *q* |

| | |
|---|---|
| $p \Leftrightarrow q$ | $p$ es equivalente a $q$, es decir, $(p \Rightarrow q) \wedge (q \Rightarrow p)$ |
| $\forall\ a \in A$ | Para todos los elementos $a$ del conjunto $A$ |
| $\exists\ a \in A$ | Existe un elemento $a$ en el conjunto $A$ |

### 5.3 Relaciones

Las relaciones se designan por una abreviatura escrita con minúsculas y subrayada (por ejemplo rel).

Sean los conjuntos $A$ y $B$. Una relación binaria rel entre $A$ y $B$ es un subconjunto de su producto cartesiano: $\underline{\text{rel}} \subseteq A \times B$. El elemento $a \in A$ está relacionado con $b \in B$ si $(a,b) \in \underline{\text{rel}}$ (o, de otra forma, $a$ rel $b$). Por analogía, una relación $n$-aria es un subconjunto de $A_1 \times A_2 \times ... \times A_n$.

El *dominio* dom(rel) de la relación $\underline{\text{rel}} \subseteq A \times B$ se define como el conjunto que contiene todos los elementos de $A$ que están relacionados con algún $b \in B$, o sea:

$$\underline{\text{dom(rel)}} = \{a \in A \mid \exists\ b \in B : (a,b) \in \underline{\text{rel}}\}$$

Una relación (binaria) rel *sobre* $A$ es un subconjunto de $A \times A$.

### 5.4 Funciones

Los nombres de funciones se designan por una abreviatura escrita con minúsculas y subrayada (por ejemplo, func).

Una *función parcial* func es una relación entre dos conjuntos $A$ y $B$ que tienen la propiedad de que para cada $a \in A$ existe por lo menos un $b \in B$ tal que $\langle a,b \rangle \in \underline{\text{func}}$, es decir:

$$\forall a \in A : \forall b_1,b_2 \in B : (\langle a,b_1 \rangle \in \underline{\text{func}} \wedge \langle a,b_2 \rangle \in \underline{\text{func}}) \Rightarrow b_1 = b_2$$

Cuando se introduce una función, su firma se indica de la forma siguiente:

$$\underline{\text{func}} : A \rightarrow B$$

Una *función* (*total*) func : $A \rightarrow B$ es una función parcial que satisface dom(func) $= A$.

## 6 Significado de la conformidad

### 6.1 Introducción

La conformidad implica determinar si una implementación es una implementación válida de una especificación dada con respecto a cierta noción de lo correcto. Para formalizar el concepto de conformidad, las implementaciones se modelarán por objetos formales denominados modelos. La conformidad puede caracterizarse, sea por relaciones entre modelos de implementaciones y especificaciones, sea porque los modelos de implementaciones satisfagan requisitos especificados, o de ambos modos. En esta cláusula se define la conformidad, lo que incluye definiciones de especificaciones, implementaciones, modelos de implementaciones, relaciones de implementación y requisitos de conformidad.

### 6.2 Especificaciones

Una *especificación* (*formal*) prescribe el comportamiento de un sistema mediante una técnica de descripción formal (FDT). Si la FDT permite la utilización de parámetros, una especificación con parámetros formales se denomina una *especificación parametrizada*. Si los parámetros formales son instanciados con valores reales o si no hay parámetros formales, se dice que la especificación es una especificación instanciada.

El conjunto de especificaciones instanciadas se designa por *SPECS*. Una especificación parametrizada $s$ se considera como una función desde su espacio de parámetros $D_s$ (el conjunto de todos los valores reales posibles de los parámetros formales) a *SPECS*:

$$s : D_s \rightarrow SPECS$$

Las especificaciones suelen contener descripciones de características cuyo soporte es facultativo en un producto que implementa la norma. La posibilidad de decidir si determinadas características serán o no soportadas se conoce por opciones de implementación. Las especificaciones suelen contener tales opciones. Las opciones de implementación se representan por los parámetros formales de una especificación, esto es, la especificación está parametrizada en cuanto a sus opciones de implementación.

Una especificación que contiene *opciones de implementación* define un conjunto de especificaciones instanciadas: una especificación instanciada para cada posibilidad de elección entre las opciones. Toda especificación indicará claramente las combinaciones permitidas de soporte de opciones.

La información sobre las opciones de implementación deberá incluirse en los formularios de ICS de protocolo y de ICS de perfil. El ICS describe una determinada elección de opciones de implementación en el formulario de ICS. Una especificación parametrizada en su formulario de ICS es instanciada por un ICS. El ICS contendrá información suficiente para proporcionar valores reales para las opciones de implementación.

En lo sucesivo, en la presente Recomendación, el término *especificación* se utilizará con el significado de *especificación instanciada*.

NOTA – CTMF requiere que los ICS de protocolo/perfil se publiquen como anexo a la norma sobre protocolo/perfil.

**Ejemplo**

Supóngase que una especificación permite implementar diferentes niveles de soporte de características. Los niveles se numeran de 0 a 3. La especificación parametrizada normalizada puede representarse entonces por *s*(*nivel:* 0...3). El ICS de protocolo puede contener la información de que se pretende haber implementado *nivel = 2*. Esto significa que la especificación instanciada de la implementación sometida a prueba es, en efecto, *s*(2).

## 6.3 Implementaciones

Una implementación consiste en una combinación de componentes físicos (*hardware*) y/o lógicos (*software*). Las implementaciones tienen conectores físicos o interfaces de programación para la comunicación con su entorno o usuarios de extremo. El conjunto de implementaciones se designa por *IMPS*.

Una especificación es un objeto formal, mientras que una implementación es un objeto físico. Para formalizar el concepto de conformidad tiene que existir una relación entre estos dos objetos de naturaleza diferente. Las implementaciones no pueden someterse a un razonamiento formal, pues no son objetos formales. En consecuencia, no es posible definir directamente una relación formal entre implementaciones y especificaciones.

En el resto de esta Recomendación se supone que toda IUT $\in$ *IMPS* puede ser modelada por un elemento $m_{IUT}$ en un formalismo *MODS* (por ejemplo, sistemas de transiciones etiquetadas, máquinas de estados finitos). Este supuesto se denomina *supuesto para la prueba* (*test assumption*). Por probar ha de entenderse realizar ciertas acciones sobre la IUT con el fin de extraer una información tal que permita construir el modelo $m_{IUT}$ con suficiente detalle para tomar decisiones relativas a la conformidad.

NOTA 1 – Se supone solamente que es posible construir un modelo, y no que el modelo es conocido de antemano.

NOTA 2 – El formalismo *MODS* utilizado para modelar el comportamiento de una implementación puede ser igual al formalismo *SPECS* utilizado para la especificación.

Una implementación puede tener más de un modelo para una determinada elección de *MODS*. El supuesto para la prueba implica que no es posible diferenciar estos modelos mediante pruebas. Por consiguiente, basta con modelar una IUT mediante un $m_{IUT}$ de un solo elemento, del conjunto de modelos posibles *MODS*.

NOTA 3 – Se supone que la IUT puede modelarse con suficiente precisión, de modo que el modelo pueda representar la IUT con respecto a las propiedades prescritas por la especificación.

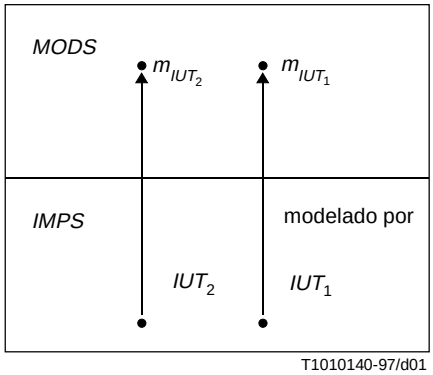


**Figura 1/Z.500 – Relación entre los elementos de *IMPS* y *MODS***

## 6.4 Conformidad de una implementación con una especificación formal

Existe conformidad entre una implementación y una especificación formal cuando la implementación es correcta con respecto a la especificación. La conformidad se define en dos partes:

- conformidad estática; y
- conformidad dinámica.

Una implementación es conforme con una especificación únicamente si hay conformidad estática y conformidad dinámica.

### 6.4.1 Conformidad estática

La *conformidad estática* implica la instanciación correcta de una especificación parametrizada. Una implementación sometida a prueba IUT, con el correspondiente enunciado de conformidad de implementación $ICS_{IUT}$ es conforme estáticamente con una especificación parametrizada $s : D_S \rightarrow SPECS$ si $ICS_{IUT}$ está contenido en el dominio de *s*, es decir, $ICS_{IUT} \in D_s$; esto significa que $s(ICS_{IUT})$ está definida. La verificación de la conformidad estática corresponde a la verificación del tipo del parámetro real $ICS_{IUT}$ con respecto al 'tipo' $D_s$.

La conformidad estática significa que $ICS_{IUT}$ es aceptado por la especificación parametrizada y, en consecuencia, que la combinación específica de opciones de implementación descritas en $ICS_{IUT}$ está permitida.

Las combinaciones y gamas admisibles de opciones de implementación en el $ICS_{IUT}$, es decir, la especificación del conjunto $D_s$, puede describirse por *requisitos de conformidad estática*. Un requisito de conformidad estática es un requisito que especifica las limitaciones impuestas a las gamas y las combinaciones de opciones implementadas y de capacidades permitidas en una implementación, por la especificación (véase CTMF, parte 1, 3.4.4).

NOTA – Si los requisitos de conformidad estática van más allá de la verificación normal del tipo de los valores de los parámetros reales en la FDT empleada, estos requisitos pueden expresarse en la parte de la descripción formal referente al comportamiento (véase el anexo A).

### 6.4.2 Conformidad dinámica

La *conformidad dinámica* comprende el comportamiento observable permitido de una implementación en instancias de comunicación, como se describe en la especificación. La conformidad dinámica entre una implementación y una especificación se caracteriza formalmente por una relación entre el modelo de la implementación y la especificación. Esta relación se denomina *relación de implementación*. Se designará por imp, donde imp tiene la firma:

$$\text{imp} \subseteq MODS \times SPECS$$

Una implementación IUT es conforme dinámicamente con una especificación instanciada *s* con respecto a una relación imp, si $m_{IUT}$ imp *s*. En este caso, $m_{IUT}$ es un modelo conforme de *s* con respecto a imp. Como tal relación de implementación, imp expresa la calidad de correcto entre la especificación *s* y la implementación IUT.

Una especificación instanciada puede tener varias implementaciones conformes. Para una especificación $s \in SPECS$ y una relación de implementación imp, el conjunto $M_s$ designa el conjunto de todos los modelos conformes en *MODS*, y viene dado por:

$$M_s = \{m \in MODS \mid m \text{ imp } s\}$$

NOTA – Para definir la conformidad tienen que existir previamente los siguientes objetos: una especificación $s \in SPECS$, una implementación IUT $\in IMPS$, una documentación de las posibilidades de elección entre las opciones $ICS_{IUT}$, y una relación de implementación. La relación de implementación no es universal; para diferentes áreas de aplicación pueden utilizarse diferentes relaciones de implementación. En 6.2 se presenta un ejemplo de relación de implementación para Estelle, LOTOS y SDL, y en el anexo A se dan más detalles.

La figura 2 ilustra cómo una especificación instanciada $s \in SPECS$ determina un conjunto de implementaciones conformes $I_s$. El conjunto $I_s$ designa el conjunto de implementaciones que pueden ser modeladas por modelos de $M_s$. En consecuencia, el conjunto $I_s$ es el conjunto de implementaciones correctas de la especificación *s*.

Suponiendo que *s* es una especificación de protocolo, elementos de $I_s$ podrán comunicar utilizando ese protocolo. Si *s* es la especificación de una pila completa de protocolos (un perfil) y se trata de una aplicación distribuida, elementos de $I_s$ podrán interoperar.

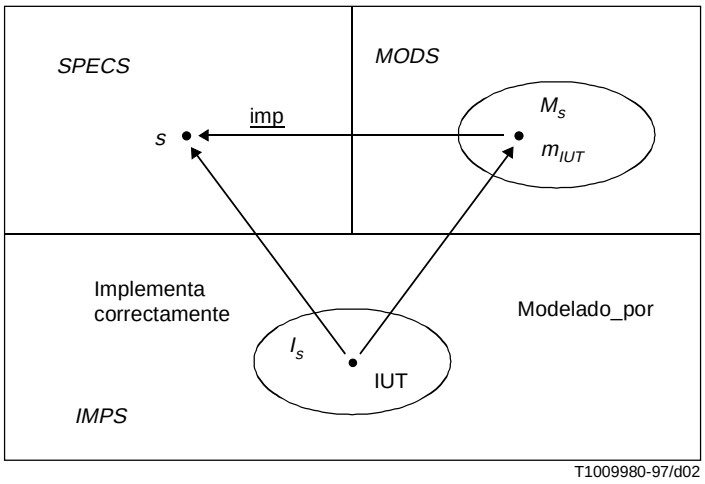


**Figura 2/Z.500 – Relaciones entre *IMPS*, *MODS* y *SPECS***

**Ejemplo**

Son ejemplos de relaciones de implementación frecuentemente utilizadas, cuando se decide que tanto *MODS* como *SPECS* se expresen en Estelle o SDL, la *equivalencia de los rastros* (*trace equivalence*) y el *orden previo de los rastros* (*trace preorder*). Si se requiere la equivalencia de los rastros, el conjunto de rastros de ejecución de la implementación será igual al conjunto de rastros de la especificación. En el caso de orden previo de los rastros, el primero de los conjuntos mencionados deberá estar contenido en el segundo. (Esto significa que sólo hay que implementar una parte del comportamiento especificado.)

Son ejemplos de relaciones de implementación frecuentemente utilizadas para LOTOS la *equivalencia de los fallos* (*failure equivalence*) y el *orden previo de los fallos* (*failure preorder*). Si se requiere el orden previo de los fallos, el conjunto de rastros de la implementación deberá estar contenido en el conjunto de rastros de la especificación, y la implementación no producirá atascos (*deadlocks*) que no estén especificados.

### 6.4.3 Requisitos de conformidad dinámica

En la cláusula 6, la conformidad se ha definido en forma abstracta por medio de una especificación instanciada junto con una relación de implementación entre los conjuntos *MODS* y *SPECS*. Por otra parte, en CTMF, la definición de la conformidad dinámica se basa en el concepto de una colección de requisitos de conformidad dinámica. Ambos planteos pueden utilizarse para especificar un comportamiento observable conforme. Pueden utilizarse por separado o en combinación. En 6.4.2, la conformidad se definió por medio de una especificación instanciada con una relación de implementación. En esta subcláusula se demuestra que la utilización de requisitos de conformidad dinámica es un posible refinamiento de esta definición, y que ambos planteos definen el mismo concepto con el mismo poder de expresión.

Un requisito de conformidad dinámica es un requisito que especifica el comportamiento observable que se permite en instancias de comportamiento (véase CTMF, parte 1, 3.4.3). Es una propiedad que debe ser satisfecha por (el modelo de) la implementación para que sea conforme.

Los requisitos de conformidad dinámica se expresan en un lenguaje de requisitos. *REQS* designa el conjunto de todos los requisitos que pueden expresarse en un lenguaje dado. En el planteo basado en requisitos, una especificación instanciada $s \in SPECS$ se expresa como un conjunto de requisitos $R_S \subseteq REQS$, que es $SPECS = Powerset(REQS)$. Un elemento de $r \in R_S$ representa un requisito individual de conformidad dinámica. En general, el conjunto $R_S$ puede ser infinito.

La conformidad dinámica entre una implementación y una especificación en el planteo basado en requisitos se caracteriza formalmente por una relación entre el modelo de la implementación y la especificación. Esta relación se denomina *relación de satisfacción*. Se designa por sat, donde sat tiene la firma:

$$\underline{\text{sat}} \subseteq MODS \times REQS$$

Una implementación IUT es conforme dinámicamente con la especificación $R_S \subseteq REQS$ si el modelo $m_{IUT}$ de IUT satisface todos los requisitos de conformidad en $R_S$. El conjunto $M_{R_S}$ de modelos de implementaciones conformes en el planteo basado en requisitos viene dado por:

$$M_{R_S} = \{m \in MODS \mid \forall r \in R_S : m \underline{\text{sat}}\ r\}$$

Para un determinado requisito de conformidad $r_i \in R_S$, el conjunto $M_{r_i}$ denota el conjunto de todos los modelos en *MODS* que satisfacen el requisito $r_i$, es decir, $M_{r_i} = \{m \in MODS \mid m \underline{\text{sat}}\ r_i\}$. La figura 3 muestra cómo la intersección de los conjuntos $M_{r_i}$ determina el conjunto $M_{R_S}$ de implementaciones conformes.

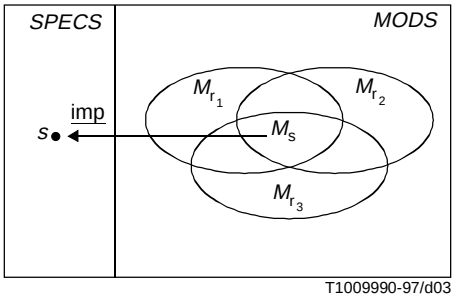


**Figura 3/Z.500 – Requisitos de conformidad e implementaciones conformes**\

### 6.4.4 Combinación de especificaciones

Especificaciones formales diferentes, no necesariamente expresadas en el mismo lenguaje formal, pueden combinarse para definir un conjunto único de modelos de implementaciones conformes. Si las especificaciones $s_1$, $s_2$, $s_3$, .. $s_n$ con las relaciones de implementación $\underline{\text{imp}}_1$, $\underline{\text{imp}}_2$, $\underline{\text{imp}}_3$, ... $\underline{\text{imp}}_n$, definidas para la misma clase de modelos de implementaciones *MODS*, definen el conjunto de implementaciones conformes $M_{s_1}$, $M_{s_2}$, $M_{s_n}$, respectivamente, entonces el conjunto de implementaciones conformes definidas por $s_1$, $s_2$, $s_3$, .. $s_n$ es $M_{s_1} \cap M_{s_2} \cap .. \cap M_{s_n}$.

Las especificaciones $s_1$, $s_2$, $s_3$, .. $s_n$ *son consistentes si esta intersección no está vacía*. La consistencia implica que la colección de especificaciones es implementable. Este requisito tiene que ser previamente satisfechos para que se puedan combinar las especificaciones.

Un caso particular de combinación de especificaciones se presenta cuando se combina una especificación de comportamiento instanciada *s* (con relación de implementación $\underline{\text{imp}}$) con una especificación de requisito *R* (con relación de satisfacción $\underline{\text{sat}}$). El requisito en *R* puede especificar requisitos adicionales de implementaciones conformes, o puede servir como una especificación alternativa para exactamente el mismo conjunto de implementaciones conformes. En este último caso se dice que *s* (con $\underline{\text{imp}}$) y *R* (con $\underline{\text{sat}}$) son *compatibles*:

$$\forall\ m \in MODS : m\ \underline{\text{imp}}\ s \Leftrightarrow \forall\ r \in R : m\ \underline{\text{sat}}\ r$$

# 7 Conceptos de pruebas

## 7.1 Introducción

Las pruebas constituyen un medio de extraer información sobre un sistema haciendo experimentos con el mismo. Los experimentos se realizan ejecutando casos de prueba. La ejecución de un caso de prueba conduce a una observación a partir de la cual se puede determinar si un sistema tiene o no cierta propiedad. En esta cláusula se definen los conceptos básicos empleados para describir la ejecución de casos de prueba (no se hace referencia a una noción de conformidad, ni a una propiedad que se prueba). Estos conceptos básicos incluyen el entorno en que se ejecutan los casos de prueba, el modelado formal de este entorno, casos de prueba, y sucesiones de pruebas, la ejecución de casos de prueba, las observaciones que pueden hacerse durante la ejecución de las pruebas, la interpretación de las observaciones y el propósito de la prueba en relación con un caso de prueba y con una sucesión de pruebas.

## 7.2 Arquitectura de prueba

La *arquitectura de prueba* es una descripción del entorno en que se prueba la IUT. Describe los aspectos importantes de la manera en que la IUT es incorporada en otros sistemas durante el proceso de prueba y en que la IUT comunica con el probador a través de los sistemas en que está incorporada. La figura 4 proporciona una vista esquemática de la arquitectura de prueba.

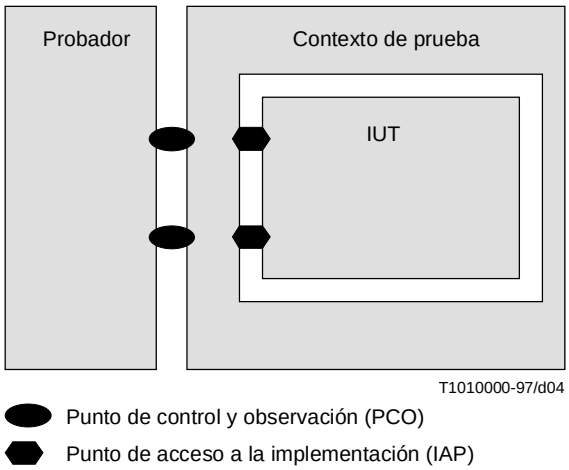


**Figura 4/Z.500 – Arquitectura de prueba**

Una *arquitectura de prueba* consiste en:

– un probador;

– una implementación sometida a prueba (IUT, *implementation under test*) (véase 3.1);

– un contexto de prueba;

– puntos de control y observación (PCO, *points of control and observation*) (véase 3.1);

– puntos de acceso a la implementación (IAP, *implementation access points*).

El *probador* es la implementación de una sucesión de pruebas (véase 7.3). Efectúa los experimentos ejecutando casos de prueba y observando los resultados. El probador comunica con el contexto de prueba a través de los PCO, e indirectamente con la IUT a través del contexto de prueba. El probador puede estar formado por diferentes componentes (por ejemplo, un probador inferior y un probador superior; véase CTMF).

El *contexto de prueba* es el sistema en el que se incorpora la IUT y a través del cual la IUT comunica con el probador. Relaciona eventos que se producen en los PCO, en la comunicación entre el contexto de prueba y el probador, con eventos que se producen en los IAP, en la comunicación entre el contexto de prueba y la IUT.

Un *punto de acceso a la implementación* (IAP) es un punto de interacción en la arquitectura de prueba, en el cual la IUT interactúa con su entorno, es decir, con el contexto de prueba, y a través del contexto de prueba (o sea, indirectamente) con el probador.

NOTA – En ciertos casos, los IAP y los PCO pueden coincidir.

## 7.3 Modelo formal de la arquitectura de prueba

Para poder razonar sobre el proceso de prueba en una configuración formal, las entidades de la arquitectura de prueba tienen que estar modeladas formalmente.

**Sucesión de pruebas** – La descripción formal del probador se da en una *sucesión de pruebas*. Una sucesión de pruebas especifica el conjunto completo de experimentos que habrán de ser ejecutados por el probador. El probador es el implementador de la sucesión de pruebas.

Los experimentos que constituyen una sucesión de pruebas se denominan *casos de prueba*. Cada caso de prueba especifica el comportamiento del probador en un experimento individual que prueba un aspecto de la IUT, y que conduce a una observación y un veredicto (véase 7.4.1).

El lenguaje formal en que se expresa un caso de prueba se denomina la *notación de prueba*. La notación de prueba se designa por *TESTS*. En consecuencia, una sucesión de pruebas, por el hecho de ser un conjunto de casos de prueba, es un elemento de *Powerset*(*TESTS*).

Se supone que las especificaciones de casos de prueba y de sucesiones de pruebas son implementadas correctamente en el probador, de acuerdo con la semántica de la notación de prueba. Este supuesto es razonable, ya que tales especificaciones son mucho menos complejas que las especificaciones de sistemas.

**Implementación sometida a prueba** – De acuerdo con el supuesto para la prueba (véase 6.3), la IUT se expresa formalmente por su modelo $m_{IUT} \in MODS$.

**Contexto de prueba** – El modelo formal de un contexto de prueba es una transformación del comportamiento tal como es observado en los IAP al comportamiento tal como es observado en los PCO. Se describe como una función *C* sobre *MODS*:

$$C : MODS \rightarrow MODS$$

En consecuencia, el comportamiento de la IUT tal como es observado en los PCO se expresa formalmente como $C(m_{IUT})$.

**Punto de control y observación y punto de acceso a la implementación** – Los puntos PCO e IAP se modelan formalmente como puntos de interacción en *MODS*. Las propiedades de estos puntos de interacción dependen de la forma en que las interacciones entre entidades están definidas en el formalismo *MODS* que se ha elegido. Deberá procederse con sumo cuidado para asegurar que el modelado de interacciones en *MODS* refleja fielmente las interacciones que se producen en los puntos PCO e IAP de la arquitectura de prueba.

## 7.4 Ejecución de pruebas

La ejecución (de pruebas) de una sucesión de pruebas $T \subseteq TESTS$ consiste en la ejecución del probador que implementa *T* en combinación con la IUT y el contexto de prueba para cada caso de prueba $t \in T$. La ejecución de un caso de prueba se denomina *ejecución de caso de prueba*. La ejecución de caso de prueba tiene por objetivo realizar experimentos con la IUT para investigar si la IUT responde o no correctamente a cierto comportamiento.

### 7.4.1 Ejecución de caso de prueba

Por ejecución de caso de prueba ha de entenderse la ejecución de la implementación de un caso de prueba $t \in TESTS$ en combinación con la IUT y el contexto de prueba. Durante la ejecución se hace una *observación*. Puede incluir un registro cronológico de las interacciones que se producen, un veredicto (preliminar), y todo lo que se considere de interés para la determinación del resultado de la ejecución del caso de prueba. Cuando la ejecución de un caso de prueba conduce a una observación – $\sigma \in OBS$ – el resultado se define por una asignación de veredicto $\underline{verd}_t$, que puede depender del caso de prueba $t \in T$:

$$\underline{verd}_t : OBS \rightarrow \{\text{éxito, no concluyente, fracaso}\}$$

Una implementación sometida a prueba *pasa* con éxito (*passes*) un caso de prueba *t* implementado (correctamente, en un contexto de prueba particular dado) únicamente si la ejecución de prueba de la IUT con *t* conduce a una observación $\sigma$ a la que se ha asignado el veredicto de éxito:

$$\text{IUT } \underline{passes}\ t \Leftrightarrow \underline{verd}_t\,(\sigma) = \text{éxito}$$

**Ejemplo**

El siguiente cuadro presenta ejemplos de observaciones. Cada fila del cuadro contiene resultados de mediciones. La secuencia de mediciones comprende la observación de una ejecución de prueba.

La medición de un evento se designa por el tipo del evento observado en (@) la ubicación de la interacción observada.

| Tiempo | Evento | Parámetros |
|---|---|---|
| 0003 | Conectar<br>@ PCO-1 | emisor = "1223"<br>dirección = "4545"<br>nivel = 3 |
| 0005 | Aceptar<br>@ PCO-1 | emisor = "4545"<br>dirección = "1223"<br>nivel = 2 |
| 0012 | Desconectar<br>@ PCO-1 | emisor = "1223"<br>dirección = "4545"<br>motivo = 5 |

### 7.4.2 Modelo formal de ejecución de caso de prueba

Para la interpretación del resultado de la ejecución de un caso de prueba hay que definir, por medio de una función, un modelo de ejecución de caso de prueba. La función exec calcula las observaciones para modelos de IUT ($m_{IUT}$) contenidos en un modelo de un contexto de prueba ($C$):

$$\underline{\text{exec}}: TESTS \times MODS \rightarrow OBS$$

La expresión $\underline{\text{exec}}(t,C(m_{IUT}))$ modela la observación, hecha por el caso de prueba $t$, de la IUT modelada como $m_{IUT}$ en el contexto de prueba $C$.

Si exec modela correctamente la ejecución de prueba, es decir, la ejecución de prueba conduce a una observación $\sigma$ únicamente si $\underline{\text{exec}}(t,C(m_{IUT})) = \sigma$, entonces, puede concluirse, a partir de la ejecución de prueba de una IUT con un caso de prueba $t$, que:

$$IUT \ \underline{\text{passes}} \ t \Leftrightarrow \underline{\text{verd}}_t \ (\underline{\text{exec}}(t,C(m_{IUT}))) = \text{éxito}$$

El subconjunto de *MODS* para el cual $\underline{verd}_t \ (\underline{exec}(t,C(m))) = $ éxito se denomina *propósito de prueba formal* $P_t$ :

$$P_t = \{m \in MODS \mid \underline{\text{verd}}_t \ (\underline{exec}(t,C(m))) = \text{éxito}\}$$

Por consiguiente, el objetivo de probar una IUT con un caso de prueba $t$ es determinar si el modelo de la IUT es un miembro de su propósito de prueba formal $P_t$, es decir, IUT pasa con éxito $t$ únicamente si $m_{IUT}$ es un elemento de $P_t$.

El propósito de prueba formal de un caso de prueba puede ir acompañado de una descripción en lenguaje ordinario. Esta descripción en lenguaje ordinario del objetivo de la prueba se denomina el *propósito de prueba* (*informal* o *en lenguaje natural*) (véase 3.1).

Un propósito de prueba formal como un subconjunto de *MODS* puede especificarse utilizando los mismos métodos empleados para especificar conjuntos de implementaciones conformes, es decir, o bien mediante una especificación de comportamiento instanciada junto con una relación de implementación, o bien por un requisito o conjunto de requisitos junto con una relación de satisfacción. Esta especificación formal que designa un propósito de prueba formal puede, a su vez, denominarse propósito de prueba formal.

**Ejemplo**

Un diagrama de secuencia de mensajes (MSC, *message sequence chart*) describe una secuencia de comportamiento. Junto con la relación de implementación *orden previo inverso de los rastros* (todas las secuencias de comportamiento del MSC deberán estar contenidas en la implementación) y el MSC especifican un propósito de prueba formal, a saber, el conjunto de modelos de implementaciones que contienen el comportamiento especificado por el MSC.

NOTA – El propósito de prueba formal $P_t$ está relacionado con el caso de prueba $t$. No está prescrito cuál de ellos se obtiene a partir del otro; esto es: dado un caso de prueba $t$, se puede calcular su propósito de prueba formal $P_t$, o, dado un propósito de prueba formal $p$ (por ejemplo, expresado como un requisito formal), se puede desarrollar un caso $t_p$ de prueba tal que $P_{t_p} = p$. Tal caso de prueba puede no existir.

### 7.4.3 Ejecución de sucesiones de pruebas

El proceso de ejecutar una sucesión de pruebas consiste en ejecutar consecutivamente cada uno de los casos de prueba que constituyen la sucesión de pruebas. A cada ejecución de estos casos de prueba se asigna un veredicto, como se indica en 7.4.1. Para que una sucesión de pruebas sea ejecutable tiene que ser finita. Sin embargo, cuando la ejecución de una sucesión de pruebas se modela con la función exec, no se requiere que la sucesión de pruebas sea finita.

Una implementación sometida a prueba IUT pasa con éxito una sucesión de pruebas $T \subseteq TESTS$ únicamente si pasa con éxito todos los casos de prueba que constituyen la sucesión de pruebas:

$$\text{IUT} \ \underline{\text{passes}} \ T \Leftrightarrow \forall t \in T : \text{IUT} \ \underline{\text{passes}} \ t$$

Si IUT pasa con éxito una sucesión de pruebas se deduce que el modelo de IUT está contenido en todos los propósitos de prueba formal de los casos de prueba que constituyen la sucesión de pruebas:

$$IUT \ \underline{\text{passes}} \ T \Leftrightarrow m_{IUT} \in \bigcap_{t \in T} P_t$$

El conjunto $\bigcap_{t \in T} P_t$ es el propósito de prueba formal de $T$. Se designa por $P_T$. Al igual que los propósitos de prueba formales de casos de prueba, el propósito de prueba formal puede ir acompañado de un propósito de prueba (informal o en lenguaje natural) para dar una descripción en lenguaje ordinario del objetivo de las pruebas con $T$.

# 8 Pruebas de conformidad

## 8.1 Introducción

En esta cláusula se utilizan los conceptos de prueba definidos en la cláusula 7 para probar la propiedad de conformidad definida en la cláusula 6. Se incluyen definiciones de pruebas de conformidad, generación de pruebas a partir de la especificación formal, cobertura de sucesión de pruebas y maneras de limitar el tamaño de la sucesión de pruebas.

## 8.2 Definición de pruebas de conformidad

Probar la conformidad es determinar por medio de pruebas si una implementación es conforme con su especificación. La prueba de conformidad tiene por finalidad recoger información para construir un modelo $m_{IUT}$ del comportamiento de la implementación IUT. Este modelo se utiliza para decidir si $m_{IUT}$ es un elemento del conjunto de modelos de implementaciones conformes, es decir, si $m_{IUT} \in M_S$.

En general no es posible evaluar con certeza absoluta la conformidad entre implementaciones y especificaciones debido, entre otras cosas, al no determinismo en la implementación, restricciones en cuanto a las posibilidades de observación y control de las implementaciones, y a la limitación de orden práctico de sólo poder ejecutar un número finito de pruebas. Una sucesión de pruebas $T$ puede tener las siguientes propiedades, que dependen de la relación entre $P_T$ y $M_S$.

**Exhaustiva (*exhaustive*)**: La sucesión de pruebas $T$ es exhaustiva si el conjunto $P_T$ de todos los modelos que pasan con éxito la sucesión de pruebas $T$ es un subconjunto del conjunto de modelos conformes $M_S$: $P_T \subseteq M_S$. Esto significa que todas las implementaciones que pasan con éxito la sucesión de pruebas son conformes.

**Sana (*sound*)**: La sucesión de pruebas $T$ es sana si el conjunto de modelos conformes $M_S$ es un subconjunto del conjunto $P_T$ de modelos que pasan con éxito la implementación IUT: $M_S \subseteq P_T$. Esto significa que todas las implementaciones que no pasan con éxito la sucesión de pruebas no son conformes.

**Completa (*complete*)**: La sucesión de pruebas $T$ es completa si es sana y exhaustiva, es decir, el conjunto de modelos conformes es igual al conjunto de modelos que pasan con éxito la implementación: $P_T = M_S$.

Si la sucesión de pruebas $T$ no es ni sana ni exhaustiva, las pruebas no permiten sacar ninguna conclusión en cuanto a la conformidad.

Lo ideal es que una sucesión de pruebas sea completa. En general, no es posible construir una sucesión de pruebas exhaustiva finita.

NOTA – No es probable encontrar en la práctica sucesiones de pruebas exhaustivas. La exhaustividad es un concepto útil para la argumentación teórica sobre un modelo (infinito) de sucesiones de pruebas, y como un medio de comparación para sucesiones de pruebas realizadas en la práctica (véase 8.5: Cobertura).

## 8.3 Generación de pruebas

En el proceso de generación de pruebas se crea una sucesión de pruebas (es decir, un conjunto de casos de prueba) a partir de una especificación formal. El uso de una técnica de descripción formal (FDT) es un requisito previo para la generación automática de pruebas.

La generación de pruebas se define por una función $\underline{\text{gen}}$ que proporciona una sucesión de pruebas para una especificación instanciada, dada una relación de implementación $\underline{\text{imp}}$ y un contexto de prueba $C$:

$$\underline{\text{gen}}^{C,\underline{\text{imp}}} : SPECS \rightarrow Powerset(TESTS)$$

La sucesión de pruebas generadas se expresa en una notación de prueba (véase 7.3: Notación de prueba).

Se requiere que la sucesión de pruebas generadas sea sana (véase 8.2). Por tanto, una sucesión de pruebas generadas tiene que cumplir la siguiente propiedad:

$$\forall m \in MODS, m \,\underline{\text{imp}}\, s \Rightarrow m \in P_T$$

donde

$$T = \underline{\text{gen}}^{C,\underline{\text{imp}}}(s)$$

La generación de pruebas, tal como se ha definido anteriormente, actúa sobre especificaciones instanciadas $\underline{\text{gen}}^{C,\underline{\text{imp}}} : SPECS \rightarrow Powerset(TESTS)$. Una sucesión de pruebas para una especificación $s : D_S \rightarrow SPECS$ e $ICS$ se obtienen mediante $\underline{\text{gen}}^{C,\underline{\text{imp}}}(s(ICS))$.

La instanciación de la especificación parametrizada $s : D_S \rightarrow SPECS$ con el *ICS* puede aplazarse hasta que haya finalizado la generación de pruebas. Esto significa que la generación de pruebas actúa sobre especificaciones parametrizadas, al generar una sucesión de pruebas parametrizadas sobre el *ICS*:

$$\underline{\text{pgen}}^{C,\underline{\text{imp}}} : (D \rightarrow SPECS) \rightarrow (D \rightarrow Powerset(TESTS))$$

donde *D* es el dominio de todos los enunciados de implementación posibles.

Una sucesión de pruebas para una especificación $s : D_S \rightarrow SPECS$ y un enunciado de conformidad de implementación *ICS* se obtienen mediante:

$$(\underline{\text{pgen}}^{C,\underline{\text{imp}}} : (s : D_S \rightarrow SPECS))(ICS)$$

Se requiere que las sucesiones de pruebas parametrizadas sean sanas para cada posible instanciación con un *ICS*.

## 8.4 Reducción del tamaño de las sucesiones de pruebas

Cada una de las sucesiones de pruebas utilizadas para pruebas de conformidad deberá ser sana (véase 8.3), lo que significa que cada ejecución de prueba que dé por resultado un veredicto de fracaso indica en realidad un error en la implementación sometida a prueba (véase 8.2). No se requiere la exhaustividad de las sucesiones de pruebas, lo que significa que no se detectarán necesariamente todos los errores en una implementación sometida a prueba (véase 8.2). Usualmente, el número de casos de prueba que se necesitarían para que una sucesión de pruebas fuera exhaustiva sería muy grande, incluso infinito, lo que implica que la prueba exhaustiva es irrealizable en la práctica. Para reducir el tamaño de una sucesión de pruebas con respecto al tamaño de una sucesión de pruebas exhaustiva ideal, de modo que pueda realizarse en la práctica, se han identificado diferentes estrategias de reducción del tamaño de las sucesiones de pruebas.

### 8.4.1 Modelo de fallo

Las estrategias de reducción del tamaño de las sucesiones de pruebas pueden presentarse por medio de un *modelo de fallo*. Un modelo de fallo *F* es un conjunto de modelos de implementaciones no conformes. Se expresa como un subconjunto de $MODS - M_S$:

$$F \subseteq MODS - M_S$$

Un modelo de fallo puede describirse como una modificación (un mutante) de la especificación $s \in SPECS$. Considérese que $\Delta_S \in SPECS$ contiene una modificación con respecto a la especificación original *s*; entonces, el modelo de fallo descrito por $\Delta_S$ es $(M_{\Delta_S} - M_S)$.

El mayor modelo de fallo es el conjunto de todas las implementaciones no conformes $MODS - M_S$.

**Ejemplo**

Supóngase que el conjunto de especificaciones *SPECS* y que el conjunto de modelos *MODS* son, ambos, iguales al conjunto de máquinas de estados finitos de entrada/salida. Para una especificación *s* dada pueden identificarse dos tipos de mutantes:

a) mutantes con fallos de transferencia, o sea, el estado final de las transiciones probadas puede ser diferente del estado final prescrito por la especificación *s*;

b) mutantes con fallos de salida, o sea, la salida observada para una determinada entrada puede no ser la prescrita por la especificación.

### 8.4.2 Estrategias de limitación del tamaño de las sucesiones de pruebas

Las estrategias de reducción del tamaño de las sucesiones de pruebas garantizan la obtención de una sucesión de pruebas sana, sea comenzando desde una sucesión de pruebas *T* sana pero probablemente demasiado grande, sea comenzando desde una especificación *s*, una relación de implementación imp, y una función de generación de pruebas $\underline{\text{gen}}^{C,\underline{\text{imp}}}$ que genera sucesiones de pruebas $\underline{\text{gen}}^{C,\underline{\text{imp}}}$ sanas, pero probablemente demasiado grandes. Se han identificado las siguientes estrategias:

1) Se toma cualquier subconjunto $T'$ de la sucesión de pruebas *T*. El modelo de fallo para el que la prueba corresponde al subconjunto $T'$ es el complemento de su propósito de prueba formal: $F = MODS - P_{T'}$.

2) Se mitiga (es decir, se hace menos restrictiva) la especificación *s* convirtiéndola en $s'$, de modo que $M_S \subset M_{S'}$ (es decir, se imponen menos requisitos a la implementación), y se genera una sucesión de pruebas sana para esta especificación menos restrictiva: $\underline{\text{gen}}^{C,\underline{\text{imp}}}(s')$ es una sucesión de pruebas sana para *s* con respecto a imp.

Un caso especial de una especificación de requisito $R \subseteq REQS$ es la especificación $s$ menos restrictiva $R' \subset R$ que consiste en una selección de requisitos de conformidad dinámica.

3) Se hace menos rigurosa la relación de implementación imp convirtiéndola en imp′ de modo que $\{m \in MODS \mid m \text{ imp } s\} \subset \{m \in MODS \mid m \text{ imp}' s\}$ (es decir, se permite que más especificaciones sean correctas), y se genera una sucesión de pruebas sana para esta relación de implementación menos rigurosa: $\text{gen}^{C,\text{imp}}(s)$ (es una sucesión sana para $s$ con respecto a imp′.

4) Se formula un supuesto de prueba (véase 6.3) más riguroso, es decir, en vez de suponer $m_{IUT} \in MODS$, se supone $m_{IUT} \in MODS'$ con $MODS' \subset MODS$.

5) Se especifica explícitamente un modelo de fallo $F \subset MODS - M_s$, por ejemplo considerando mutantes $\Delta_s$ de $s$, y se genera una sucesión de pruebas sana que detecta, por lo menos, todas las implementaciones no conformes en $F$, es decir $P_T \cap F = \emptyset$.

Se pueden combinar diferentes estrategias de reducción del tamaño de las sucesiones de pruebas para reducir el tamaño de una sucesión de pruebas.

**Ejemplo**

Si se tiene una especificación de una máquina de estados que se basa en la gama de los números enteros con, por consiguiente, un espacio (teóricamente) infinito de estados, $MODS = SPECS$, y si la relación de implementación requiere que todas las secuencias de comportamiento (*rastros*) de la especificación sean mostradas por la implementación, se puede aplicar la siguiente secuencia de estrategias de limitación del tamaño de las sucesiones de pruebas:

- se formula el supuesto de prueba más riguroso de que el número de estados de cualquier implementación está forzado a ser un número determinado $n$;
- se considera la relación de implementación menos rigurosa: todos los rastros de la especificación hasta la longitud $l$ tienen que ser mostrados por la relación de implementación;
- se considera la especificación menos restrictiva $s'$ de tal modo que $s'$ contenga solamente un subconjunto finito de los rastros de $s$.

Ahora, toda función de generación de pruebas que genere sucesiones de pruebas sanas para la relación de implementación menos rigurosa, genera, cuando se aplica a $s'$, una sucesión de pruebas sana para la especificación original con respecto a la relación de implementación original.

NOTA – Obsérvese la diferencia entre un mutante $\Delta_s$ y una especificación menos restrictiva $s'$ de $s$. Un mutante describe los fallos esperados para los cuales se generan pruebas (el modelo de fallo es un subconjunto del conjunto de implementaciones especificadas por $\Delta_s$) una especificación menos restrictiva especifica implementaciones que *no* están probadas (el modelo de fallo es un subconjunto del *complemento* de las implementaciones especificadas por $s'$).

## 8.5 Cobertura de fallos

La *cobertura de fallos* es una medida normalizada del grado en que una sucesión de pruebas se aproxima a la exhaustividad con respecto a un modelo de fallo $F$. Esto significa que expresa una cuantificación de la calidad de una sucesión de pruebas en base a sus capacidades de detección de errores. Una medida de cobertura puede utilizarse para comparar sucesiones de pruebas; una cobertura alta expresa una calidad alta.

La cobertura de fallo se expresa como una función con la siguiente firma:

$$\text{cov}_F : Powerset(TESTS) \rightarrow [0,1]$$

con el requisito de que la cobertura debe aumentar si se detectan más implementaciones erróneas en $F$ [$P_T$ es el propósito de prueba formal de la sucesión de pruebas $T$ (véase 7.4.1); $F - P_T$ es el subconjunto, del conjunto $F$, que contiene todas los modelos de implementación que fracasan con $T$]:

$$F - P_{T_1} \subseteq F - P_{T_2} \Rightarrow \text{cov}_F(T_1) \leq \text{cov}_F(T_2)$$

Si se omite $F$, se supone que éste es el conjunto de todas las implementaciones erróneas: $\text{cov}(T) = \text{cov}_{MODS - M_s}(T)$.

## 8.6 Coste de las sucesiones de pruebas

Los costes en que se incurre para generar, mantener, implementar y ejecutar una sucesión de pruebas se expresan como el coste de esa sucesión de pruebas. Esto significa que el coste expresa una cuantificación de los esfuerzos en forma de dinero, tiempo, etc. necesarios para una sucesión de pruebas. Una medida de coste puede utilizarse para comparar sucesiones de pruebas; un coste bajo expresa un nivel bajo de esfuerzos.

El coste puede expresarse como una función con la siguiente firma:

$$\underline{\text{cost}} : \mathit{Powerset}(\mathit{TESTS}) \rightarrow R_{\geq 0}$$

con el requisito de que el coste aumenta con el tamaño de la sucesión de pruebas:

$$T_1 \subseteq T_2 \Rightarrow \underline{\text{cost}}(T_1) \leq \underline{\text{cost}}(T_2)$$

El coste depende generalmente del número y de la longitud de los casos de prueba que constituyen la sucesión de pruebas.

# 9 Cumplimiento

## 9.1 Introducción

El término *cumplimiento (compliance)* se refiere al cumplimiento de los requisitos especificados en esta Recomendación. Este término se utiliza para distinguir entre el *cumplimiento* de (o con) esta Recomendación y la *conformidad* de una implementación sometida a prueba con una especificación (véase CTMF).

Esta Recomendación define un marco para el uso de métodos formales en las pruebas de conformidad. Dicho marco está destinado a los implementadores, probadores, y especificadores que intervienen en las pruebas de conformidad, que pueden usarlo como guía para la definición de la conformidad y el proceso de prueba de una implementación con respecto a una especificación que se expresa como una descripción formal.

El marco de esta Recomendación se presenta en un alto nivel de abstracción; por ejemplo, se hace abstracción de algoritmos específicos de generación de pruebas, e incluso de una técnica de descripción formal específica. El marco define terminología, conceptos abstractos, y requisitos mínimos relativos a estos conceptos, así como relaciones entre los conceptos. En consecuencia, para utilizar este marco se requiere la instanciación de estos conceptos mediante decisiones concretas en lo concerniente a la técnica de descripción formal (*SPECS*), el conjunto de modelos (*MODS*), la relación de implementación (imp), los algoritmos de generación de pruebas, etc., al mismo tiempo que se demuestran los requisitos impuestos a estos conceptos y las relaciones entre ellos.

En esta cláusula se indican explícitamente los requisitos que debe satisfacer dicha instanciación para cumplir con esta Recomendación. Asimismo, se identifican los supuestos en que se basa el proceso de prueba de conformidad, cuya validación está fuera del ámbito del proceso de prueba de conformidad propiamente dicho. Los requisitos para cada una de las cláusulas precedentes se indican en subcláusulas individuales de esta cláusula. En el anexo A figuran posibles instanciaciones para técnicas de descripción formal.

## 9.2 Cumplimiento con la cláusula 6: Significado de la conformidad

Para cumplir con la cláusula 6 de esta Recomendación los participantes en el proceso de prueba de conformidad deben identificar y llegar a un acuerdo sobre lo siguiente:

1) Una especificación parametrizada, designada por $s : D_s \rightarrow \mathit{SPECS}$ en la cláusula 6 – El espacio de parámetros $D_s$ identifica todas las posibilidades para las opciones de implementación de la especificación. *SPECS* es un formalismo de especificaciones instanciadas. Se supone que la especificación parametrizada es correcta y que ha sido validada, y sirva de punto de referencia para el proceso de prueba de conformidad.
2) Una implementación sometida a prueba IUT, junto con su correspondiente enunciado de conformidad de implementación, designado por $\mathit{ICS}_{IUT}$ en la cláusula 6 – La IUT con el correspondiente $\mathit{ICS}_{IUT}$ deberá ser estáticamente conforme con $s : D_s \rightarrow \mathit{SPECS}$, es decir, $\mathit{ICS}_{IUT} \in D_s$.
3) Un formalismo de modelado, designado por *MODS* en la cláusula 6, tal, que cualquier implementación posible considerada pueda suponerse modelada por un elemento de *MODS*.
4) Una relación de implementación, designada por imp en la cláusula 6, tal, que $\underline{\text{imp}} \subseteq \mathit{MODS} \times \mathit{SPECS}$, o, si $\mathit{SPECS} = \mathit{Powerset}(\mathit{REQS})$, siendo *REQS* un lenguaje lógico o de propiedades, una relación de satisfacción, designada por sat en la cláusula 6, tal, que $\underline{\text{sat}} \subseteq \mathit{MODS} \times \mathit{REQS}$.

Queda entendido que la IUT es conforme con $s : D_s \rightarrow \mathit{SPECS}$ únicamente si $\mathit{ICS}_{IUT} \in D_s$ y el modelo de la IUT, $m_{IUT}$, está asociado por la relación de implementación imp con $s(\mathit{ICS}_{IUT})$, es decir, $m_{IUT}$ imp $s(\mathit{ICS}_{IUT})$.

Si se utiliza más de una especificación para definir el conjunto de modelos de implementaciones, las especificaciones deberán ser consistentes (véase 6.4.4). La IUT es conforme con la colección de especificaciones únicamente si es conforme con cada una de ellas.

## 9.3 Cumplimiento con la cláusula 7: Conceptos de pruebas

Para cumplir con la cláusula 7 de esta Recomendación, los participantes en el proceso de prueba de conformidad deben identificar y llegar a un acuerdo sobre lo siguiente:

1) un probador, una implementación sometida a prueba IUT, un contexto de prueba, y sus interfaces mutuas denominadas puntos de acceso a la implementación IAP (entre la IUT y el contexto de prueba), puntos de control y observación PCO (entre la IUT y el probador);
2) una sucesión de pruebas, designada por $T$ en la cláusula 7, tal, que $T \subseteq TESTS$, siendo *TESTS* una notación de prueba;
3) una función $C : MODS \rightarrow MODS$ que modela el contexto de prueba;
4) para cada $t \in T$: una asignación de veredicto $\underline{verd}_t : OBS \rightarrow$ {éxito, fracaso}, donde *OBS* es un conjunto de observaciones posibles hechas durante la ejecución de la prueba;
5) una función de modelado de ejecución de prueba $\underline{exec} : TESTS \times MODS \rightarrow OBS$, que se supone que modela correctamente la ejecución de prueba de un caso de prueba con una IUT contenida en el contexto de prueba.

Queda entendido que la ejecución de prueba de $T$ sobre IUT pasa con éxito – IUT $\underline{passes}$ $T$ – únicamente si $m_{IUT} \in P_T$ (véase 7.4.2).

## 9.4 Cumplimiento con la cláusula 8: Pruebas de conformidad

Para cumplir con la cláusula 8 de esta Recomendación, los participantes en el proceso de prueba de conformidad deben identificar y llegar a un acuerdo sobre lo siguiente:

1) una función de generación de pruebas $\underline{gen}^{C,\underline{imp}} : SPECS \rightarrow Powerset(TESTS)$, tal, que las sucesiones de pruebas generadas son sanas para el dominio aplicable;
2) ninguna, una o más estrategias de reducción del tamaño de las sucesiones de pruebas (véase § 8.4.2);
3) facultativamente, una función de cobertura $\underline{cov}_F : Powerset(TESTS) \rightarrow [0,1]$ con respecto a un modelo de fallo $F \subseteq MODS$. Deberá demostrarse que la función $\underline{cov}_F$ es una función de cobertura (véase 8.5);
4) facultativamente, una función de coste $\underline{cost} : Powerset(TESTS) \rightarrow R_{\geq_0}$: deberá demostrarse que la función $\underline{cost}$ es una función de coste (véase 8.6).

Queda entendido que una sucesión de pruebas generada, cuando se ejecuta de modo que cumpla con 9.3, da una indicación sobre la conformidad de una IUT que cumple con 9.2. Si $\neg$ (IUT $\underline{passes}$ $T$), la IUT no es conforme; en cambio, si IUT $\underline{passes}$ $T$, la única conclusión que puede sacarse es que no hay una indicación de no conformidad.

# Anexo A

Este anexo explica una interpretación de terminología en las técnicas de descripción formal Estelle, LOTOS y SDL. No define nuevos términos que no se encuentren ya en la parte principal de esta Recomendación . Da una interpretación específica del lenguaje a los términos que conviene considerar dentro del contexto específico de una de las técnicas de descripción formal.

NOTA – Este anexo sólo presenta ejemplos sobre las definiciones generales y no tiene por finalidad proponer un estilo recomendado de especificación para las pruebas. El anexo es de naturaleza descriptiva más bien que prescriptiva, e informativa más bien que normativa.

## A.1 Especificaciones

### A.1.1 Introducción

La especificaciones son descripciones de comportamiento expresadas en una de las técnicas de la descripción formal (FDT) normalizadas: Estelle, LOTOS o SDL. El conjunto *SPECS* designa el conjunto de todas las especificaciones instanciadas descritas en una determinada técnica de descripción formal.

Se utiliza un ejemplo común para ilustrar la mayor parte de la terminología de esta Recomendación para las técnicas de descripción formal Estelle, LOTOS, y SDL. Esto permite la comparación entre las diferentes técnicas de descripción formal y su interrelación con la terminología de esta Recomendación.

El ejemplo común describe el comportamiento de una entidad iniciadora en un protocolo orientado a la conexión, es decir, un protocolo en el cual hay que establecer previamente una conexión para poder enviar datos. A continuación se da una descripción informal del protocolo. La descripción informal no tiene por finalidad dar una descripción precisa y completa del comportamiento del protocolo. Describe el comportamiento del protocolo en la medida que sea suficiente para su presentación como un ejemplo común en las diferentes técnicas de descripción formal.

**Ejemplo común**

El ejemplo común describe una entidad iniciadora en un protocolo orientado a una conexión previa. La figura A.1 muestra la arquitectura.

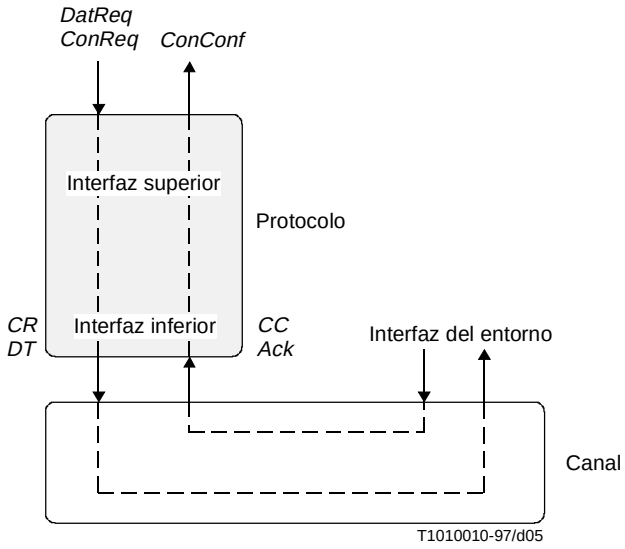


**Figura A.1/Z.500 – Arquitectura del ejemplo común**

Pueden distinguirse dos fases: la *fase de conexión* (en esta fase se establece la conexión) y la *fase de datos* (en esta fase el iniciador envía datos). Las siguientes limitaciones son aplicables a la fase de conexión:

– En el estado inicial, el iniciador espera que aparezca una petición de conexión en el nivel superior (*ConReq*).

– La petición de conexión en el nivel superior (*ConReq*) va seguida de una petición de conexión en el nivel inferior (*CR*).

– Tras una petición de conexión en el nivel inferior (*CR*) se recibe una confirmación de conexión en el nivel inferior (*CC*) de la entidad receptora.

– Una confirmación de conexión en el nivel inferior (*CC*) va seguida de una confirmación de conexión en el nivel superior (*ConConf*).

– En cualquier momento, la entidad iniciadora puede recibir una petición de desconexión (*DisReq*) en el nivel superior. Una petición de desconexión hace que el protocolo vuelva al estado inicial.

Se pasa a la fase de datos después de establecida una conexión. En la fase de datos, el iniciador puede enviar datos al receptor. Las siguientes limitaciones son aplicables a la fase de datos:

– La fase de datos sólo puede comenzar después de haberse enviado una confirmación de conexión en el nivel superior (*ConConf*).

– Una petición de datos en el nivel superior (*DatReq*) va seguida de datos en el nivel inferior (*DT*).

– Tras datos en el nivel inferior (*DT*) puede aparecer un acuse de recibo en el nivel inferior (*AK*).

– Solamente tras la recepción de un acuse de recibo en el nivel inferior (*AK*) puede producirse una nueva petición de datos en el nivel superior (*DatReq*).

– Durante la fase de datos, la entidad iniciadora puede recibir una petición de desconexión (*DisReq*) en el nivel superior, que hará retornar el protocolo al estado inicial.

El ejemplo común comprende también una descripción de una arquitectura de prueba, para ilustrar los conceptos de 7.2. La interfaz superior de la entidad iniciadora puede ser alcanzada directamente por el probador, mientras que la interfaz inferior sólo puede ser alcanzada a través de un *medio*. Se supone que este medio, o bien transmite correctamente los mensajes, o los pierde. No puede crear, duplicar ni reordenar mensajes.

### A.1.2 Especificaciones en Estelle

La figura A.2 muestra una descripción Estelle del ejemplo común presentado en A.1.1. Obsérvese que se emplea no-determinismo para modelar el hecho de que el canal puede perder mensajes.

```
specification Example;
  default individual queue;
  timescale seconds;

  channel ISAP(User, Protocol);
    by User    : ConReq;DatReq;Dis;
    by Protocol: ConConf;

  channel MSAP(Protocol, Channel_S);
    by Protocol : CR;DT;
    by Channel_S: CC;Ack;

  module Protocol_M
    ip U: ISAP(Protocol);
       L: MSAP(Protocol)
  end;

  body Protocol_B for Protocol_M

  state disconnected, wait, connected, sending;

  initialize to disconnected begin end;

  trans

  from disconnected to wait
    when U.ConReq
      begin output L.CR end;

  from wait to connected
    when L.CC
      begin output U.ConConf end;


  from connected to sending
    when U.DatReq
      begin output L.DT end;

  from sending to connected
    when L.Ack

      begin end;

  from wait, connected, sending to disconnected
    when U.Dis
      begin end;

  end;
```

```
module Channel_M
   ip E: MSAP(Protocol)
      L: MSAP(Channel_S);
end;

body Channel_B for Channel_M

state empty;

initialize to empty begin end;

trans

when L.CR
   begin output E.CR end;
   begin end;

when E.CC
   begin output L.CC end;
   begin end;

when L.DT
   begin output E.DT end;
   begin end;

when E.Ack
   begin output L.Ack end;
   begin end;

end;

modvar Protocol_I: Protocol_M;
       Channel_I: Channel_M;


initialize
   begin
      init Protocol_I with Protocol_B;
      init Channel_I with Channel_B;
      connect Protocol_I.L to
   Channel_I.L
   end;

end.
```

**Figura A.2/Z.500 – Especificación Estelle del ejemplo común**

Para definir la conformidad, en A.4.2 se definen relaciones de implementación. Sin embargo, no es conveniente definir relaciones de implementación significativas basadas en la descripción sintáctica de Estelle. La forma usual de definirlas es expresar primeramente la semántica de la especificación Estelle en algún tipo de modelo (matemático) más fundamental, y después definir relaciones de implementación basadas en ese modelo matemático.

La semántica de la especificación Estelle se expresa en cierto tipo de sistema de transiciones etiquetadas en que se distinga entre acciones de entrada y acciones de salida. Estos sistemas se designan por máquinas de estados de entrada-salida (IOSM, *input-output state machines*).

**Definición 1 (IOSM):** una máquina de estado de entrada-salida (IOSM) es un cuádruple (o tupla de 4) $M = \langle S, L, T, s_0 \rangle$ donde:

– $S$ es un conjunto finito, no vacío, de estados;

– $L$ es un conjunto finito, no vacío, de interacciones;

– $T \subseteq S \times (\{?,!\} \times L) \cup \{\tau\}) \times S$ es la relación de transición. Cada elemento de $T$ es una transición, de un estado de origen a un estado de destino. Esta transición está asociada a una acción observable (de entrada *?a* o de salida *!a*), o a la acción interna $\tau$.

– $s_0$ es el estado inicial de la IOSM.

El conjunto *SPECS* se elige de modo que sea el conjunto de la totalidad de la IOSM. La figura 3 muestra el modelo IOSM de la especificación Estelle del ejemplo común. Obsérvese que el modelo IOSM no permite indicar en qué punto de interacción se produce un determinado mensaje. Esta es la razón por la cual es necesario redenominar los mensajes para indicar diferentes puntos de interacción.

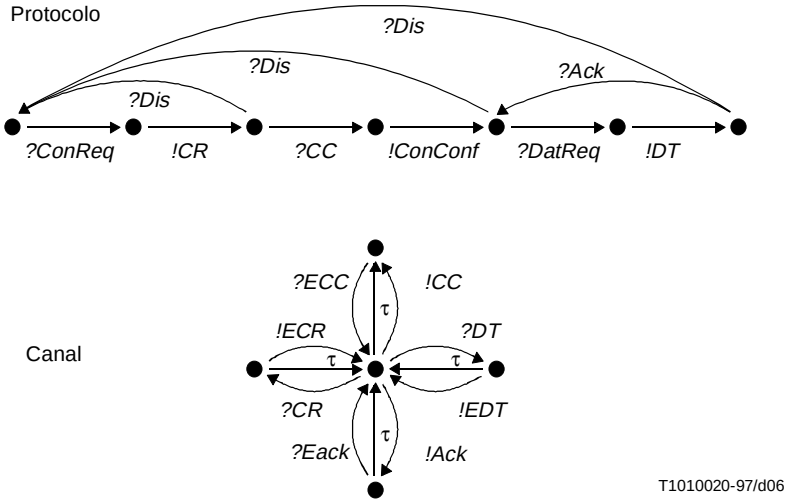


**Figura A.3/Z.500 – Modelo IOSM de la especificación Estelle de la figura A.2**

NOTA – No existen todavía trabajos sobre esta materia que hayan sido plenamente aceptados. La anterior presentación se basa en el trabajo descrito en [Phalippou 92] y [Phalippou 94], que es uno de los trabajos disponibles en la actualidad sobre esta materia. Podrían utilizarse otros modelos matemáticos para describir la semántica de la especificación Estelle de una manera apropiada para una definición de relaciones de implementación.

Se puede citar:

– Máquinas Mealy [Hopcroft 79] en las que se basan los métodos más tradicionales de generación de pruebas para sistemas de entrada y salida (W, UIO, DS, etc.).

– Un modelo derivado de máquinas Mealy denominado PNFSM (extensiones parcialmente especificadas y no determinísticas de máquinas Mealy) [Luo 93].

### A.1.3 Especificaciones en LOTOS

Una especificación LOTOS es una construcción de lenguaje que satisface la sintaxis descrita en [ISO 8807]. Una especificación LOTOS tiene por finalidad describir el comportamiento del sistema. Pueden obtenerse especificaciones de sistemas en diferentes niveles de abstracción prescindiendo de detalles internos del sistema que no ofrezcan interés.

Las siguientes restricciones son inherentes a las especificaciones LOTOS:

– LOTOS presupone que la comunicación es *síncrona*, es decir, que las entidades participan síncronamente en las interacciones.

– La comunicación es *atómica*, es decir, la comunicación entre las entidades participantes es correctamente para todas las entidades, o no lo es para ninguna de ellas.

– LOTOS sólo puede describir aspectos de tiempos relativos. Los aspectos de tiempos absolutos no pueden describirse.

La figura A.4 muestra una especificación LOTOS del ejemplo común descrito en A.1.1.

```
 (* specification : Initiator (BASIC LOTOS)

   This specification describes the Initiator Protocol *)

specification Initiator_Protocol[ConReq, ConConf, DatReq, CR, CC
                                 DT,Ack, Dis]                    : exit
behaviour
   Initiator [ConReq, ConConf, DatReq, CR, CC, DT, Ack, Dis]
where
   process Initiator [ConReq, ConConf, DatReq, CR, CC, DT, Ack, Dis] : exit :=
           Connection_Phase [ConReq, CR, CC, ConConf, Dis ] >>
           Data_Phase[ConReq, ConConf, DatReq, CR, CC, DT, Ack, Dis ]
   endproc (* Initiator *)

   process Connection_Phase [ConReq, CR, CC, ConConf, Dis] : exit :=
           ConReq ; CR; (CC ; ConConf ; exit
                         []
                         Dis; Connection_Phase [ConReq, CR, CC, ConConf, Dis])
   endproc (* Connection_Phase *)

   process Data_Phase [ConReq, ConConf, DatReq, CR, CC,
                       DT, Ack, Dis]                    : exit :=
           DatReq ; DT ; (Ack ; Data_Phase[ConReq, ConConf, DatReq, CR, CC,
                                           DT, Ack, Dis]
                          [] Dis; Initiator [ConReq, ConConf, DatReq, CR,
                                             CC, DT, Ack, Dis])
           []
           Dis; Initiator [ConReq, ConConf, DatReq, CR, CC, DT, Ack, Dis]
   endproc (* Initiator *)
endspec (* Initiator_Protocol *)
```

**Figura A.4/Z.500 – Especificación LOTOS del protocolo iniciador**

La semántica de las especificaciones LOTOS se expresa como sistemas de transiciones etiquetadas (LTS, *labelled transition systems*). Los sistemas de transiciones etiquetadas se representan por diagramas dirigidos cuyos bordes están etiquetados.

**Definición 2 (sistema de transiciones etiquetadas):** Un sistema de transiciones etiquetadas es un cuádruple (o tupla de 4) $TS = \langle S, L, T, s_0 \rangle$ en el cual:

– $S$ es un conjunto (contable), no vacío, de estados;

– $L$ es un conjunto (contable) de acciones observables;

– $T \subseteq S \times (L \cup \{\tau\}) \times S$ es la relación de transición, donde la etiqueta especial $\tau \notin L$ representa una acción no observable (o interna);

– $s_0 \in$ S es el estado inicial.

No se distingue entre acciones de entrada y acciones de salida ya que la comunicación es síncrona, y, por tanto, no existe la noción de entradas y salidas. La comunicación asíncrona se modela en LOTOS, para lo cual se modela modelando explícitamente el medio intermedio entre entidades que comunican (véase A.6.3).

El formalismo de los sistemas de transiciones etiquetadas es un formalismo muy fundamental de descripción de comportamiento. Muchos otros formalismos (por ejemplo FSM, EFSM, IOSM) pueden expresarse mediante sistemas de transiciones etiquetadas.

Para las especificaciones escritas en LOTOS, el conjunto *SPECS* es instanciado por el conjunto de todos los sistemas de transiciones etiquetadas *LTS* que pueden representar especificaciones LOTOS. En la figura A.5 se representa el sistema de transiciones etiquetadas que corresponde al ejemplo común descrito en A.1.1. El estado inicial se indica por un punto gris grueso.

### A.1.4 Especificaciones en SDL

En las figuras A.11, A.12, A.13, A.14 y A.15 se representa una especificación en SDL del ejemplo común descrito en A.1.1.

La conformidad dinámica se refiere solamente al comportamiento externo de un sistema. Una especificación SDL no expresa explícitamente el comportamiento externo de un sistema. Por tanto, para probar la conformidad dinámica se necesita una representación del comportamiento observable, derivada de la especificación SDL. Una representación adecuada del comportamiento externo para una especificación SDL se obtiene mediante un árbol de comunicación asíncrona (ACT, *asynchronous communication tree*) o mediante un sistema de transiciones etiquetadas (LTS). Para una definición formal de los ACT, véase [Hogrefe 88].

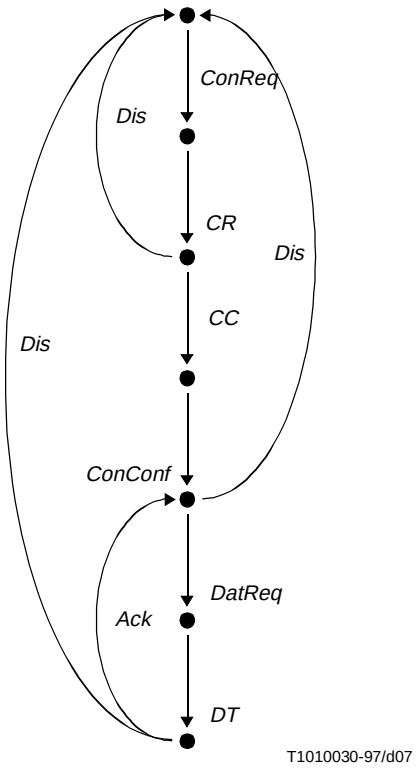


**Figura A.5/Z.500 – Modelo LTS de especificación LOTOS**

Por consiguiente, cuando se utiliza SDL para la especificación de sistemas, el conjunto *SPECS* es el conjunto de todos los ACT o LTS que pueden derivarse de cualquier especificación SDL. Con relación al ejemplo, en la figura A.6 se muestra una pequeña parte de una representación ACT del comportamiento observable. La representación ACT asocia a cada estado información sobre señales en los diferentes canales. En la figura A.6 sólo se indican canales que transmiten una señal en el estado actual. Si todos los canales del sistema están vacíos, esto se indica por $Q_i = \langle\,\rangle$. Sin embargo, la información de estado no se necesita cuando los casos de prueba se definen a partir de la representación ACT.

El término especificación SDL en este contexto se refiere solamente a *especificaciones instanciadas*. Las especificaciones SDL genéricas representan un número de especificaciones instanciadas diferentes, es decir, una especificación SDL genérica representa un conjunto de representaciones en el conjunto *SPECS*.

## A.2 Opciones de implementación y especificaciones instanciadas

### A.2.1 Introducción

El formulario ICS (IPf, *ICS proforma*) de una especificación de protocolo formal se describe por sus parámetros formales, y el ICS de una implementación por sus parámetros reales. Teóricamente, la parametrización de la especificación formal debe efectuarse de tal manera que el examen de conformidad estática (*static conformance review*) equivalga a una verificación de tipo en la FDT. Esto proporcionaría a estos conceptos una semántica bien estudiada.

Las opciones de implementación presentan dos aspectos:

– Variables, que se utilizan, por ejemplo, para designar opciones (por ejemplo, respuestas a preguntas de tipo Sí/No).

– Restricciones impuestas a los valores posibles de estas variables.

El primer aspecto es fácilmente modelado por cualquier FDT que permita la parametrización de especificaciones mediante valores. En cuanto al segundo aspecto, estas restricciones pueden ser comprobadas por un predicado (función/expresión booleana) que utilice parámetros. Sin embargo, es difícil modelar las restricciones de manera que reduzcan el examen de conformidad estática a una verificación de tipo.

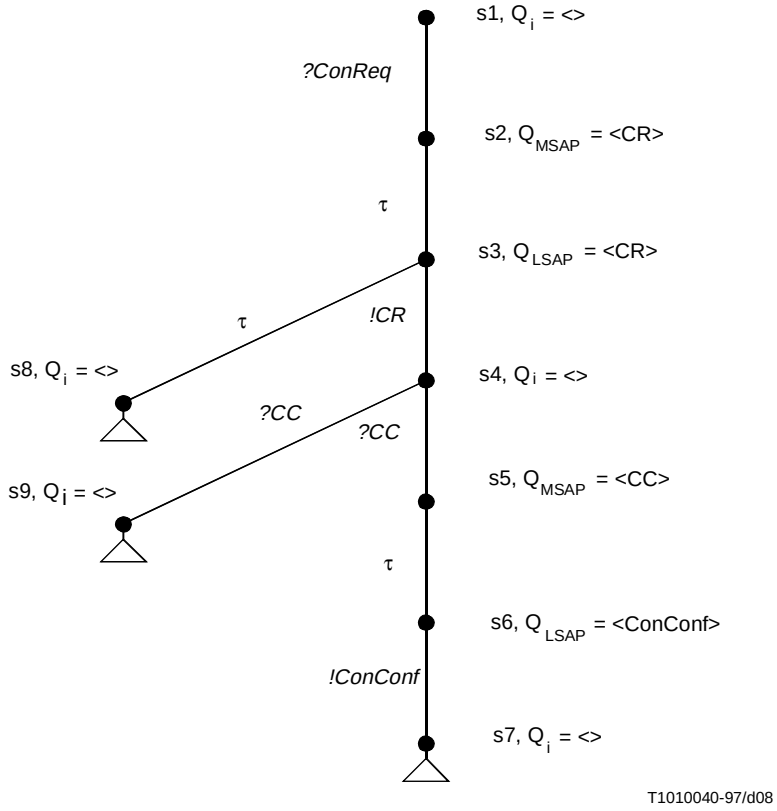


**Figura A.6/Z.500 – Representación ACT de una parte del comportamiento observable**

En la siguiente subcláusula se presentan ejemplos (limitados) que, por el momento, permiten llegar a la siguiente conclusión:

> El formulario IPf de una especificación de protocolo formal puede describirse como sus parámetros formales y el ICS de una implementación como sus parámetros reales, de manera que el examen de conformidad estática se describa por un predicado en la especificación formal.

A modo de ejemplo, se trata una versión simplificada del protocolo de transporte. En esta versión sólo se tiene en cuenta los requisitos relativos a clases soportadas. Los requisitos se toman del cuadro D.5.2 del formulario PICS del protocolo de transporte [ISO 8703] (limitados a las clases de 0 a 4).

Cada fila de este cuadro identifica una sola opción, cuyo valor puede ser Sí o No (columna titulada Soporte). La O en la columna titulada Estado significa que la capacidad es facultativa (opcional). El implementador puede llenar la casilla con Sí o No. La O.1 en las filas C0 y C2 significa que ambas son facultativas, pero que el implementador debe soportar al menos una de ellas (por lo que al menos una de ellas debe tener la respuesta Sí). El estado C0:O en la fila C1 significa que C1 es facultativa si C0 está soportada, y de lo contrario está prohibida.

**Cuadro A.1/Z.500 – Clases soportadas del protocolo de transporte**

| Índice | Clase | Referencias | Estado | Soporte |
|---|---|---|---|---|
| C0 | Clase 0 | 14 | O.1 | Sí No |
| C1 | Clase 1 | 14 | C0:O | Sí No |
| C2 | Clase 2 | 14 | O.1 | Sí No |
| C3 | Clase 3 | 14 | C2:O | Sí No |
| C4 | Clase 4 | 14 | C2:O | Sí No |

En resumen, este cuadro impone las siguientes restricciones:

– por lo menos la Clase 0 o la Clase 2 tiener que ser implementada;

– la Clase 1 requiere la Clase 0; y

– la Clase 3 y la Clase 4 requieren la Clase 2.

Las subcláusulas siguientes contienen especificaciones en las técnicas de descripción formal Estelle, LOTOS y SDL; en estas especificaciones sólo se muestran los principales elementos en lo que respecta al modelado del formulario PICS.

### A.2.2 Opciones de implementación y especificaciones instanciadas en Estelle

Estelle no proporciona construcciones predefinidas para modelar opciones, pero el ICS y el formulario PICS pueden modelarse utilizando construcciones de un lenguaje normalizado. Esto puede hacerse de varias maneras. En el ejemplo que sigue se muestra una forma de modelar opciones de implementación y especificaciones instanciadas en Estelle.

El formulario ICS se modela mediante un registro formado por campos booleanos que representan los campos de *índice* del formulario ICS. Las limitaciones impuestas a estos campos (es decir, los campos de *estado* del formulario ICS) son comprobados por una función denominada `Static_conformance_review` (examen de conformidad estática). Como la especificación parametrizada es modelada por un módulo Estelle parametrizado, la especificación instanciada se crea asignando un valor a este parámetro en la transición `initialize`.

```
specification Transport_Protocol;

type PICS_Proforma =
     record
       C0 : boolean; (* ref. 14 *)
       C1 : boolean; (* ref. 14 *)
       C2 : boolean; (* ref. 14 *)
       C3 : boolean; (* ref. 14 *)
       C4 : boolean; (* ref. 14 *)
     end;

var PICS : PICS_Proforma;

function Static_Conformance_Review (P : PICS_Proforma) : boolean;
        (* check static conformance requirements: *)
        (* (C0 or C2) and (C1 => C0) and ((C3 or C4) => C2) *)
        begin
          Static_Conformance_Review := ((P.C0 or P.C2) and
                                        (not(P.C1) or P.C0) and
                                        (not(P.C3 or P.C4) or P.C2));
        end;

procedure Get_PICS_Value (var P : PICS_Proforma);
primitive; (* get PICS value *)

(* Channel, Module header, Module body and Module variable definitions *)

initialize
   begin
     Get_PICS_Value(PICS);
     if (Static_Conformance_Review(PICS)) then
     begin
      (* instantiate appropriate modules *)
       ...
     end
   end;
end.
```

### A.2.3 Opciones de implementación y especificaciones instanciadas en LOTOS

Una especificación LOTOS puede parametrizarse explícitamente. Los tipos de datos ("sorts") utilizados en la lista de parámetros se definen globalmente. En el ejemplo, el único tipo de datos utilizado es bool, que se toma de la biblioteca normalizada de tipos predefinidos. Futuras mejoras de LOTOS pueden permitir que la descripción de los requisitos de conformidad estática se haga de manera que el examen de conformidad estática equivalga a una verificación del tipo.

```
SPECIFICATION Transport_Protocol [t,n] (c0,c1,c2,c3,c4 : bool) : NOEXIT

LIBRARY Boolean
ENDLIB

BEHAVIOUR ...

ENDSPEC (* Transport_Protocol *)
```

Se puede utilizar parámetros en expresiones booleanas, en las denominadas guardas, para seleccionar/restringir aún más el comportamiento.

```
SPECIFICATION Transport_Protocol [t,n] (c0,c1,c2,c3,c4 : bool) : NOEXIT

LIBRARY Boolean
ENDLIB

BEHAVIOUR TPEntity[t,n](c0,c1,c2,c3,c4)

WHERE

   PROCESS TPEntity [t,n] (c0,c1,c2,c3,c4 : bool) : NOEXIT :=
           ...
           [c4] -> Splitting[t,n]
           []
           [not(c4)] -> NoSplitting[t,n]
           ...
   ENDPROC (* TPEntity *)

ENDSPEC (* Transport_Protocol *)
```

El siguiente ejemplo muestra la utilización de requisitos de conformidad estática.

```
SPECIFICATION Transport_Protocol [t,n] (c0,c1,c2,c3,c4 : bool) : NOEXIT

LIBRARY Boolean
ENDLIB

BEHAVIOUR [ClassesConform(c0,c1,c2,c3,c4)] -> TPEntity[t,n](c0,c1,c2,c3,c4)

WHERE

   TYPE StaticConformance IS Boolean
   OPNS ClassesConform : bool, bool, bool, bool, bool -> bool
   EQNS FORALL c0, c1, c2, c3, c4 : bool
        OFSORT bool
        ClassesConform(c0,c1,c2,c3,c4) = (c0 or c2) and
                                         ((c3 or c4) implies c2) and
                                         (c1 implies c0)
   ENDTYPE (* StaticConformance *)

   PROCESS TPEntity [t,n] (c0,c1,c2,c3,c4 : bool) : NOEXIT :=
           ...
   ENDPROC (* TPEntity *)

ENDSPEC (* Transport_Protocol *)
```

NOTA – Esta forma de especificar los requisitos de conformidad estática tiene un inconveniente: si los parámetros reales no satisfacen ClassesConform, el comportamiento especificado es STOP. ¡Esto significa que una implementación que no hiciera nada sería conforme! Algo similar puede decirse también en cuanto a los ejemplos para las otras técnicas de descripción formal. Una solución a este problema consiste en utilizar uno de los mejoramientos de LOTOS, o sea, el concepto de módulo. A continuación se presenta una alternativa que, en realidad, no se recomienda, ya que produce especificaciones muy confusas. Se muestra solamente como un ejemplo de la forma en que puede contornearse el problema antes mencionado. Se utiliza un tipo de datos cuyos valores son todas las combinaciones posibles que son conformes estáticamente. Esta solución es también posible en las otras FDT.

```
SPECIFICATION Transport_Protocol [t,n] (classes : ValidClasses) : NOEXIT

LIBRARY Boolean
ENDLIB

TYPE  ValidClasses
SORTS ValidClasses
OPNS  only0, only01, only2, only23, only24, only234,
      only02, only023, only024, only0234,
      only012, only0123, only0124, only01234:        -> ValidClasses
ENDTYPE (* ValidClasses *)

BEHAVIOUR

  LET c0:bool = has0(classes),
      c1:bool = has1(classes),
      c2:bool = has2(classes),
      c3:bool = has3(classes),
      c4:bool = has4(classes)
  IN TPEntity[t,n](c0,c1,c2,c3,c4)
```

```
  WHERE
  TYPE SupportedClasses IS ValidClasses, Boolean
  OPNS has0, has1, has2, has3, has4 : ValidClasses -> bool
  EQNS OFSORT bool
       has0(only0)  = true
       has0(only01) = true
       ...
  ENDTYPE (* SupportedClasses *)

  ...

ENDSPEC (* Transport_Protocol *)
```

### A.2.4 Opciones de implementación y especificaciones instanciadas en SDL

En el marco FMCT las especificaciones pueden ser parametrizadas para tener en cuenta *opciones de implementación* diferentes. Para una implementación determinada, las opciones seleccionadas se definen en el documento ICS. Cuando estos valores de parámetros se aplican a la especificación parametrizada, el resultado es una *especificación instanciada*.

En el sistema genérico SDL se utilizan especificaciones para especificar sistemas parametrizados. Cuando se han aplicado los parámetros, la especificación instanciada se designa como *especificación de sistema específica*.

En el ejemplo que sigue, los parámetros se indican declarándolos `EXTERNAL`. Para verificar el ICS y determinar su conformidad estática se introduce un canal especial: el canal de error. Si la especificación es parametrizada por un ICS no conforme estáticamente, se envía el valor booleano `FALSE` por este canal, y de lo contrario no se envía nada por el canal.

```
SYSTEM TransportProtocol

SYNONYM C0 Boolean = EXTERNAL;
SYNONYM C1 Boolean = EXTERNAL;
SYNONYM C2 Boolean = EXTERNAL;
SYNONYM C3 Boolean = EXTERNAL;
SYNONYM C4 Boolean = EXTERNAL;

CHANNEL ErrorChannel
  /* Channel to convey FALSE if the static conf. review fails */
  FROM StatConfReview TO ENV
  WITH Boolean;
ENDCHANNEL;

BLOCK StatConfReview;
  /* This block contains only one process which performs
     the static conformance review */

  SIGNALROUTE ErrorRoute
  FROM StaticReviewer TO ENV
  WITH Boolean;

  CONNECT ErrorChannel AND ErrorRoute;

  PROCESS StaticReviewer (1, 1);
          DCL correct Boolean;

          START;
          TASK correct := (C0 OR C2) AND ((C3 OR C4) => C2) AND (C1 => C0);
          /* if correct = true, then the PICS conform */
          DECISION correct;
                   (TRUE) : STOP;          /* conforming: do nothing    */
                   (FALSE): OUTPUT FALSE; /* not conf: send FALSE via
                                             channel ErrorChannel       */
          ENDDECISION;
  ENDPROCESS;
ENDBLOCK;

BLOCK
  ...
ENDBLOCK;

...

ENDSYSTEM;
```

El ejemplo muestra cómo puede codificarse el ICS, en la especificación SDL, mediante la parametrización de la especificación SDL.

Pueden definirse especificaciones de sistemas que permitan la implementación de opciones diferentes. Puede haber dependencias entre las opciones de implementación. Las opciones de implementación y sus posibles dependencias se definen en el formulario ICS. En la [ISO 9646], las opciones reales implementadas en una implementación se especifican en el enunciado de conformidad de implementación. El ejemplo muestra cómo puede codificarse el ICS en la especificación SDL mediante la parametrización de la especificación SDL.

## A.3 Implementaciones y modelos de implementaciones

### A.3.1 Introducción

Una implementación es un objeto físico, no formal. Para todas las técnicas de descripción formal, el conjunto *IMPS* designa el conjunto de todas las implementaciones. Puesto que las implementaciones son objetos no formales, no es posible someterlas a un razonamiento matemático. Un modelo es una abstracción formal de la implementación, que es adecuada para someterla a un razonamiento matemático sobre la implementación.

Las pruebas tienen por finalidad construir un modelo de una implementación basado en las observaciones que pueden hacerse en experimentos realizados sobre la implementación. La clase de observaciones que pueden hacerse, sobre la implementación, después de la realización de los experimentos, determina el nivel de abstracción del modelo que se construye. En general, cuanto más pueda observarse de la implementación, tanto más detallado será el modelo que podrá construirse de esa implementación.

En la práctica, sólo se realiza un número limitado de experimentos sobre la implementación. Por cuestiones de realización en la práctica, no es posible modelar exactamente todos los aspectos de la implementación. El modelo obtenido mediante pruebas sólo debe reflejar el comportamiento de la implementación en los experimentos que se realizaron.

### A.3.2 Implementaciones y modelos de implementaciones en Estelle

Las implementaciones son objetos físicos, cuyo conjunto se designa por *IPMS*. Sin embargo, para los estudios de pruebas en un nivel teórico, las implementaciones deben ser modeladas por objetos matemáticos. Para modelar implementaciones es necesario tomar decisiones similares a las que se toman en el caso de especificaciones. Una posibilidad sería describir implementaciones en Estelle. Sin embargo, para el propósito de la prueba (con el fin de definir relaciones de implementación) es más conveniente representar las implementaciones por máquinas IOSM. En consecuencia, se dispone que el conjunto *MODS* sea el conjunto de todas las máquinas IOSM, el conjunto *IOSM*.

En la figura A.7, se presentan ejemplos de algunas implementaciones del protocolo de la figura A.3. Los ejemplos se han limitado a implementaciones que están "próximas" a la especificación, ya que, cuando se desarrollan productos, es improbable que se produzca algo muy diferente de lo esperado. Sin embargo, como se explicará en la subcláusula siguiente, algunas de estas implementaciones son conformes y otras no lo son.

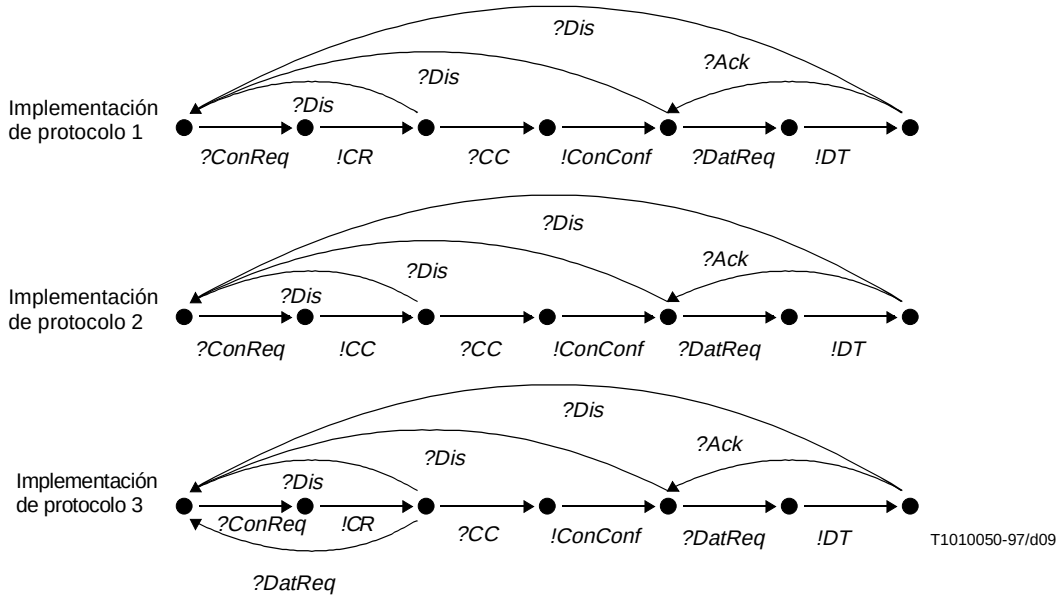


**Figura A.7/Z.500 – Modelos IOSM de implementaciones**

### A.3.3 Implementaciones y modelos de implementaciones en LOTOS

Un requisito previo para la construcción de un modelo de una implementación que, según se pretende, implementa un sistema descrito por una especificación LOTOS, es la existencia de un formalismo adecuado para expresar el modelo (por ejemplo, el formalismo debe poder expresar todos los detalles de interés para las pruebas). Asimismo, el formalismo deberá elegirse de manera que sea fácil determinar si la implementación satisface la especificación LOTOS [esto es, si el modelo de la implementación está relacionado con la especificación LOTOS por una relación de implementación; (véase A.4.3)].

Para describir modelos de implementaciones pueden utilizarse muchos formalismos (por ejemplo, sistemas de transiciones). No obstante, para las implementaciones de sistemas descritas por especificaciones LOTOS se suele elegir el formalismo de modelo *MODS* igual al conjunto de sistemas de transiciones etiquetadas *LTS*.

La figura A.8 representa algunos modelos de implementaciones del sistema descrito por la especificación de la figura A.5.

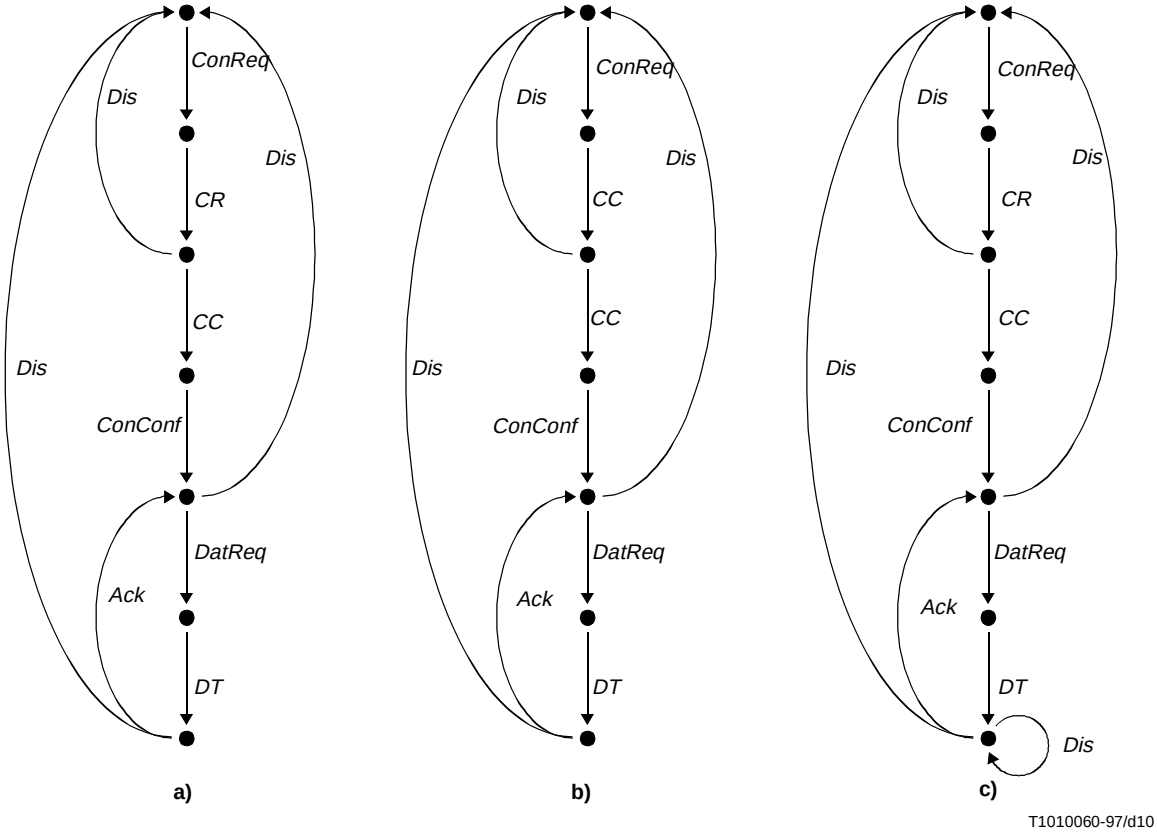


**Figura A.8/Z.500 – Modelos LTS de implementaciones**

### A.3.4 Implementaciones y modelos de implementaciones en SDL

Por implementaciones ha de entenderse sistemas ejecutables o físicos que implementan sistemas especificados en SDL. El conjunto *IMPS* contiene todas esas implementaciones.

En el proceso de pruebas de conformidad sólo puede aplicarse métodos formales cuando los elementos del proceso son elementos con una semántica formal. En tal situación, las implementaciones del conjunto *IMPS* no pueden utilizarse para esta finalidad. Sin embargo, se supone que para cada implementación en *IMPS* existe un modelo formal que representa las propiedades de la implementación. Este supuesto se designa por *supuesto para la prueba* y la aceptación del supuesto es básica para la totalidad del planteo formal.

Para cada implementación en *IPMS* existe por lo menos un modelo en el conjunto de modelos *MODS*. si existen más modelos para una implementación determinada estos modelos son similares hasta el punto de que no pueden ser diferenciados por la prueba. El conjunto *MODS* tiene también que constar de modelos que puedan servir de base para la definición de relaciones formalizadas con el conjunto de especificaciones *SPECS*. Puede también darse el caso de que los formalismos utilizados en los dos conjuntos sean los mismos, por ejemplo, ACT o LTS.

Para ilustrar los conceptos del marco formal se suponen las siguientes implementaciones del bloque protocolo del ejemplo de especificación. Las implementaciones se designan por $I_1$, $I_2$, e $I_3$:

- $I_1$es una implementación cuyo modelo es idéntico al modelo de la especificación. Esto significa que el bloque descripción mostrado en las figuras A.12 y A.13 pueden también considerarse como un modelo para esta implementación.
- También $I_2$ implementa el protocolo especificado en la figura A.13, con la salvedad de que el comportamiento puede expresarse en SDL como se muestra en la figura A.9. Esto es, la implementación envía una señal *DT* en vez de una señal *CR* cuando se recibe una petición de conexión.
- $I_3$ implementa de manera similar el protocolo definido en la especificación SDL. Además, en el estado *conectado* puede recibir una nueva petición de conexión y volver a iniciar la fase de conexión como se muestra en la figura A.9.

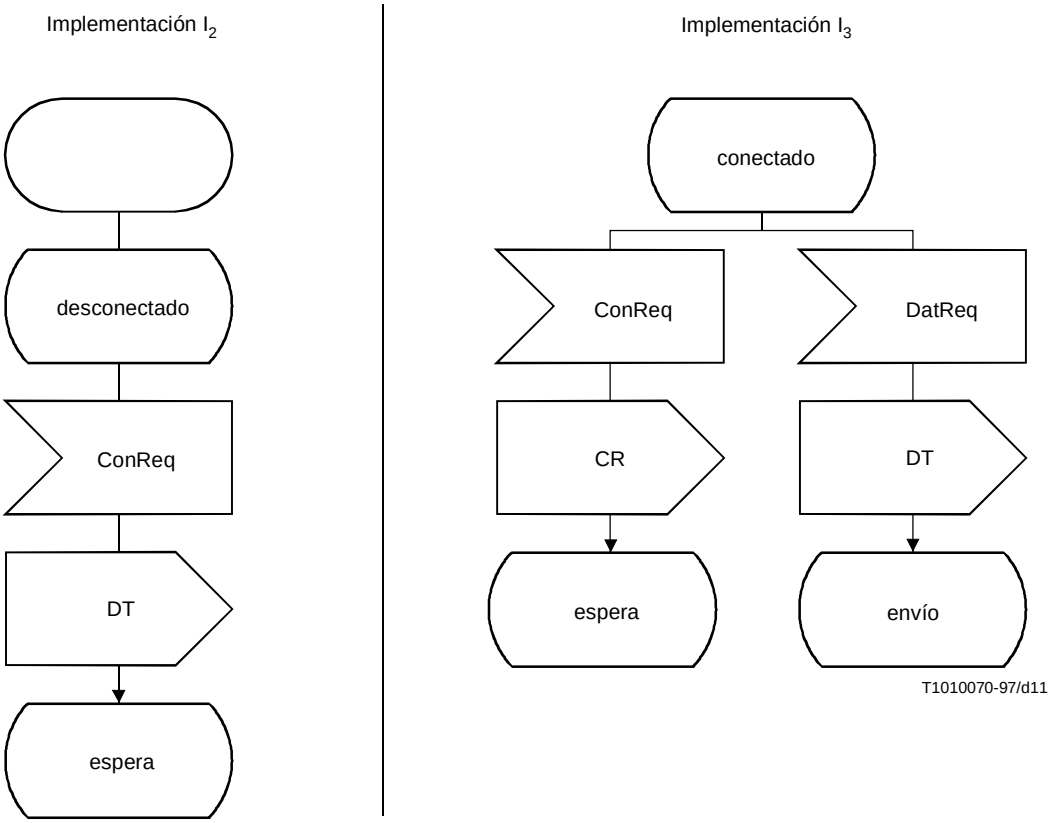


**Figura A.9/Z.500 – Comportamiento cambiado de las implementaciones $I_2$ e $I_3$**

## A.4 Conformidad mediante relaciones de implementación

### A.4.1 Introducción

Una implementación es conforme con su especificación si el modelo de la implementación está relacionado con la especificación por alguna relación de implementación imp. Como tal, la relación imp $\subseteq$ *MODS* $\times$ *SPECS* constituye la noción de corrección de una implementación con respecto a la especificación.

### A.4.2 Conformidad mediante relaciones de implementación en Estelle

Como se ha dicho anteriormente, la definición de las relaciones de implementación se basa en el modelo IOSM, y no en la descripción sintáctica de Estelle. Entre las relaciones de implementación aceptadas generalmente para los autómatas de entrada-salida está la equivalencia o inclusiones de los rastros, o una variante de la equivalencia de los rastros que se adapta para autómatas especificados de manera incompleta, denominada cuasiequivalencia (definida en PNFSM). Cuando la cuasiequivalencia se adapta al modelo IOSM, esta relación se convierte en la siguiente (las salidas permitidas después de un rastro dado $\sigma$ en IOSM S se designan por $O(\sigma, S)$]:

$$R_5(I,S) \text{ iff } (\forall\ \sigma \in Tr(S))\ (\sigma \in Tr(I) \Rightarrow (O(\sigma,I) = O(\sigma,S)))$$

En esta exposición se ha optado por tomar esta relación $R_5(I,S)$como un ejemplo de relación imp. De acuerdo con esta relación de implementación:

– la implementación 1 de la figura A.7 es conforme, como era de esperar, ya que esta implementación es igual a la especificación;

– la implementación 2 de la figura A.7 no es conforme: hay un error en el programa del protocolo, la entidad de protocolo envía una PDU de CC en lugar de una PDU de CR cuando recibe una *ConReq*;

– la implementación 3 de la figura A.7 es conforme. Esto podría ser sorprendente, pues esta implementación tiene un comportamiento extraño: si la entidad de protocolo recibe una *DatReq* antes de que se haya establecido el enlace, se interrumpe la conexión (esto produce el mismo efecto que una petición de desconexión). La conformidad puede explicarse de la manera siguiente: este comportamiento pertenece a la parte no especificada del protocolo, y la relación de implementación elegida expresa que la implementación puede actuar como desee en las partes no especificadas de la especificación. Sin embargo, se puede elegir una relación de implementación diferente con una interpretación diferente de las partes no especificadas.

### A.4.3 Conformidad mediante relaciones de implementación en LOTOS

Cuando se describen especificaciones en el lenguaje LOTOS, una relación de implementación es una relación entre elementos del conjunto *LTS* (el conjunto elegido para la instanciación de *MODS*) y elementos del conjunto *LTS* (el conjunto elegido para, con dicho conjunto, instanciar *SPECS*). Se han hecho numerosas propuestas de relaciones de implementación que proporcionan una comprensión intuitiva de la corrección. Estas propuestas no se examinan en esta Recomendación. La utilización de relaciones de implementación en LOTOS se ilustrará considerando un ejemplo particular de esa relación.

Una implementación aceptada generalmente para especificaciones LOTOS es la relación de *conformidad*, designada por conf. La relación conf capta la idea de que una implementación *I* es una implementación correcta para una especificación *S* únicamente si la implementación *I* no contiene atascos (*deadlocks*) que no fueron especificados en *S*.

A continuación se da una definición de la relación conf, donde $Tr(S)$ designa el conjunto de rastros de *S*, *L* designa el universo de acciones observables, *a* designa una acción observable y $\overset{\sigma}{\Rightarrow}$ designa la secuencia $\sigma$ de acciones observables:

$$I \ \underline{\text{conf}} \ S \ = \text{def} \quad \forall \sigma \in Tr(S), \forall A \subseteq L:$$
$$\text{si } \exists I' : I \overset{\sigma}{\Rightarrow} I' \text{ y } \forall a \in A : I' \overset{a}{\not\Rightarrow}$$
$$\text{entonces } \exists S' : S \overset{\sigma}{\Rightarrow} S' \text{ y } \forall a \in A : S' \overset{a}{\not\Rightarrow}$$

Si se elige la relación conf como la relación de implementación (es decir, como la noción de corrección), se puede comprobar cuáles de los modelos indicados en la figura A.8 son implementaciones correctas del modelo de especificación A.5.

– El modelo **a)** de la figura A.8 es un modelo correcto (de la implementación) de la especificación de acuerdo con la relación de implementación conf. En particular, el modelo iguala la especificación en todos los aspectos de su comportamiento. En consecuencia, toda implementación que satisfaga este modelo es una implementación correcta de la especificación descrita en la figura A.5 para la relación de implementación conf.

– Toda implementación que satisfaga el modelo **b)** de la figura A.8 no es conforme con la especificación de la figura A.5. Esto se debe a que la especificación puede realizar la secuencia de acciones *ConReq* · *CR*, mientras que la implementación no puede realizarla. Dado que la relación conf no permite que éste sea un modelo correcto de una implementación, la implementación propiamente dicha no es conforme.

– Toda implementación que satisfaga el modelo **c)** de la figura A.8 no es conforme con la especificación de la figura A.5. Aunque la implementación puede realizar la secuencia:

$$ConReq \cdot CR \cdot CC \cdot ConConf \cdot DatReq \cdot DT \cdot Dis \cdot Dis$$

y la especificación no puede realizarla, no se introducen en la implementación atascos que no hayan sido especificados. Por consiguiente, en presencia de la relación de implementación conf se puede llegar a la conclusión de que todas las implementaciones con el modelo **c)** de la figura A.8 son implementaciones correctas de la especificación.

### A.4.4 Conformidad mediante relaciones de implementación en SDL

Una *relación de implementación* define un criterio para la conformidad de una implementación. Una implementación es conforme cuando el par de modelos de la implementación y la especificación SDL es un miembro de la relación de implementación.

Varias de las relaciones de implementación definidas se basan en la existencia de secuencias similares de eventos en la implementación y la especificación. Las secuencias de eventos observables pueden derivarse de representaciones ACT y LTS de una especificación SDL. Una secuencia de eventos observables se designa por un *rastro* (*trace*) y un *conjunto de rastros Tr*(*S*) denota el conjunto de todos los rastros posibles de una especificación *S*. Un rastro del árbol ACT mostrado en la figura A.6 es ⟨*ConReq·CR·CC·ConConf*⟩.

Solo una clase muy Limitada de especificaciones SDL especifican un comportamiento que se expresa por un conjunto finito de rastros. Esto se debe a la propiedad de cola no limitada de los canales SDL y a las colas de entrada de procesos. Incluso para sistemas simples especificados en SDL, debe ser posible enviar, por el canal, cualquier número de señales del entorno al sistema. Así, en la especificación del ejemplo, debe ser posible enviar cualquier número de señales *ConReq* al sistema. Por tanto, en la práctica, no es posible una prueba exhaustiva del comportamiento observable de un sistema SDL. No obstante, los modelos de rastros son útiles para la definición de un requisito formal de conformidad del comportamiento dinámico de una implementación.

Una relación que suele utilizarse como relación de implementación es la relación de inclusión de rastro $\leq_{tr}$ (orden previo de los rastros). Dos modelos $S_1$ y $S_2$ satisfacen la relación $S_1 \leq_{tr} S_2$ únicamente si el conjunto de rastros de $S_1$ es un subconjunto del conjunto de rastros de $S_2$.

La relación de inclusión de rastros puede utilizarse como una relación de implementación en dos formas que dependen de la manera en que se interpretan los requisitos de la especificación. Si se supone que la especificación especifica el *comportamiento máximo permitido* de una implementación, el conjunto de rastros de una implementación conforme es un subconjunto del conjunto de rastros de la especificación. Para una implementación conforme *I* y una especificación *S*, esto se designa por $I \leq_{tr} S$ o $(I,S) \in \leq_{tr}$.

La otra solución para la interpretación de los requisitos especificados por una especificación SDL es suponer que ésta define el *comportamiento mínimo requerido* de una implementación. En este caso, el conjunto de rastros de una especificación *S* es un subconjunto de una implementación *I* conforme, $I \geq_{tr} S$.

La relación de implementación de comportamiento máximo permitido implica la imposición de un requisito muy poco restrictivo a la implementación conforme. Una implementación que no puede ejecutar ninguna acción externa es una implementación conforme de toda especificación, ya que el conjunto vacío es un subconjunto de todo conjunto de rastros. No es posible realizar pruebas para determinar la relación de implementación de comportamiento mínimo requerido, ya que la mayor parte de las especificaciones SDL tienen conjuntos de rastros infinitos.

En el cuadro A.2 se muestran las implementaciones que son conformes con respecto al ejemplo especificado (designado aquí por *S*) y las dos relaciones de implementación.

**Cuadro A.2/Z.500 – Visión general de la conformidad de $I_1$, $I_2$, e $I_3$ con respecto a diferentes relaciones de implementación**

| **(Impl, Spec) satisface** | $\leq_{tr}$ | $\geq_{tr}$ |
|---|---|---|
| $(I_1,S)$ | verdadero | verdadero |
| $(I_2,S)$ | falso | falso |
| $(I_3,S$ | falso | verdadero |

Dado que el conjunto de rastros de la implementación $I_1$ es idéntico al de la especificación, tanto el comportamiento máximo permitido como el comportamiento mínimo requerido son satisfechos para $I_1$. La implementación $I_2$ no satisface $\leq_{tr}$ ya que el rastro ⟨*ConReq·DT*⟩ no es un miembro del conjunto de rastros de la especificación. De manera similar, la relación de implementación $\geq_{tr}$ no se satisface tampoco, pues el conjunto de rastros de $I_2$ no incluye el rastro ⟨*ConReq·CR*⟩. Para la implementación $I_3$, la relación de implementación $\leq_{tr}$ no se satisface, ya que la implementación puede, por ejemplo, efectuar el rastro ⟨*ConReq·CR·CC·ConConf·ConReq·CR*⟩, que no es un rastro de la especificación. Dado que $I_3$ puede efectuar todos los rastros de la especificación, es conforme de acuerdo con la relación de implementación $\geq_{tr}$.

El modelo de una implementación no se conoce de antemano para la prueba de conformidad. Sólo puede obtenerse un conocimiento aproximado realizando experimentos sobre la implementación y observando las respuestas. Las especificaciones de sistema no determinísticas hacen imposible garantizar que un modelo de implementación es una descripción completa del comportamiento posible.

En el ejemplo de especificación puede que no sea posible determinar si una implementación puede realizar un rastro específico. Esto sucede con el rastro ⟨*ConReq*·*CR*⟩. El canal no fiable puede siempre descartar la señal *CR* de manera que nunca aparezca como un evento observable en el entorno. Sin embargo, no es posible, partiendo de un número de experimentos en los que no se ha observado la señal *CR*, determinar si el rastro ha sido implementado. Por tanto, las relaciones de implementación de orden previo de los rastros pueden proporcionar una base sana como un criterio de conformidad únicamente si se utiliza un supuesto adicional sobre la especificación y/o la implementación. Por ejemplo, puede suponerse que se proporciona información sobre el no-determinismo de la especificación se resuelve en la implementación.

Otra solución consiste en definir relaciones de implementación que tengan en cuenta las limitaciones de las pruebas de conformidad reales de sistemas especificados en SDL. Las relaciones **asco** y **aconf** son dos de estas relaciones de implementación propuestas en [TrVe 92] para relaciones de implementación que pueden utilizarse en sistemas SDL que comunican con el entorno a través de un solo canal.

Para satisfacer estas relaciones de implementación sólo se considera un subconjunto del conjunto de rastros. Este subconjunto incluye solamente los rastros que son mínimos con respecto a la adición de señales a un canal y la reordenación de eventos externos como consecuencia de retardos en los canales. Por ejemplo, el rastro ⟨*ConReq*·*CR*⟩ es un rastro mínimo con relación al cual se ha de probar la implementación. Ejemplos de rastros para los que este rastro es mínimo son ⟨*ConReq*·*CR*·*ConReq*⟩, en el que se agrega una señal de entrada y ⟨*ConReq*·*ConReq*·*CR*⟩, en que se desplaza la señal de entrada *ConReq* para situarla delante de la señal de salida *CR*.

La relación **asco** se satisface si para cada rastro mínimo en el que la última acción es una señal de salida, la implementación puede realizar solamente un subconjunto de las acciones posibles de acuerdo con la especificación. Para **aconf**, esto debe ser cierto solamente para todos los rastros de prefijo de los rastros mínimos. En el cuadro A.3 se muestra cómo las tres implementaciones son diferenciadas por las dos relaciones.

**Cuadro A.3/Z.500 – Conformidad de $I_1$, $I_2$, e $I_3$ con respecto a las relaciones de implementación asco y aconf**

| (Impl, Spec) satisface | asco | aconf |
|---|---|---|
| ($I_1$,*S*) | verdadero | verdadero |
| ($I_2$,*S*) | falso | falso |
| ($I_3$,*S*) | verdadero | verdadero |

Para la implementación $I_1$, se satisfacen ambas relaciones de implementación, ya que para cada rastro se puede observar el mismo conjunto de eventos para la especificación y la implementación. Para la implementación $I_2$, el rastro mínimo ⟨*ConReq*·*CR*⟩, donde el conjunto de eventos después de ⟨*ConReq*⟩ para la implementación es {*deadlock*,*DT*}, mientras que el conjunto de eventos de la especificación es {*deadlock*,*CR*}. Dado que el conjunto de eventos de la implementación no es un subconjunto de la especificación, $I_2$ es una implementación no conforme para ambas relaciones. Finalmente, puesto que la implementación se prueba solamente con respecto a rastros mínimos de la especificación, la clase de comportamiento adicional como en la implementación $I_3$ no se considera, e $I_3$ es una implementación conforme.

## A.5 Conformidad mediante requisitos

Otra posible forma de caracterizar la conformidad entre una implementación y una especificación es por medio de requisitos. Para la notación de los requisitos se utiliza un lenguaje de requisitos *REQS*. Una implementación es conforme con su especificación si las propiedades escritas en el lenguaje *REQS* que son válidas para la especificación son satisfechas por la implementación.

Por ejemplo, el lenguaje *REQS* puede ser instanciado por lenguajes lógicos, como Hennessy-Milner Logic, CTL. Las verificaciones para determinar si un modelo de una implementación satisface cierto requisito se denomina verificación de modelo. La verificación de modelo sólo puede realizarse cuando el modelo de una implementación está explícitamente disponible.

Como otro ejemplo, se introduce a continuación un lenguaje de propiedad simple. Este lenguaje de propiedad puede utilizarse para expresar propiedades de sistemas de transiciones etiquetadas.

Para $\sigma \in L^*$ y $A \subseteq L$, el conjunto *REQS* es instanciado por REQS = { **after** $\sigma$ **from** $A$ | $\sigma \in L^*$ and $A \subseteq L$ }. Un sistema de transiciones etiquetadas *T* satisface la propiedad **after** $\sigma$ **from** si $\exists T' : T \stackrel{\sigma}{\Rightarrow} T'$ y $\forall a \in A : T' \stackrel{a}{\Rightarrow}$.

En la figura A.8, los modelos **a)** y **c)** satisfacen la propiedad **after** *ConReq* **from** {*CR*}, pero el modelo **b)** no la satisface.

## A.6 Arquitectura de prueba

### A.6.1 Introducción

La arquitectura de prueba es una descripción del entorno en que se prueba la IUT. Es necesario modelar el entorno de la IUT porque su comportamiento puede no ser directamente observable por el probador debido a posibles limitaciones en la observabilidad impuestas, por ejemplo, por un medio de comunicación intermedio entre el probador y la IUT.

Cada FDT tiene facilidades para modelar el entorno de una IUT.

### A.6.2 Arquitectura de prueba en Estelle

La arquitectura de prueba se describe en Estelle mediante las siguientes construcciones:

– Se utilizan construcciones *módulo* para describir los componentes (probador, IUT, contexto de prueba).

– Las construcciones *cuerpo* describen el comportamiento de los diversos componentes.

– Los *puntos de interacción* representan los PCO y los IAP.

Como los módulos Estelle están enlazados por canales que tienen una semántica de cola FIFO (*first in first out*, es decir, basada en el principio de "primero en entrar, primero en salir"), estas opciones de modelado son compatibles con la semántica de los PCO, descrita por ISO 9646 (y que corresponde a la semántica de los PCO en el lenguaje TTCN).

Este método proporciona un modelado explícito del contexto de prueba como un componente, por sí. Se ha tratado de representar el contexto de prueba de una manera más próxima a la descrita en el cuerpo principal de la Recomendación, es decir, como una función de transformación sobre el conjunto *MODS* (en el presente caso, sobre el conjunto IOSM) (véase, por ejemplo [Phalippou 92]).

### A.6.3 Arquitectura de prueba en LOTOS

LOTOS no proporciona construcciones *especiales* para modelar el comportamiento de entornos de sistemas. En lugar de esto, el entorno se especifica simplemente como cualquier otro proceso. Dentro de los entornos, no se distingue entre modelado de entornos y modelado de sistemas.

Dado que la comunicación en LOTOS es síncrona, la comunicación asíncrona se modela modelando el medio de comunicación intermedio (por ejemplo, colas FIFO). En la especificación que sigue se presenta una especificación LOTOS básica que comunica por medio de una cola. En este ejemplo, todas las entradas del sistema que aparecen en la puerta a tienen que atravesar una cola fiable. La comunicación entre la cola y el sistema está oculta. Obsérvese que esta cola no es una cola FIFO (es decir, no se basa en el principio de "primero en entrar, primero en salir").

```
specification QueueCommunication[a, b, x, y] : noexit
behaviour
  hide ia in System [ia, b, x, y] |[ia]| Queue [ia, a]
where
  process System [a, b, x, y] : noexit:=
  a ;  x ;  stop [] b ;  (x ;  stop [] y ;  stop )
  endproc (* System *)

  process Queue[ia, a] : noexit:=
  a ; (ia ; stop ||| Queue [ia, a])
  endproc (* Queue *)
endspec (* QueueCommunication *)
```

En LOTOS, el probador, la IUT y el contexto de prueba se modelan por procesos LOTOS. Los PCO y los IAP se modelan por puertas (es decir, por construcciones en LOTOS para modelar puntos de interacción).

### A.6.4 Arquitectura de prueba en SDL

El único requisito que una especificación SDL impone al entorno de un sistema es que obedezca las constricciones prescritas en la especificación del sistema. Por tanto, para modelar las propiedades de un entorno en que ha de probarse una implementación, el contexto tendrá que especificarse como parte de la especificación del sistema. En el ejemplo común, el medio (*canal*) no fiable puede considerarse, como tal, un contexto de prueba para el protocolo.

Las entidades de la arquitectura de prueba se definen en una especificación SDL utilizando las mismas construcciones que las empleadas en la especificación de sistema original. El probador, el contexto de prueba y la IUT pueden especificarse utilizando la construcción *bloque*. El comportamiento de las entidades se define por *procesos* de los bloques. La comunicación entre las entidades (los bloques) de la arquitectura de prueba se modela por canales. Puesto que los canales en SDL son colas FIFO, pueden utilizarse para modelar puntos PCO de acuerdo con [ISO 9646]. También se utilizan canales para especificar puntos IAP.

Con referencia al ejemplo común, las entidades de la arquitectura de prueba pueden identificarse como sigue en el diagrama de sistema de la figura A.13. El bloque *Protocol_M* especifica la IUT, y las propiedades del contexto de prueba se especifican por el bloque *Medium_M*. Los PCO de la arquitectura de prueba son los canales *ISAP* y *LSAP* mientras que los IAP son los canales *ISAP* y *MSAP*. En este ejemplo, tanto la IUT como el contexto de prueba se especifican por un solo bloque; en general, se puede utilizar un conjunto de bloques para especificar cualquiera de estas entidades.

## A.7 Especificaciones de pruebas

### A.7.1 Introducción

Las pruebas son los procedimientos que se utilizan para verificar la conformidad de la implementación con respecto a la especificación. Los sistemas de pruebas pueden considerarse componentes distribuidos que interactúan con las implementaciones. Como los sistemas de pruebas son, simplemente, un tipo particular de sistemas distribuidos, las pruebas pueden describirse por medio de las FDT normalizadas Estelle, LOTOS y SDL. Las especificaciones de pruebas en estas FDT se examinan en A.7.2, A.7.3 y A.7.4.

Sin embargo, el marco para pruebas de conformidad de [ISO 9646] utiliza algunos conceptos particulares (como los de arquitectura de prueba, PCO, estructura de sucesiones de pruebas, veredictos), que no son construcciones predefinidas de las FDT normalizadas Estelle, LOTOS y SDL. En cambio, el lenguaje TTCN ha sido especialmente diseñado para describir formalmente casos de prueba. Las especificaciones de pruebas en TTCN se examinan en A.7.5.

### A.7.2 Especificaciones de pruebas en Estelle

Estelle ha sido concebido como un lenguaje de especificación, no como un lenguaje de descripción de pruebas. Por tanto, a diferencia de TTCN, no tiene construcciones predefinidas que representen los conceptos de prueba que aparecen en el cuerpo principal de esta Recomendación (caso de prueba, sucesión de pruebas, evento de prueba, veredicto). Desde luego, stelle puede considerarse como un lenguaje de programación, y es posible describir en Estelle la parte comportamiento de los casos de prueba (que, en el marco de entrada-salida, corresponde a mensajes de entrada y mensajes de salida). Además, los conceptos de prueba antes mencionados pueden modelarse en Estelle tomando determinadas decisiones (por ejemplo, utilización de una variable para codificar el veredicto, utilización de un cuerpo para representar el comportamiento de un caso de prueba, etc.). Sin embargo, éstas no son más que decisiones de modelado en las que, por naturaleza, hay muchas trampas. En consecuencia, en este anexo no se utilizará Estelle como un lenguaje de descripción de pruebas. Cuando se describan especificaciones en Estelle, se utilizará TTCN como lenguaje de descripción de pruebas. Dado que ambos lenguajes utilizan un mecanismo de comunicación de entrada-salida basado en colas FIFO, ponerlos en relación para la ejecución de pruebas no plantea ningún problema (véase A.8).

### A.7.3 Especificaciones de pruebas en LOTOS

El propio lenguaje LOTOS puede utilizarse para la especificación de pruebas [Bri 88, Tre 92]. Un caso de prueba puede modelarse por un proceso LOTOS en que cada estado es etiquetado por uno de los elementos {**éxito**, **fracaso**, **no concluyente**}. En la figura A.10 se muestra un ejemplo de tal prueba.

No es fácil modelar veredictos en LOTOS. Una posibilidad es utilizar la etiqueta especial $\sigma$ (terminación con éxito) para pasar un veredicto.

Al igual que Estelle y SDL, LOTOS no ha sido concebido para describir casos de pruebas. El lenguaje TTCN ha sido concebido para la notación de pruebas. Sin embargo, las expresiones TTCN representan casos de prueba que son asíncronos por naturaleza. Dado que la comunicación LOTOS es síncrona, esto puede ser un problema cuando se describen pruebas mediante expresiones TTCN.

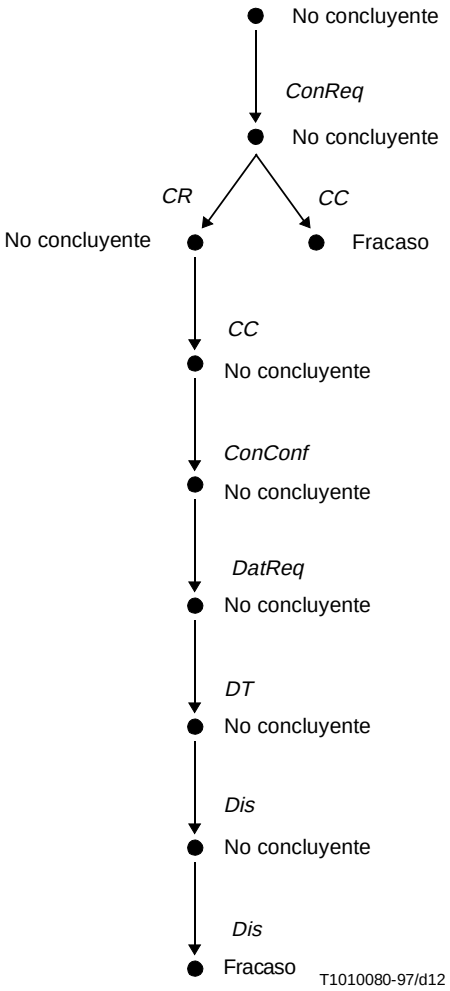


**Figura A.10/Z.500 – Representación de un caso de prueba como un sistema de transiciones etiquetadas**

### A.7.4 Especificaciones de pruebas en SDL

SDL es una técnica de especificación de carácter general para el modelado de sistemas de comunicación. Puede utilizarse también para la especificación de casos de prueba en las pruebas de conformidad. Sin embargo, no está especialmente concebido para este campo de aplicación. Por tanto, para utilizar SDL en la especificación de casos de prueba, como se definen en [ISO 9646], los conceptos de esta Recomendación deben definirse mediante construcciones SDL.

La mayor parte de los conceptos [ISO 9646] pueden modelarse directamente en SDL. Loas conceptos de PCO e intercambios de mensajes pueden modelarse por canales e instancias de señales. Un caso de prueba puede modelarse como un procedimiento SDL, y una sucesión de pruebas como un proceso. Por tanto, el orden de ejecución de caso de prueba se define en la definición de proceso. Las asignaciones de veredicto pueden modelarse por variables asociadas a cada caso de prueba. Las asignaciones de veredicto que se producen como resultado de ejecuciones de casos de prueba pueden pasarse al entorno por medio de señales. Este método está definido en [R1072]. Dado que se puede utilizar ASN.1 en combinación con SDL para definiciones de datos [Z105] las reglas para datos en [ISO 9646] pueden utilizarse directamente.

Algunos conceptos definidos en la metodología de prueba de [ISO 9646] no pueden especificarse en SDL. Este es el caso de los requisitos en tiempo real y el tratamiento de mensajes imprevistos. En SDL, los requisitos en tiempo real no pueden modelarse, pues la semántica SDL define solamente un modelo discreto para el tiempo, y no se prescribe nada en cuanto al periodo de tiempo que toma realizar una transición. En una especificación SDL sólo se da significado a señales declaradas. El tratamiento de mensajes imprevistos, que en TTCN está cubierto por la construcción *otherwise* (de lo contrario) sólo puede modelarse mediante una declaración explícita de señales que representarían esos mensajes.

### A.7.5 Especificaciones de pruebas en TTCN

El lenguaje normalizado notación combinada de árbol y tabla (TTCN, *tree and tabular combined notation*), ha sido concebido para la notación y especificación de casos de pruebas de conformidad que pueden expresarse de forma abstracta mediante el control y observación de unidades de datos de protocolo y primitivas de servicio abstracto. TTCN se proporciona en dos formas:

– Una forma gráfica, adecuada para su lectura por las personas;

– Una forma procesable por la máquina, adecuada para la transmisión de descripciones TTCN entre máquinas.

Los casos de prueba descritos por expresiones TTCN designan casos de prueba *abstractos*, que son compilados para formar casos de prueba ejecutables, que habrán de ser ejecutados en la máquina de prueba física.

TTCN es un lenguaje de alto nivel para la especificación de casos de prueba. Sin embargo, al igual que en el caso de las especificaciones (véase A.1), es más fácil definir los aspectos formales en un modelo más básico. Dado que TTCN tiene una semántica de entrada-salida, se ha utilizado *IOSM* como un modelo semántico de casos de prueba TTCN.

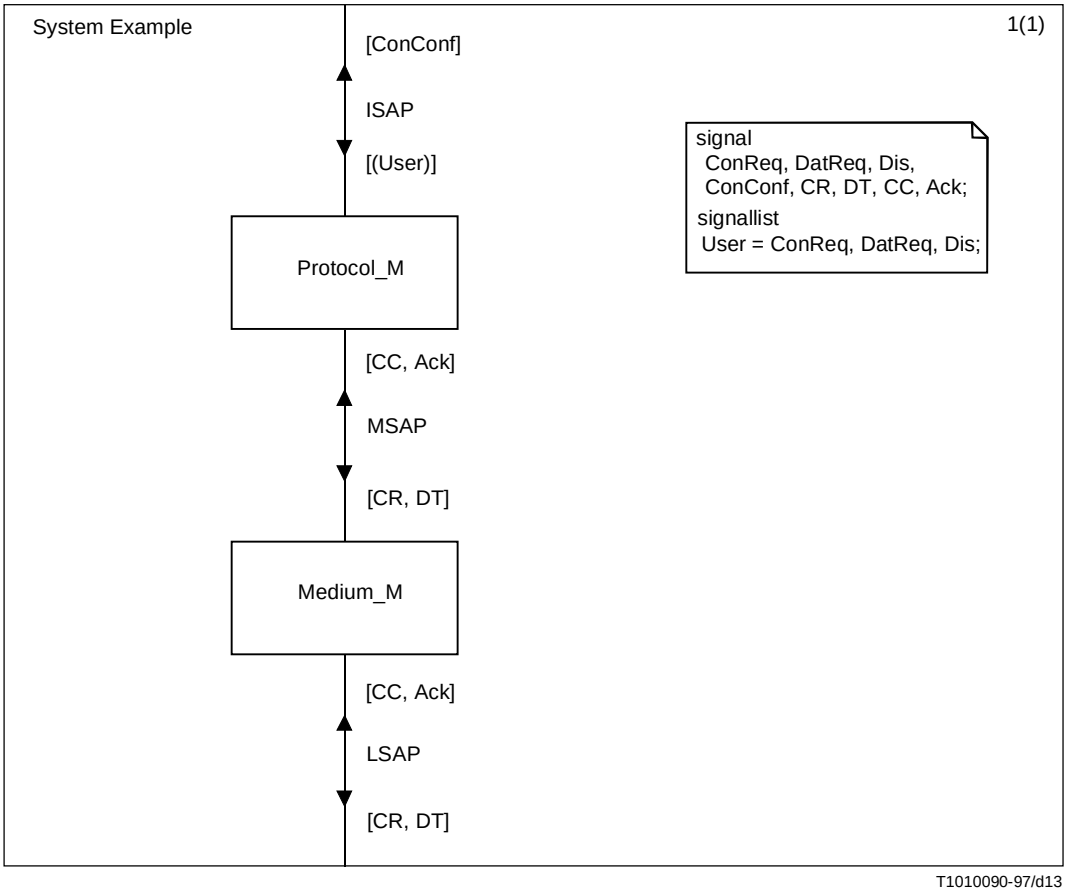


**Figura A.11/Z.500 – El diagrama de sistema del protocolo**

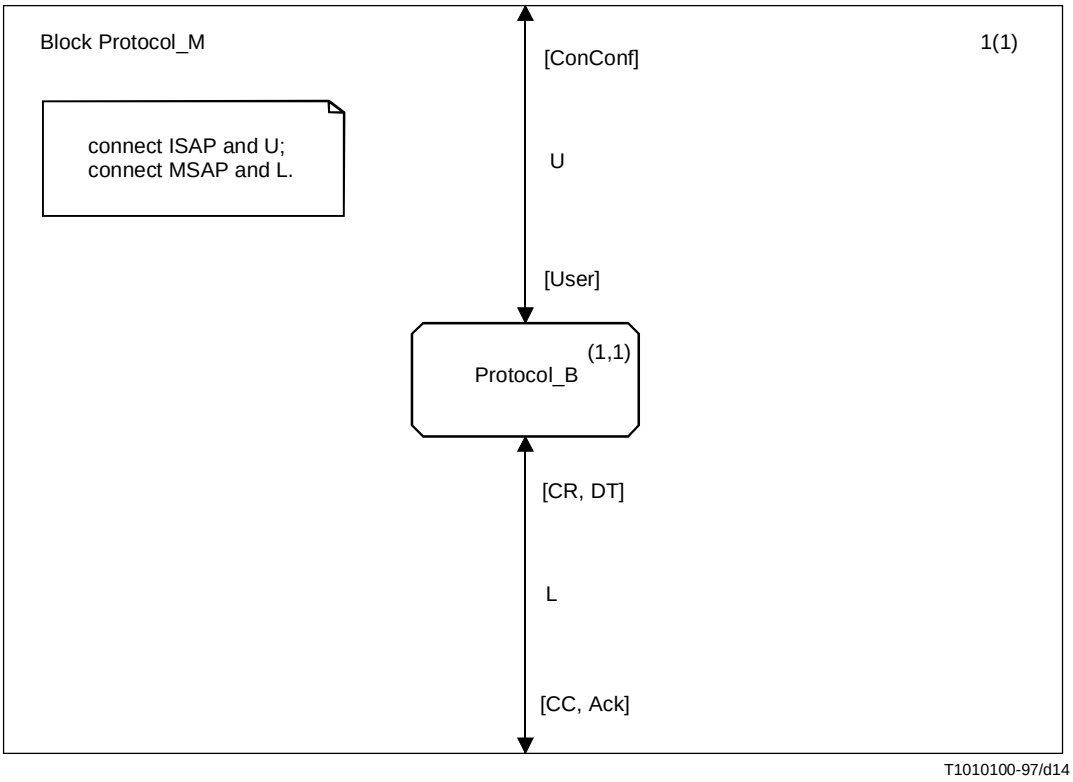


**Figura A.12/Z.500 – El bloque del proceso emisor**

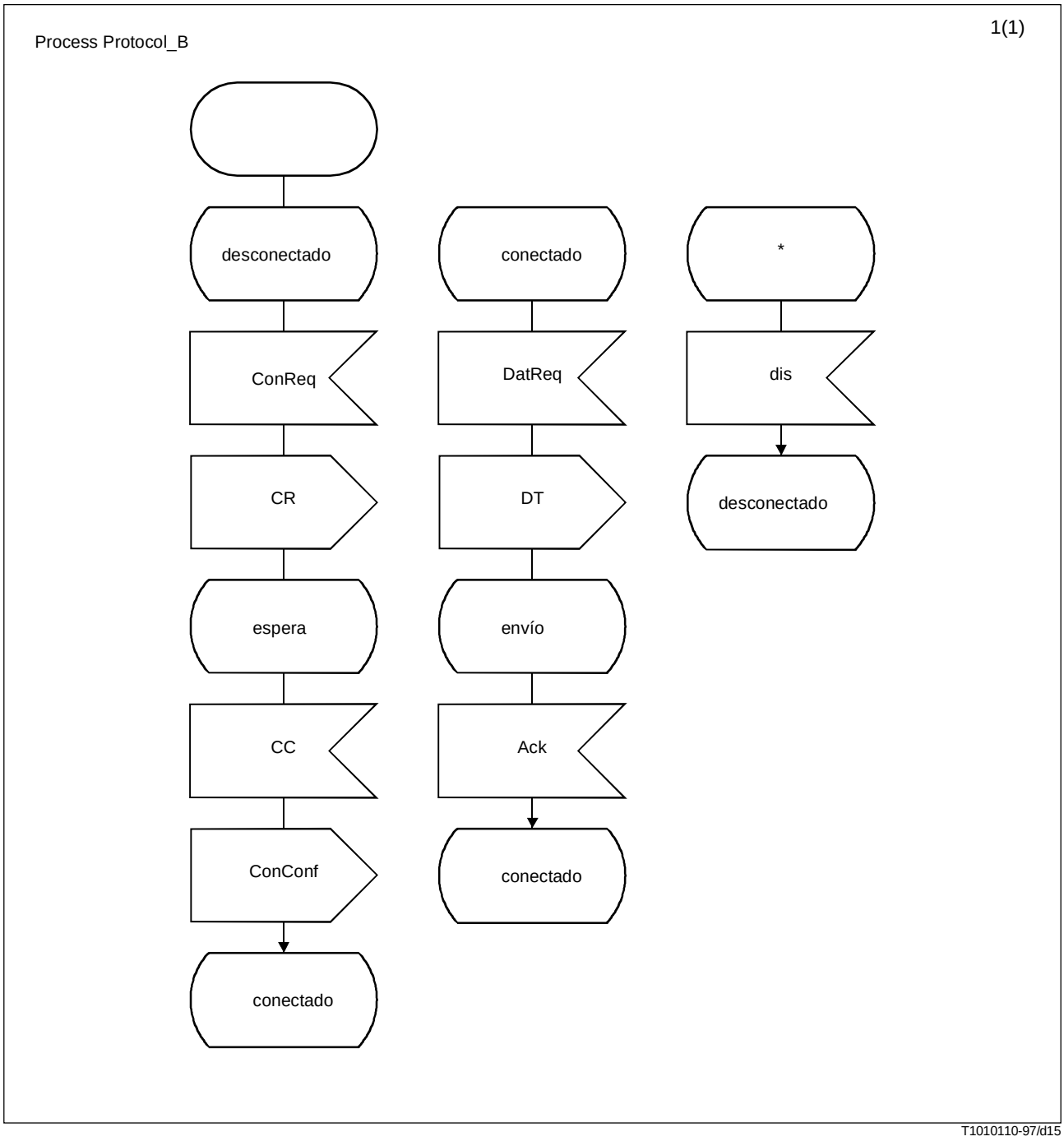


**Figura A.13/Z.500 – El proceso emisor**

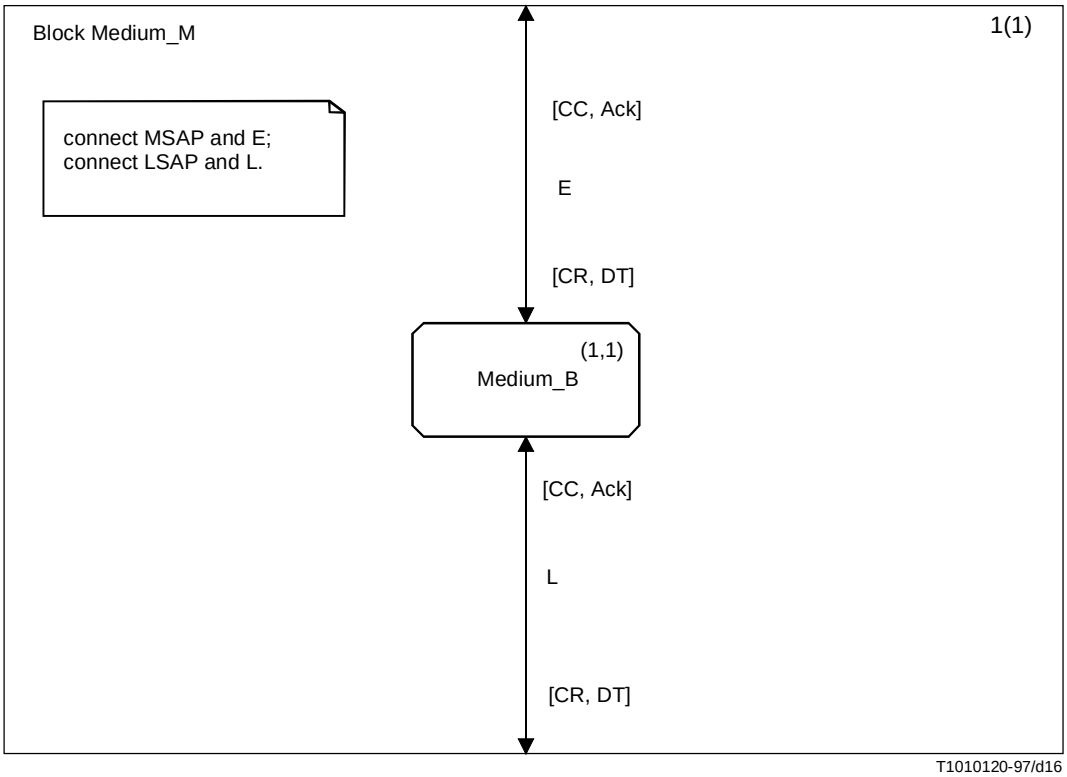


**Figura A.14/Z.500 – El bloque que modela al medio no fiable**

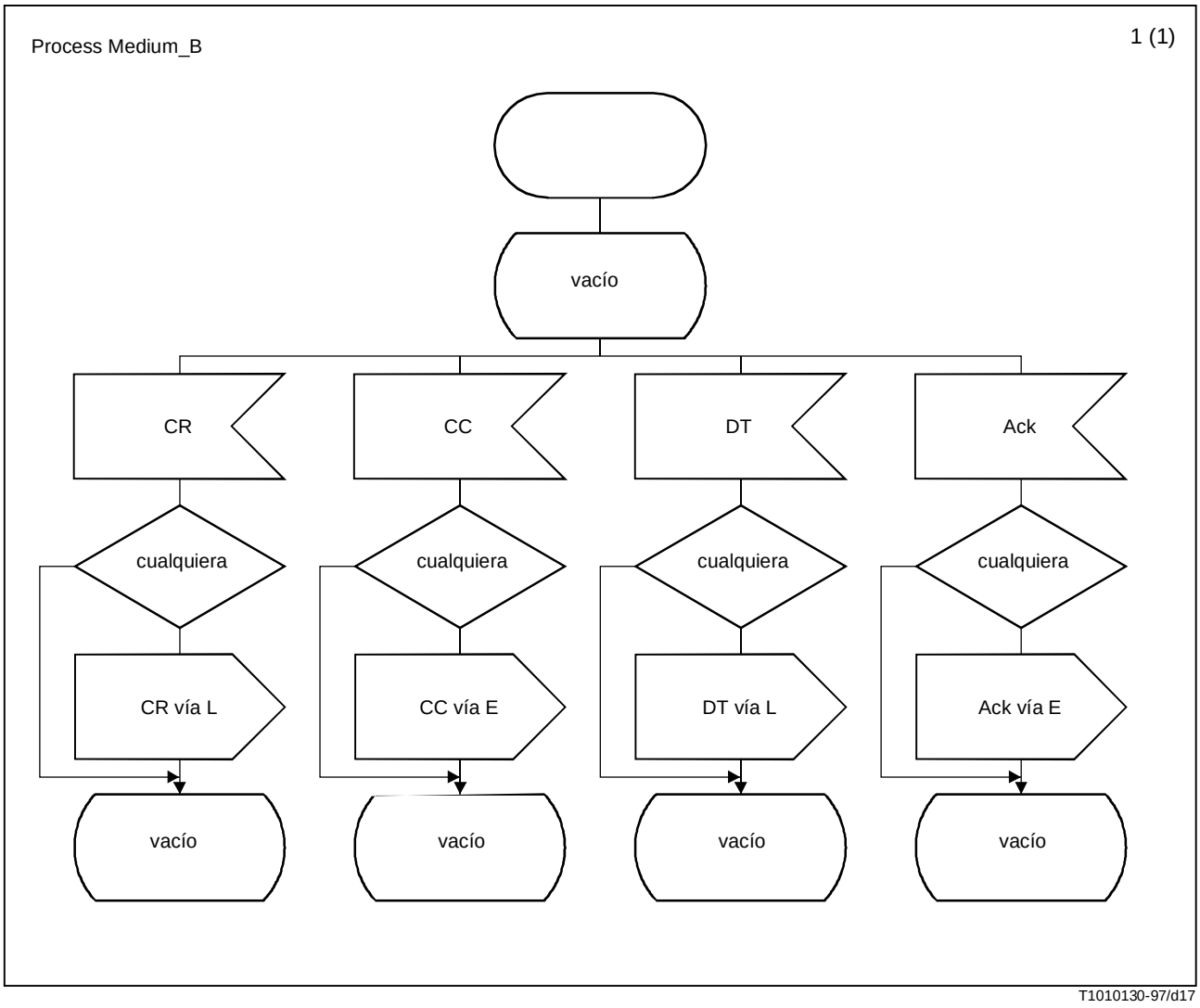


**Figura A.15/Z.500 – El comportamiento del medio no fiable**

## A.8 Referencias


[Bri 88] BRINKSMA (E.): A theory for the derivation of tests, *Protocol Specification*, Testing and Verification VIII, pp. 63-74, North Holland, 1987.

[Hogrefe 88] HOGREFE (D.): Automatic generation of test cases from SDL specifications, *SDL Newsletter*, N.º 12, 1988.

[Hopcroft 79] HOPCROFT (J.) y ULLMAN (J.): Introduction to automata theory, languages and computation, Addison-Wesley publishing company, 1979.

[ISO 8073] *Information technology – Telecommunications and information exchange between systems – Open Systems Interconnection – Protocol for providing the connection-mode transport service*, International Standard IS-8073, 3.ª edición, 1992

[ISO 8807] *Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour*, International Standard IS-8807, ISO, 1989.

[ISO 9646] *Information technology – Open Systems Interconnection – Conformance testing methodology and framework*, International Standard IS-9646, ISO.

[Luo 93] LUO (G.), PETRENKO (A.) y BOCHMANN (G.V.): Selecting test sequences for partially-specified nondeterministic finite state machines, Publication N.º 864, Université de Montréal, febrero de 1993.

[Phalippou 92] PHALIPPOU (M.): The limited power of testing – *Proceedings of the 5th International Workshop on Protocol Test Systems*, Montréal, septiembre de 1992.

[Phalippou 94] PHALIPPOU (M.): Relations d'implantation et hypothèses de test sur des automates à entrées et sorties, *Thèse de l'Université de Bordeaux I*, septiembre de 1994.

[R1072] ITACA, IBCN Testing Architecture for Conformance Assessment, R1072 ITACA – N.º 365, 1992.

[Tret 92] TRETMANS (J.): A Formal Approach to conformance Testing, *PhD thesis*, *University of Twente*, 1992.

[TrVe 92] TRETMANS (J.) y VERHAARD (L.): A queue model relating synchronous and asynchronous communication, *Memorandum INF-92-04*, University of Twente, Enschede, The Netherlands. TFL RR 1992-1, *Tele Danmark Research*, 1992.

[Z105] Recomendación UIT-T Z.105, *SDL combinado con ASN.1 (SDL/ASN.1)*, *UIT*, *1995*.

## 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 | Transmisiones de señales radiofónicas, de televisión 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 Z** | **Lenguajes de programación** |