UNIÓN INTERNACIONAL DE TELECOMUNICACIONES

# UIT-T T.173

(07/97)

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

SERIE T: TERMINALES PARA SERVICIOS DE TELEMÁTICA

# Representación de intercambio de guiones MHEG-3

Recomendación UIT-T T.173

(Anteriormente Recomendación del CCITT)

RECOMENDACIONES DE LA SERIE T DEL UIT-T

**TERMINALES PARA SERVICIOS DE TELEMÁTICA**

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

**RECOMENDACIÓN UIT-T T.173**

# REPRESENTACIÓN DE INTERCAMBIO DE GUIONES MHEG-3

## Orígenes

La Recomendación UIT-T T.173 ha sido preparada por la Comisión de Estudio 16 (1997-2000) del UIT-T y fue aprobada por el procedimiento de la Resolución N.° 1 de la CMNT el 10 de julio 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 ha recibido/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

**Página**

**Página**

**Página**

## Introducción

La presente Recomendación, que es equivalente a ISO/CEI 13522-3, es una Recomendación genérica que amplía la representación codificada de la clase de objeto guión MHEG definida en las Recomendaciones UIT-T de la serie T.170 y especialmente en las Recomendaciones T.171 y T.172. La presente Recomendación especifica la representación de intercambio de guión MHEG (MHEG-SIR) para el contenido de objetos guión, es decir, la codificación del componente de datos guión de la clase guión MHEG.

Las Recomendaciones UIT-T de la serie T.170 comprenden las siguientes Recomendaciones aprobadas y en proyecto:

– T.170 (1998), *Marco de las Recomendaciones de la serie T.170.*

– T.171 (1996), *Protocolos para servicios audiovisuales interactivos: Representación codificada de objetos multimedios e hipermedios.*

– T.172 (1998), *MHEG-5 – soporte para aplicaciones interactivas de nivel básico.*

– T.173 (1997), *Representación de intercambio del guión del MHEG-3.*

– T.174 (1996), *Interfaz de programación de aplicación para MHEG-1.*

– T.175 (1998), *Interfaz de programación de aplicación para MHEG-5.*

– T.176 (1998), *Interfaz de programación de aplicación para instrucciones y control de medios de almacenamiento digital.*

Ciertas Recomendaciones UIT-T de la serie T.170 son equivalentes a las partes de ISO/CEI 13522. Esta norma consta de las siguientes partes, bajo el título general Tecnología de la información – Codificación de información de multimedios e hipermedios:

– Parte 1: Notación de base [*Base notation*; (*ASN.1*)].

– Parte 3: Representación del intercambio de guiones MHEG *(MHEG script interchange representation*).

– Parte 4: Procedimiento de registro MHEG (*Registration procedure for MHEG format identifier*).

– Parte 5: Soporte de aplicaciones interactivas en el nivel de base (*Support for base-level interactive applications*).

– Parte 6: Soporte de aplicaciones interactivas potenciadas (*Support for enhanced interactive applications*).

Los anexos A a G forman parte integrante de esta Recomendación. Los apéndices I a IV son exclusivamente informativos.

**Recomendación T.173**

# REPRESENTACIÓN DE INTERCAMBIO DE GUIONES MHEG-3

*(Ginebra, 1997)*

## 1 Alcance

Esta Recomendación tiene por finalidad ampliar la representación codificada de la clase de objeto guión MHEG definida por la Rec. UIT-T T.171 (e ISO/CEI 13522-1) [5] y la Rec. UIT-T T.172 (e ISO/CEI 13522-5) [7].

Esta Recomendación especifica la representación de intercambio de guión MHEG (MHEG-SIR) para el contenido de objetos guión, es decir, la codificación del componente datos guión de la clase guión MHEG.

Los motores MHEG son componentes del sistema o de la aplicación que tratan, interpretan y presentan objetos de MHEG. Esta Recomendación especifica también las semánticas de guiones intercambiados. Estas semánticas se definen en términos de requisitos mínimos impuestos al comportamiento de motores MHEG que soportan la interpretación de guiones intercambiados.

Esta Recomendación es aplicable a todas las aplicaciones que intercambian información multimedios e hipermedios.

## 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.

[1] Recomendación UIT-T X.680 (1994) | ISO/CEI 8824-1:1995, *Tecnología de la información – Notación de sintaxis abstracta uno: Especificación de la notación básica*.

[2] Recomendación UIT-T X.690 (1994) | ISO/CEI 8825-1:1995, *Tecnología de la información – Reglas de codificación de notación de sintaxis abstracta uno: Especificación de las reglas de codificación básica, de las reglas de codificación canónica y de las reglas de codificación distinguida*.

[3] Recomendaciones UIT-T X.290 a X.296 (1995), Metodología y marco de las pruebas de conformidad de interconexión de sistemas abiertos de las Recomendaciones sobre los protocolos para aplicaciones del UIT-T.

ISO/CEI 9694 Parts 1 to 5, *Information technology – Open Systems Interconnection – Conformance testing methodology and framework.*

*Part 1: 1994, General concepts.*

*Part 2: 1994, Abstract Test Suite specification.*

*Part 3: 1992, The Tree and Tabular Combined Notation (TTCN).*

*Part 4: 1994, Test scalization*.

*Part 5: 1994, Requirements on test laboratories and clients for the conformance assessment process.*

[4] ISO/CEI 10646-1:1993, *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane.*

[5] Recomendación UIT-T T.171 (1996), *Protocolos para servicios audiovisuales interactivos: Representación codificada de objetos multimedios e hipermedios.*

ISO/CEI 13522-1:1997, *Tecnología de la información – Codificación de información multimedios e hipermedios – Parte 1: Representación de objeto de MHEG – Notación de base*.

[6] ISO/CEI 13522-4:1996, *Information technology – Coding of multimedia and hypermedia information: – Part 4: MHEG registration procedure*.

[7] Recomendación UIT-T T.172 (1998), *MHEG-5 – Soporte para aplicaciones interactivas de nivel básico*.

ISO/CEI 13522-5:1997, *Information technology – Coding of multimedia and hypermedia information – Part 5: Support for base-level interactive applications*.

[8] ISO/CEI 14750[1], *Information technology – Open Distributed Processing – Interface Definition Language.* (Anteriormente ISO/CEI 14750-1.)

[9] IEEE 754:1985, *IEEE standard for binary floating-point arithmetic*.

## 3 Definiciones y abreviaturas

### 3.1 Definiciones

Para los fines de esta Recomendación, son aplicables las siguientes definiciones que figuran en la Rec. UIT-T X.680 | ISO/CEI 8824-1 [1], la Rec. UIT-T X.690 | ISO/CEI 8825-1 [2].

**3.1.1 interfaz de programación de aplicación (API, *application programming interface*)**: Frontera a través de la cual una aplicación informatizada utiliza facilidades de lenguajes de programación para invocar servicios informatizados. Estas facilidades pueden incluir procedimientos u operaciones, objetos de datos compartidos y medios para la resolución de identificadores.

**3.1.2 atributo (*attribute*)**:

1) Atributo MHEG (véase ISO/CEI 13522-1 [5]).

2) Atributo IDL (véase más abajo).

**3.1.3 motor MHEG-3 conforme (*conforming MHEG-3 engine*)**: Motor MHEG-3 cuya implementación es conforme con las disposiciones de esta Recomendación.

**3.1.4 guión intercambiado MHEG-3 conforme (*conforming MHEG-3 interchanged script*)**: Guión intercambiado que es conforme con las disposiciones de esta Recomendación.

**3.1.5 objeto (de) MHEG-3 conforme (*conforming MHEG-3 object*)**: Objeto de guión MHEG cuya representación codificada es conforme con las disposiciones de esta Recomendación.

**3.1.6 trama (*frame*)**: Registro (*record*) de elementos en la pila de llamadas que define un contexto de ejecución; uno de estos registros se inserta en la pila de llamadas cada vez que se invoca una rutina, para memorizar el contexto de ejecución actual; se saca un registro de la pila de llamada

---

[1] Actualmente en la etapa de proyecto.

cuando se retorna de la rutina, para restablecer el contexto de ejecución que existía en el instante en que se invocó la rutina.

**3.1.7 hipermedios [*hypermedia (adj.)*]**: (como adjetivo) Se refiere al acceso de información monomedio y multimedio por interacción con enlaces explícitos.

**3.1.8 guión intercambiado (*interchanged script*)**: La representación codificada del atributo "script data" de un objeto de guión MHEG.

**3.1.9 lenguaje de definición de interfaz (IDL, *interface definition language*)**: Notación formal utilizada para especificar tipos y objetos mediante la definición de la interfaz que ellos proporcionan, según lo definido por ISO/CEI 14750-1 [8].

**3.1.10 atributo IDL (*IDL attribute*)**: Asociación, caracterizada por un nombre y un tipo, entre un objeto y un valor; se declara como parte de una interfaz IDL; se hace visible a los clientes como un par de operaciones: un accesor (*get*) y un modificador (*set*); si el atributo es de lectura solamente, sólo proporciona un accesor.

**3.1.11 excepción IDL (*IDL exception*)**: Mensaje que puede señalarse cuando se presenta una condición excepcional durante el tratamiento de una petición de una operación IDL; se define en un módulo IDL y puede tener miembros, que se retornan al llamante junto con el identificador de mensaje.

**3.1.12 instancia IDL (*IDL instance*)**: Objeto que proporciona las operaciones, firmas y semánticas especificadas por una interfaz IDL; su creación y gestión son específicas de la implementación.

**3.1.13 interfaz IDL (*IDL interface*)**: Descripción, mediante el lenguaje IDL, de un conjunto de operaciones que un cliente puede solicitar de un objeto IDL.

**3.1.14 objeto IDL (*IDL object*)**: Entidad identificable, encapsulada, que proporciona uno o más servicios que pueden ser solicitados por un cliente.

**3.1.15 operación IDL (*IDL operation*)**: Servicio que puede ser solicitado y proporcionado por un objeto IDL; se define dentro de una interfaz IDL por un nombre, una firma que define el tipo de sus parámetros y el valor de retorno, y la lista de excepciones a que pueden señalarse como consecuencia de su invocación.

**3.1.16 guión mh; mh-script**: Representación interna, dentro de un motor MHEG, de un objeto de guión MHEG "disponible" ("available").

**3.1.17 acción MHEG (*MHEG action*)**: Operación que se aplica a objetos de MHEG y consiste en combinaciones de acciones elementales MHEG en serie o en paralelo.

**3.1.18 objeto de acción MHEG (*MHEG action object*)**: Objeto (de) MHEG que describe acciones MHEG.

**3.1.19 aplicación MHEG (*MHEG application*)**: Aplicación que comprende el intercambio de objetos de MHEG dentro de la misma aplicación o con otra aplicación.

**3.1.20 objeto (de) MHEG conforme (*MHEG conforming object*)**: Objeto de información cuya representación codificada es conforme con las disposiciones de una de las partes de ISO/CEI 13522.

**3.1.21 acción elemental MHEG (*MHEG elementary action*)**: Una de las operaciones básicas aplicables a objetos de MHEG; corresponde a una primitiva MHEG API.

**3.1.22 motor MHEG (*MHEG engine*)**: Proceso o conjunto de procesos capaces de interpretar objetos de MHEG.

**3.1.23 entidad MHEG (*MHEG entity*)**: Cualquier objeto de MHEG, object rt, datos de contenido, datos de guión, zócalo (*socket*), canal u otra construcción definidos por ISO/CEI 13522.

**3.1.24 enlace MHEG (*MHEG link*)**: Objeto de MHEG que define relaciones espaciales-temporales entre objetos de MHEG expresadas en términos de condiciones de iniciación y acciones.

**3.1.25 objeto de MHEG (*MHEG object*)**: Representación codificada de una instancia de una clase de objeto de MHEG.

**3.1.26 clase de guión MHEG (*MHEG script class*)**: Clase de MHEG que define una estructura para intercambiar datos de guión en una forma codificada especificada.

**3.1.27 objeto de guión MHEG (*MHEG script object*)**: Representación codificada de una instancia de una clase de guión MHEG.

**3.1.28 MHEG API**: La interfaz API proporcionada por un motor MHEG a aplicaciones MHEG para la manipulación de objetos de MHEG.

**3.1.29 MHEG-3**: **(*como adjetivo*)** Se aplica a entidades que son conformes con las disposiciones de esta Recomendación.

**3.1.30 aplicación MHEG-3 (*MHEG-3 application*)**: Aplicación MHEG que intercambia guiones consigo misma y/o con otras aplicaciones como el componente "script data" de objetos de guión MHEG de acuerdo con la representación especificada en esta Recomendación.

**3.1.31 motor MHEG-3 (*MHEG-3 engine*)**: Motor MHEG que procesa e interpreta guiones intercambiados MHEG-SIR.

**3.1.32 perfil MHEG-3 (*MHEG-3 profile*)**: Perfil de la Recomendación T.173.

**3.1.33 MHEG-SIR**:

1) La representación de intercambio de guión definida por esta Recomendación.
2) (como adjetivo) Se aplica a una entidad definida como parte de esta representación de intercambio de guión.

**3.1.34 pila de llamadas (MHEG-SIR) [*(MHEG-SIR) call stack*]**: Pila que es asociada, por la máquina virtual MHEG-SIR, con cada guión rt en funcionamiento y que contiene una trama de llamada para cada invocación de función activa.

**3.1.35 código MHEG-SIR (*MHEG-SIR code*)**: Secuencia codificada de instrucciones MHEG-SIR.

**3.1.36 constante (MHEG-SIR) [*(MHEG-SIR) constant*]**: Valor estático, caracterizado por un tipo y un nombre, que se declara dentro de un guión intercambiado y cuyo valor es accesible globalmente y permanece inalterado durante la ejecución del guión.

**3.1.37 tipo construido (MHEG-SIR) [*(MHEG-SIR) constructed type*]**: Tipo descrito como una combinación de otros tipos formada mediante uno de los siguientes constructores: sequence, string, array, union, structure.

**3.1.38 identificador de datos (MHEG-SIR) [*(MHEG-SIR) data identifier*]**: Número entero que identifica unívocamente el nombre de un elemento de datos de un guión intercambiado (constante, variable global, variable dinámica, variable local).

**3.1.39 excepción (MHEG-SIR) [*(MHEG-SIR) exception*]**: Mensaje señalado durante la invocación de un servicio.

**3.1.40 función (MHEG-SIR) [*(MHEG-SIR) function*]**: Secuencia de código que tiene un nombre, cuya ejecución puede ser invocada por un guión intercambiado; puede ser una rutina, una función predefinida o un servicio.

**3.1.41 identificador de función (MHEG-SIR) [*(MHEG-SIR) function identifier*]**: Número entero que identifica unívocamente una función en un guión intercambiado.

**3.1.42 variable global (MHEG-SIR) [*(MHEG-SIR) global variable*]**: Variable con alcance global.

**3.1.43 instrucción (MHEG-SIR) [*(MHEG-SIR) instruction*]**: Unidad elemental de código de un guión intercambiado MHEG-SIR; consiste en un código de operación seguido de cero o más operandos.

**3.1.44 unidad de ejecución de instrucción (MHEG-SIR) [*(MHEG-SIR) instruction execution unit*]**: En un intérprete de guión MHEG-SIR, la unidad de procesamiento virtual que ejecuta una instrucción MHEG-SIR.

**3.1.45 guión intercambiado (MHEG-SIR) [*(MHEG-SIR) interchanged script*]**: Guión intercambiado codificado de acuerdo con MHEG-SIR.

**3.1.46 variable local (MHEG-SIR) [*(MHEG-SIR) local variable*]**: Variable con alcance local dentro de una rutina.

**3.1.47 mensaje (MHEG-SIR) [*(MHEG-SIR) message*]**: Evento que el intérprete de guión puede recibir durante la ejecución del guión; puede ser predefinido (excepción MHEG API, operación y excepción MHEG-3 API, excepción interna) o declarado en un guión intercambiado (excepción proporcionada por una interfaz externa).

**3.1.48 identificador de mensaje (MHEG-SIR) [*(MHEG-SIR) message identifier*]**: Número entero que identifica unívocamente un mensaje en un guión intercambiado.

**3.1.49 cola de mensajes (MHEG-SIR) [*(MHEG-SIR) message queue*]**: Cola que es asociada con cada guión rt en funcionamiento por la máquina virtual MHEG-SIR y que contiene los mensajes previstos para el guión rt.

**3.1.50 referencia de objeto (MHEG-SIR) [*(MHEG-SIR) object reference*]**: Valor MHEG-SIR que representa una instancia IDL y que se pasa como el parámetro de una llamada externa para solicitar un servicio de esta instancia.

**3.1.51 operando (MHEG-SIR) [*(MHEG-SIR) operand*]**: Parámetro de una instrucción; se codifica inmediatamente del código de operación de la instrucción.

**3.1.52 lote (MHEG-SIR) [*(MHEG-SIR) package*]**: Conjunto de funciones externas proporcionadas por un módulo del entorno de ejecución y que son accesibles a un guión rt y están declaradas en un guión intercambiado; comprende servicios y excepciones.

**3.1.53 parámetro (MHEG-SIR) [*(MHEG-SIR) parameter*]**: Un determinado dato que se intercambia con una llamada a función, un mensaje o una instrucción.

**3.1.54 pila de parámetro (MHEG-SIR) [*(MHEG-SIR) parameter stack*]**: Pila que la máquina virtual MHEG-SIR asocia con cada guión rt en funcionamiento y que se utiliza para proporcionar parámetros a instrucciones y extraer resultados de instrucciones.

**3.1.55 tipo predefinido (MHEG-SIR) [*(MHEG-SIR) predefined type*]**: Un tipo cuya descripción e identificador están predefinidos por esta Recomendación, por lo que no es necesario declararlo en guiones intercambiados; puede ser un tipo primitivo o un tipo construido.

**3.1.56 tipo primitivo (MHEG-SIR) [*(MHEG-SIR) primitive type*]**: Tipo predefinido básico, por oposición a tipo construido.

**3.1.57 rutina (MHEG-SIR) [*(MHEG-SIR) routine*]**: Función que se declara en un guión intercambiado, junto con el código de máquina virtual que define su semántica.

**3.1.58 unidad de ejecución de guión rt (MHEG-SIR) [*(MHEG-SIR) rt-script execution unit*]**: En un intérprete de guión MHEG-SIR, la unidad de procesamiento virtual que ejecuta el código de guión.

**3.1.59 intérprete de guión (MHEG-SIR) [*(MHEG-SIR) script interpreter*]**: La parte de un motor MHEG-3 que trata e interpreta guiones intercambiados.

**3.1.60 servicio (MHEG-SIR) [*(MHEG-SIR) service*]**: Función externa que se declara en un guión intercambiado y cuya implementación por el entorno de ejecución lo hace accesible a un guión rt en la plataforma de ejecución.

**3.1.61 variable (MHEG-SIR) [*(MHEG-SIR) variable*]**: En la máquina virtual MHEG-SIR unidad de memoria caracterizada por un nombre y un tipo, cuyo valor puede ser cambiado en cualquier momento en el que su alcance esté activo, y cuyo valor más reciente puede leerse.

**3.1.62 máquina virtual (MHEG-SIR) [*(MHEG-SIR) virtual machine*]**: Descripción abstracta de las unidades de memoria y del motor de ejecución de instrucción de un intérprete de guión MHEG-SIR.

**3.1.63 multimedios [*(multimedia (adj.)*]**: (como adjetivo) Califica a todo aquello que trabaje con varios medios de representación de tipos diferentes.

**3.1.64 aplicación multimedios e hipermedios (*multimedia and hypermedia application*)**: Aplicación que proporciona la presentación al usuario de información multimedios y la navegación interactiva por el usuario a través de esta información.

**3.1.65 aplicación multimedios (*multimedia application*)**: Aplicación que proporciona presentación al usuario de información multimedios.

**3.1.66 especificación de correspondencia de plataforma (*platform mapping specification*)**: Especificación de la manera en que las implementaciones de motor MHEG-3 harán corresponder especificaciones IDL con componentes del entorno de ejecución en un tipo de plataforma.

**3.1.67 cola (*queue*)**: Colección de elementos que son insertados y suprimidos en un orden de "primero en entrar, primero en salir" (FIFO, *first-in first-out*).

**3.1.68 guión rt (*rt-script*)**: Instancia (o ejemplar, o copia) en la fase de ejecución de un guión mh, creada por un motor MHEG.

**3.1.69 alcance (*scope*)**: Contexto de referencia para una variable; si es global, la variable puede ser referenciada por cualquier instrucción del guión; si es local, la variable sólo puede ser referenciada en el contexto de ejecución local.

**3.1.70 lenguaje de guión (*scripting language*)**: Lenguaje de programación concebido para permitir a programadores no profesionales un diseño fácil y rápido de aplicaciones.

**3.1.71 representación de intercambio de guiones (SIR, *script interchange representation*)**: Representación codificada utilizada por una aplicación para intercambiar guiones con el fin de implementar un comportamiento dinámico.

**3.1.72 pila (*stack*)**: Colección de elementos que son insertados o suprimidos en un orden de último en entrar primero en salir (LIFO, *last-in first-out*).

## 3.2 Abreviaturas

En esta Recomendación se utilizan las siguientes siglas.

| | |
|---|---|
| API | Interfaz de programas de aplicación (*application programming interface*) |
| ASN.1 | Notación de sintaxis abstracta uno (*abstract syntax notation one*) |
| CEI | Comisión Electrotécnica Internacional |
| CORBA | Arquitectura de intermediario de respuesta de objeto común (*common object request broker architecture*) |

| | |
|---|---|
| CS | Pila de llamadas (*call stack*) |
| CT | Tabla de constantes (*constant table*) |
| DER | Reglas de codificación distinguida (*distinguished encoding rules*) |
| DID | Identificador de datos (*data identifier*) |
| DT | Tabla de datos (*data table*) |
| EBNF | Forma Backus-Naur ampliada (*extended Backus-Naur form*) |
| ER | Registro de error (*error register*) |
| ETR | Informe técnico de ETSI (*ETSI technical report*) |
| FID | Identificador de función (*function identifier*) |
| FIFO | Primero en entrar primero en salir (*first-in first-out*) |
| FP | Puntero de trama (*frame pointer*) |
| FR | Registro de función (*function register*) |
| GT | Tabla de definiciones de variables globales (*global variable definition table*) |
| HT | Tabla de definiciones de manejadores (*handler definition table*) |
| IDL | Lenguaje de definición de interfaz (*interface definition language*) |
| IP | Puntero de instrucción (*instruction pointer*) |
| IR | Registro de instrucción (*instruction register*) |
| ISO | Organización Internacional de Normalización (*International Organisation for Standardization*) |
| UIT-T | Unión Internacional de Telecomunicaciones – Sector de Normalización de las Telecomunicaciones |
| JTC | Comité técnico mixto (*joint technical committee*) |
| LIFO | Último en entrar primero en salir (*last-in first-out*) |
| LT | Tabla de variables locales (*local variable table*) |
| MHEG | Grupo de Expertos en codificación de información multimedios e hipermedios (*multimedia and hypermedia information coding experts group*) |
| MID | Identificador de mensaje (*message identifier*) |
| MPEG/DSM-CC | Grupo de Expertos en imágenes animadas – Instrucciones y control de medios de almacenamiento digital (*moving picture experts group – digital storage media command and control*) |
| MQ | Cola de mensajes (*message queue*) |
| PID | Identificador de lote (*package identifier*) |
| PS | Pila de parámetros (*parameter stack*) |
| PT | Tabla de definiciones de lotes (*package definition table*) |
| QP | Puntero de cola (*queue pointer*) |
| rt | (Fase de) ejecución (*run-time*) |

| | |
|---|---|
| RT | Tabla de definiciones de rutinas (*routine definition table*) |
| SIR | Representación de intercambio de guión (*script interchange representation*) |
| SP | Puntero de pila (*stack pointer*) |
| ST | Tablas de definiciones de servicios (*service definition table*) |
| TID | Identificador de tipo (*type identifier*) |
| TLV | Valor de longitud de tipo (*type-length-value*) |
| TT | Tabla de definiciones de tipos (*type definition table*) |
| VT | Tabla de variables (*variable table*) |
| XT | Tabla de definiciones de excepciones (*exception definition table*) |

## 4 Generalidades

Las Recomendaciones UIT-T de la serie T.170 (e ISO/CEI 13522) especifican la representación codificada de objetos de información multimedios/hipermedios (objetos de MHEG) para intercambio como unidades en forma final dentro, o a través, de servicios y aplicaciones, por cualquier medio de intercambio, incluidas redes de área local, redes de telecomunicaciones de área extensa o de difusión, medios de almacenamiento, etc.

Los objetos de MHEG suelen ser producidos por herramientas informatizadas que utilizan, como una forma de fuente, aplicaciones multimedios diseñadas utilizando lenguajes de guión de multimedios. En este contexto, una de las clases de objetos de MHEG, la clase guión, tiene por finalidad complementar las otras clases MHEG al expresar la funcionalidad usualmente soportada por lenguajes de guión. Los objetos de guión expresan mecanismos de control más potentes y describen relaciones más complejas, entre objetos de MHEG, que las que pueden expresarse por objetos de MHEG de acción y de enlace, solamente. Además, los objetos de guión expresan acceso e interacción con servicios externos proporcionados por el entorno de ejecución.

Otras Recomendaciones de la serie T.170 definen la representación codificada de objetos de guión de una manera abierta, de modo que los objetos de guión puedan encapsular código de guión normalizado o código de guión privado (dícese también, propietario). Los objetos de guión encapsulan guiones que pueden ser codificados en cualquier formato de codificación registrado de acuerdo con ISO/CEI 13522-4 [6].

## 5 Conformidad

Esta Recomendación define requisitos de conformidad que deberán ser satisfechos por:

– objetos de información, es decir objetos de guión MHEG;

– implementaciones, es decir, implementaciones de motor MHEG.

### 5.1 Conformidad de objeto de información

Un objeto de guión MHEG-3 conforme deberá satisfacer los siguientes criterios:

1) su representación codificada será conforme con las disposiciones de las Recomendaciones UIT-T de la serie T.170 (e ISO/CEI 13522);

2) su representación codificada encapsulará un guión intercambiado MHEG-3 conforme.

La conformidad de objeto de información se evalúa sobre objetos de información que se intercambian para su ejecución en un terminal.

### 5.1.1 Perfiles

Esta Recomendación no define perfiles.

NOTA 1 – Sin embargo, otras normas u otras partes de ISO/CEI 13522 pueden definir perfiles MHEG-3. De acuerdo con el marco de definición de perfiles, los perfiles MHEG-3 normalizados deben ser por lo menos tan restrictivos como los definidos por esas otras normas o partes; los objetos que pretenden ser conformes con esos perfiles deben, por lo menos, ser conformes con esta Recomendación.

Un perfil MHEG-3 debe definir todo lo siguiente:

– un perfil de la máquina virtual MHEG-SIR definida por esta Recomendación;

– un perfil de IDL, así como su correspondencia con MHEG-SIR, para la expresión de la interfaz entre los guiones y el entorno externo;

– una API para la manipulación de objetos de MHEG definida por las Recomendaciones UIT-T de la serie T.170 (e ISO/CEI 13522), así como la correspondencia de esta interfaz con MHEG-SIR.

NOTA 2 – De acuerdo con las Normas de ISO, los perfiles MHEG-3 deben asegurar la compatibilidad ascendente de la codificación ASN.1, de modo que los guiones intercambiados conformes con un perfil MHEG-3 sean también conformes con esta Recomendación.

### 5.1.2 Codificación

Un guión intercambiado MHEG-3 se codificará de acuerdo con las reglas de codificación definidas en el anexo B.

### 5.1.3 Sintaxis

Un guión intercambiado MHEG-3 será conforme con la sintaxis ASN.1 definida en el anexo A.

### 5.1.4 Semántica

Un guión intercambiado MHEG-3 conforme incluirá solamente declaraciones semánticamente válidas y secuencias de instrucciones definidas en las cláusulas 12 y 13.

## 5.2 Conformidad de las implementaciones

Una implementación de esta Recomendación es un motor MHEG-3.

Un motor MHEG-3 conforme soportará la interpretación de objetos de guión MHEG-3 conformes.

Esta Recomendación define la semántica de guiones intercambiados MHEG-3. Esto implica que se imponen requisitos de conformidad no a objetos de información, sino al comportamiento de motores MHEG-3.

NOTA 1 – Aunque un guión conforme pudiera no realizar la semántica implicada por su diseñador, la forma en que se comportan los motores conformes al interpretar este guión podrá predecirse.

NOTA 2 – Esta Recomendación no considera la conformidad de un sistema, un motor o un proceso cuanto no se relacione con la interpretación de guiones intercambiados.

### 5.2.1 Requisitos de conformidad

Un motor MHEG-3 conforme cumplirá los siguientes criterios:

1) analizará e interpretará guiones intercambiados MHEG-3 conformes de acuerdo con el comportamiento de la máquina virtual definido en esta Recomendación (véase la cláusula 9);

2) soportará la comunicación con el entorno de ejecución y con objetos de MHEG de acuerdo con el comportamiento indicado por la correspondencia de la especificación IDL, definida en esta Recomendación (véanse las cláusulas 10, 11 y 14);

3) proporcionará la MHEG-3 API definida en esta Recomendación (véanse la cláusula 15 y el anexo F);

4) para los fines de la manipulación de objetos de MHEG por guiones intercambiados, soportará una MHEG API y su correspondencia de acuerdo con el marco definido en esta Recomendación (véase el anexo E);

5) para los fines de la comunicación con el entorno de ejecución, soportará una especificación de correspondencia de plataforma de acuerdo con el marco definido en esta Recomendación (véase el anexo D);

6) podrá proporcionar funciones o facilidades adicionales no requeridas por esta Recomendación o por la especificación de correspondencia de plataforma. Cada una de estas extensiones no normalizadas se identificará como tal en la documentación del sistema.

### 5.2.2 Documentación de conformidad

Para toda implementación que pretenda ser conforme con esta Recomendación se proporcionará un documento de conformidad con la información que se indica más adelante. Este documento de conformidad cumplirá los siguientes criterios:

1) indicará todas las características obligatorias requeridas por esta Recomendación, con referencia a las cláusulas y subcláusulas apropiadas;

2) incluirá, o bien la especificación de correspondencia de plataforma con la que es conforme la implementación, o bien una referencia inequívoca a una especificación de correspondencia de plataforma registrada;

3) contendrá un enunciado que indica los títulos completos, números y fechas de las normas aplicables.

4) indicará qué características facultativas definidas en esta Recomendación y en la especificación de correspondencia de plataforma están soportadas por la implementación;

5) describirá el comportamiento de la aplicación con respecto a todas las características definidas por la implementación que estén definidas en esta Recomendación y en la especificación de correspondencia de plataforma. Este requisito se cumplirá indicando estas características y proporcionando, o bien una referencia específica a la documentación del sistema, o la sintaxis y semántica completas de estas características. Este documento de conformidad puede especificar el comportamiento de la implementación con respecto a aquellas características para las que esta Recomendación o la especificación de correspondencia de plataforma indique que las implementaciones pueden variar, o cuando las características se identifiquen como no definidas o no especificadas.

Ninguna otra especificación aparte de las especificadas en esta Recomendación y en la especificación de correspondencia de plataforma estarán presentes en el documento de conformidad.

## 5.3 Conformidad de las aplicaciones

Una aplicación de esta Recomendación (denominada aplicación MHEG-3) es una aplicación MHEG que intercambia guiones consigo misma y/o con otras aplicaciones como el componente "script data" (datos de guión) de objetos de guión MHEG de acuerdo con la representación codificada especificada en esta Recomendación.

## 5.4 Métodos de prueba

Toda medición de la conformidad con esta Recomendación se realizará utilizando métodos de pruebas conformes a las Recomendaciones UIT-T de la serie X.290 (e ISO/CEI 9646) [3].

## 6 Visión de conjunto

Esta Recomendación amplía las disposiciones de otras Recomendaciones UIT-T de la serie T.170 (y de otras partes de ISO/CEI 13522) de modo que los objetos y aplicaciones de MHEG soporten la funcionalidad de lenguajes de guión de multimedios de una manera normalizada. Habida cuenta de la funcionalidad especificada por otras Recomendaciones UIT-T de la serie T.170 (y de otras partes de ISO/CEI 13522), estas ampliaciones tratan dos temas principales:

– operaciones de procesamiento de datos (véase 6.2);

– acceso a datos y funciones externos (véase 6.3).

Para el tratamiento de ambos temas, esta Recomendación especifica:

– disposiciones completas y detalladas para la codificación de guiones intercambiados;

– el comportamiento requerido de un intérprete de guión.

### 6.1 Metodología de descripción

Para la descripción de estas disposiciones, esta Recomendación sigue una metodología que considera cuatro niveles de descripción:

– nivel a): descripción textual informal;

– nivel b): descripción precisa de semántica;

– nivel c): descripción formal de sintaxis;

– nivel d): descripción formal de codificación.

Estos niveles se utilizan en las cláusulas y anexos que se indican a continuación:

– nivel a): cláusulas 8 a 11;

– nivel b): cláusulas 12 a 15;

– nivel c): anexos A, E, F, G;

– nivel d): anexos B, C.

NOTA – En los apéndices I y II (informativos) se utiliza también la descripción de nivel c).

### 6.2 Operaciones de procesamiento de datos

Para tratar las operaciones de procesamiento de datos, la MHEG-SIR define la estructura de guiones intercambiados que consisten en declaraciones de datos y declaraciones de funciones; las declaraciones de funciones encapsulan secuencias de instrucciones.

La cláusula 8 define los elementos del código de máquina virtual MHEG-SIR.

La cláusula 9 especifica la máquina virtual MHEG-SIR, es decir, un modelo de la manera en que los intérpretes de guión MHEG-SIR realizarán la interpretación de código guión MHEG-SIR. Esta máquina virtual se utiliza después para describir la semántica de las instrucciones MHEG-SIR. La cláusula 9 indica los requisitos impuestos a la funcionalidad que los intérpretes de guión deben proporcionar; sin embargo, no especifica cómo implementar esta funcionalidad.

La cláusula 12 define las declaraciones de guiones intercambiados MHEG-SIR. Especifica su estructura, es decir, la forma en que deberán representarse, y su semántica, es decir, la forma en que serán interpretados por los intérpretes de guión MHEG-SIR. La semántica se especifica mediante el formalismo de máquina virtual introducido en la cláusula 9.

La cláusula 13 define las instrucciones MHEG-SIR. Especifica su estructura, es decir, la forma en que serán representadas, y su semántica, es decir, la forma en que serán interpretadas por los

intérpretes de guión MHEG-SIR. Estas semánticas se especifican utilizando el formalismo de máquina virtual introducido en la cláusula 9.

El anexo A presenta la definición formal, en la notación ASN.1, de la sintaxis precisa de guiones intercambiados.

El anexo B presenta la definición formal de la codificación de guiones intercambiados.

El anexo C indica los elementos predefinidos de la MHEG-SIR y define su codificación.

El anexo G presenta la definición formal de las relaciones de esta Recomendación con las Recs. UIT-T T.171 (e ISO/CEI 13522-1) y T.172 (e ISO/CEI 13522-5), es decir, los objetos de MHEG en estas partes a los que se aplica la MHEG-SIR, y la forma en que ésta se aplica a aquellos.

### 6.3 Acceso a datos y funciones externos

Para tratar el acceso a datos y funciones externos, la MHEG-SIR utiliza el lenguaje IDL para describir interfaces de una manera abstracta, independiente del lenguaje, y, de ese modo, unificar la manera en que los datos y funciones externos son percibidos por los intérpretes de guión.

En el contexto de la MHEG-SIR, el lenguaje IDL se utiliza para separar claramente la manera en que la utilización (específica de la MHEG-SIR) de datos y funciones externos es expresada por guiones intercambiados, de la manera (que por lo menos depende de la plataforma y puede depender también de la aplicación) en que estos datos o funciones son proporcionados por el entorno externo. Por tanto, la MHEG-SIR define la forma en que se utilizan las interfaces, mientras que incumbe a la aplicación definir la forma en que se proporcionan.

Para permitir que los intérpretes de guión manipulen entidades MHEG e intercambien información con ellas, los motores MHEG-3 proporcionan intérpretes de guión con acceso las entidades (datos) MHEG e invocación de acciones (funciones) MHEG a través de una interfaz MHEG API definida mediante el lenguaje IDL. Los tipos y acciones MHEG están predefinidos en MHEG-SIR para lograr una codificación compacta y una interpretación eficiente de la manipulación de objetos de MHEG.

Para permitir que los intérpretes de guión cooperen con el entorno de ejecución, el entorno de ejecución proporciona acceso a sus datos y funciones de acuerdo con una especificación de correspondencia de plataforma, en IDL. Esta especificación describe cómo las operaciones IDL deben proporcionarse en una determinada plataforma de manera que los motores MHEG-3 puedan utilizarlas como servicios externos.

NOTA – Se puede proporcionar lotes en forma de bibliotecas, gestores de dispositivos, componentes del sistema operativo, procesos, servicios de telecomunicación, etc.

La cláusula 7 describe los supuestos sobre la estructura de motores MHEG-3 y sus relaciones con su entorno, de que se ha partido.

La cláusula 10 describe los mecanismos generales utilizados para ganar acceso a datos y funciones externos proporcionados por el entorno de ejecución.

La cláusula 11 describe los mecanismos generales utilizados para manipular objetos de MHEG.

La cláusula 14 especifica la correspondencia del IDL con la MHEG-SIR, es decir, los mecanismos utilizados por la representación MHEG-SIR para describir lotes IDL e invocar operaciones IDL.

La cláusula 15 especifica la estructura y la semántica de la MHEG-3 API, es decir, el conjunto de operaciones que pueden utilizarse para manipular guiones.

El anexo D especifica la plantilla en IDL para la especificación de correspondencia de plataforma, es decir, la plantilla del documento que debe llenarse y registrarse por cada tipo de plataforma, para especificar disposiciones propias de cada plataforma que deberán ser respetadas por los servicios

proporcionados por el entorno de ejecución en esta plataforma, y determinar los motores MHEG-3 con los que serán conformes, de modo que puedan cooperar con servicios proporcionados por el entorno de ejecución en esta plataforma y, por tanto, interpretar guiones que invocan esos servicios.

El anexo E especifica el marco que se utilizará para definir una MHEG API utilizando IDL y el procedimiento que se seguirá para hacerla corresponder con la MHEG-SIR.

El anexo F define la sintaxis precisa de la MHEG-3 API, utilizando la notación IDL.

## 7 Relaciones entre MHEG y MHEG-3

Esta cláusula presenta supuestos generales sobre los motores MHEG-3, que se utilizan posteriormente para describir la calidad de funcionamiento de un intérprete de guión y sus relaciones con su entorno externo.

Los motores MHEG-3 proporcionarán de alguna manera la funcionalidad que se describe más adelante; dichos motores se comportarán como está previsto, en lo que respecta a la interpretación de guiones intercambiados.

Sin embargo, los motores MHEG-3 no están obligados a implementar esta funcionalidad en la forma descrita.

NOTA – Por ejemplo, los componentes funcionales del motor MHEG-3 que se describen a continuación no tienen que corresponder a componentes (por ejemplo, informatizados) efectivos de implementaciones de motores MHEG-3.

### 7.1 Entidades MHEG

Los motores MHEG-3 tratan entidades MHEG: objetos de MHEG, objetos de mh, objetos de rt, objetos de MHEG intercambiados, zócalos (*sockets*), canales.

NOTA – Las entidades MHEG se describen con más detalle en el apéndice III.

### 7.2 Entidades funcionales

Puede considerarse que los motores MHEG-3 están formados por los siguientes componentes funcionales:

– analizador (*parser*) de objeto de MHEG: analiza objetos de MHEG intercambiados y los transforma en objetos de mh bajo el control del gestor de objetos de mh;

– gestor de objetos de mh: controla el ciclo de vida de todos los objetos de mh y permite el acceso a dichos objetos;

– gestor de objetos de rt: controla el ciclo de vida de todos los objetos de rt y permite el acceso a dichos objetos;

– resolvedor de referencias: transforma una referencia MHEG en un identificador o asa (*handle*) utilizable;

– manejador de enlaces: vigila los enlaces activos e inicia las correspondientes acciones cuando se cumplen las condiciones que provocan su ejecución;

– intérprete de acción: interpreta las acciones elementales MHEG;

– intérprete de guiones: analiza guiones intercambiados MHEG-SIR e interpreta guiones rt; da acceso al entorno de ejecución;

– agente de presentación: interfaz con el entorno de presentación; ordena la presentación de contenido rt; recibe selecciones y modificaciones efectuadas por el usuario;

– agente de acceso: interfaz con el entorno de comunicación; da acceso a objetos de MHEG intercambiados y a datos de contenido.

### 7.3 Intérprete de guión MHEG-SIR

En un motor MHEG-3, el intérprete de guiones se encargará de lo siguiente:

– analizar guiones intercambiados (proporcionados por el analizador de objeto de MHEG);
– preparar las estructuras de datos apropiadas para la ulterior ejecución de guiones rt;
– ejecutar códigos de guión;
– realizar el resultado por defecto de acciones MHEG prescritas para guiones mh o guiones rt;
– invocar el manejador apropiado (en el programa guión) para estas acciones MHEG;
– reenviar, al intérprete de acciones, las acciones elementales MHEG invocadas por el programa guión;
– gestionar el intercambio con el entorno de ejecución (localización y carga de lotes, invocación de servicios, recepción de mensajes, paso de datos), utilizando los mecanismos de comunicación apropiados, específicos de la plataforma.

## 8 Elementos de MHEG-SIR

Esta cláusula describe los principales elementos de MHEG-SIR y la forma en que deberán ser utilizados por los guiones intercambiados.

Los guiones intercambiados MHEG-SIR declaran y manipulan:

– tipos de datos;
– datos;
– funciones;
– mensajes.

Estos conceptos se definen en las subcláusulas siguientes; sin embargo, la estructura detallada de sus declaraciones se especifica en la cláusula 12.

### 8.1 Tipos de datos

Se utilizan tipos de datos para describir la estructura de:

– los datos (constantes y variables) del propio guión;
– los parámetros y valores de retorno de las rutinas del guión;
– los parámetros y valores de retorno de funciones externas;
– los parámetros de mensajes tratados por guiones.

Puesto que los guiones deben adaptarse a la firma de las funciones que pueden ser proporcionadas por el entorno externo, la MHEG-SIR define una amplia gama de tipos que corresponden a los tipos de datos IDL.

La codificación de los tipos de datos en un guión intercambiado se define en el anexo A. Esta Recomendación no impone requisitos sobre la forma en que los motores MHEG-3 representan estos tipos de datos.

La MHEG-SIR utiliza dos modalidades de tipos de datos:

– tipos predefinidos (véase 8.1.1);
– tipos declarados (véase 8.1.2).

Todos los tipos pueden ser referenciados de manera única e inequívoca por su identificador de tipo.

### 8.1.1 Tipos predefinidos

Los tipos predefinidos pueden ser primitivos o construidos.

Los tipos predefinidos tienen identificadores de tipo predefinidos, por lo que los guiones intercambiados no tienen que declararlos. La lista de los tipos predefinidos y sus identificadores figura en el anexo C.

#### 8.1.1.1 Tipos primitivos

Los tipos primitivos corresponden a los tipos primitivos IDL. A continuación se presenta la lista de los tipos primitivos MHEG-SIR:

- **void**;
- **octet**;
- **short**;
- **long**;
- **unsigned short**;
- **unsigned long**;
- **float**;
- **double**;
- **boolean**;
- **character**;
- **data identifier**;
- **object reference**.

Para facilitar la referencia, los tipos primitivos tienen códigos formados por una sola letra (códigos de letra) que se indican en el cuadro 1.

**Cuadro 1/T.173 – Códigos de letra de los tipos primitivos**

| Tipo | Código de letra |
|---|---|
| **octet** | O |
| **short** | S |
| **long** | L |
| **unsigned short** | W (por la palabra inglesa Word) |
| **unsigned long** | U |
| **float** | F |
| **double** | D |
| **boolean** | B |
| **character** | C |
| **data identifier** | I (por Identifier) |
| **object reference** | R (por Reference) |

#### 8.1.1.1.1 Tipo **void**

El tipo **void** se utilizará solamente para expresar el tipo de valor de retorno de una función. Las funciones cuyo tipo de valor de retorno es **void** no retorna ningún dato. Un guión intercambiado no tendrá constantes ni variables de tipo **void**. El tipo **void** no se utilizará en la definición de tipos construidos.

#### 8.1.1.1.2 Tipo **octet**

Los datos cuyo tipo es **octet** tomarán un valor numérico comprendido en la gama [0 .. 255]. Las variables de tipo **octet** que no tengan un valor inicial explícito se inicializarán a 0.

#### 8.1.1.1.3 Tipo **short**

Los datos cuyo tipo es **short** tomarán un valor entero con signo comprendido en la gama [–32 768 .. 32 767]. Las variables de tipo **short** que no tengan un valor inicial explícito se inicializará a 0.

#### 8.1.1.1.4 Tipo **long**

Los datos cuyo tipo es **long** tomarán un valor entero con signo, comprendido en la gama [–2147483648 .. 2147483647]. Las variables de tipo **long** que no tengan un valor inicial explícito se inicializarán a 0.

#### 8.1.1.1.5 Tipo **unsigned short**

Los datos cuyo tipo es **unsigned short** tomarán un valor entero sin signo, comprendido en la gama [0 .. 65535]. Las variables de tipo **unsigned short** que no tengan un valor inicial explícito se inicializarán a 0.

#### 8.1.1.1.6 Tipo **unsigned long**

Los datos cuyo tipo es **unsigned long** tomarán un valor entero sin signo, comprendido en la gama [0 .. 4294967295]. Las variables de tipo **unsigned long** que no tengan un valor inicial explícito se inicializarán a 0.

#### 8.1.1.1.7 Tipo **float**

Los datos cuyo tipo es **float** tomarán un valor de coma flotante de precisión simple, comprendido en la gama especificada por IEEE 754 [9]. Las variables de tipo **float** que no tengan un valor inicial explícito se inicializarán a 0.

#### 8.1.1.1.8 Tipo **double**

Los datos cuyo tipo es **double** tomarán un valor de coma flotante de precisión doble, comprendido en la gama especificada por IEEE 754 [9]. Las variables de tipo **double** que no tengan un valor inicial explícito se inicializarán a 0.

#### 8.1.1.1.9 Tipo **boolean**

Los datos cuyo tipo es **boolean** tomarán el valor "true" o el valor "false". Las variables de tipo **boolean** que no tengan un valor inicial explícito se inicializarán a "false".

#### 8.1.1.1.10 Tipo **character**

Los datos cuyo tipo es **character** tomarán un valor de carácter de los que constituyen el juego de caracteres **BMPString** definido por el plano multilingüe básico (*basic multilingual plane*) de ISO/CEI 10646-1 [4]. Las variables de tipo **character** que no tengan un valor inicial explícito tendrán un valor inicial no definido.

Los motores MHEG-3 conformes pueden enunciar que sólo adoptan un juego restringido de caracteres, por ejemplo basados en las colecciones normalizadas del anexo A de ISO/CEI 10646-1 [4]. En este caso, deberán indicar estos subconjuntos adoptados y el nivel de implementación en el documento de conformidad.

#### 8.1.1.1.11 Tipo **data identifier**

Los datos cuyo tipo es **data identifier** tomarán un valor entero sin signo comprendido en la gama [0 .. 65535]. Este valor se utiliza para identificar una constante, variable global, variable dinámica, variable local o parámetro de rutina del guión, como se especifica en 8.6.2. No habrá constantes de tipo **data identifier**. Las variables de tipo **data identifier** que no tengan un valor inicial explícito tendrán un valor inicial no definido.

#### 8.1.1.1.12 Tipo **object reference**

Los datos cuyo tipo es **object reference** tomarán como valor un asa que hace referencia a un objeto IDL al que se aplican servicios o funciones predefinidas. La codificación de referencias de objeto se define en la especificación de correspondencia de plataforma. No habrá constantes de tipo **object reference**. El tipo **object reference** no se utilizará en la definición de tipos construidos. Las variables de tipo **object reference** que no tengan un valor inicial explícito tendrán un valor inicial no definido.

Las referencias de objeto se utilizan como el primer parámetro implícito de todas las llamadas externas para especificar el objeto a que se aplica la llamada. Los valores de referencia de objeto serán proporcionados por el entorno externo, como un parámetro de salida o valor de retorno de una instrucción **external call (XCALL)**. Se utiliza la instrucción **get object reference (GETOR)** para obtener una primera referencia de objeto en el objeto raíz de un determinado lote. Se utiliza la referencia de objeto **null** para hacer referencia al objeto original de la MHEG-3 API (instancia de **ScriptInterpreter**).

### 8.1.1.2 Tipos construidos predefinidos

Para permitir que los guiones expresen más fácilmente la manipulación de datos MHEG, los tipos de datos de la MHEG-API están predefinidos.

Aunque no están definidos dentro de guiones intercambiados, los tipos construidos predefinidos, como los tipos construidos declarados, pueden expresarse utilizando constructores de tipo e identificadores de tipo, como se describe en 8.1.2. Para expresar la estructura de tipos construidos predefinidos sólo se utilizaran identificadores de tipo predefinidos.

## 8.1.2 Tipos construidos declarados

Se definirán tipos construidos utilizando un constructor y uno o varios identificadores de tipo que identifican, o bien un tipo declarado, o un tipo predefinido.

El constructor de un tipo construido será uno de los tipos siguientes:

– **sequence** (véase 8.1.2.1);

– **string** (véase 8.1.2.2.);

– **array** (véase 8.1.2.3);

– **structure** (véase 8.1.2.4);

– **union** (véase 8.1.2.5).

Los tipos declarados se definen dentro de guiones intercambiados.

Los tipos MHEG-SIR no serán redefinidos en un guión intercambiado. La estructura de un tipo declarado no deberá concordar con la estructura de un tipo predefinido ni con la de otro tipo declarado.

No habrá más de 16 384 tipos declarados en un guión intercambiado.

#### 8.1.2.1 Tipos **sequence**

Los tipos **sequence** se definirán por

– su tamaño (facultativo);

– su tipo de elemento.

El tamaño será un valor unsigned short. Representa el número máximo de elementos de la secuencia. Si la definición de tipo especifica la ausencia de tamaño, el número de elementos puede ser cualquiera, hasta el tamaño máximo. Los tipos **sequence** con un tamaño explícito se denominan tipos **sequence** acotados (*bounded*).

El tamaño máximo de cualquier tipo **sequence** es 65 535 elementos.

El tipo de elemento puede ser cualquier tipo primitivo, construido o predefinido, excepto **void** y **object reference**. Se hará referencia al tipo de elemento mediante su identificador de tipo. Las definiciones de tipo **sequence** no conducirán a una recursión infinita.

NOTA – Por consiguiente, el identificador de tipo de la secuencia (**sequence**) puede estar contenido (anidado) dentro de la definición de tipo solamente debajo de un constructor **union**.

Los datos cuyo tipo es un tipo definido utilizando el constructor **sequence** tomarán por valor una lista ordenada de cero o más valores del tipo de elemento.

Las variables de un tipo **sequence** que no tengan un valor inicial explícito serán inicializadas a una lista nula (secuencia de cero elementos).

#### 8.1.2.2 Tipos **string**

Los tipos **string** son semánticamente equivalentes a los tipos **sequence** cuyo tipo de elemento es **character**.

NOTA – Con el fin de optimizar el tratamiento de las cadenas de caracteres, para dar los valores a los elementos de la cadena de caracteres se puede emplear un procedimiento diferente del que se emplearía para las secuencias de caracteres. Por consiguiente, las cadenas de caracteres y las secuencias de caracteres siguen siendo tipos distintos, aunque semánticamente equivalentes.

Los tipos **string** se definirán por su tamaño (facultativo).

El tamaño será un valor unsigned short. Representa el número máximo de elementos de la cadena. Si la definición de tipo especifica la ausencia de tamaño, el número de elementos puede ser cualquiera, hasta el tamaño máximo. Los tipos **string** que tienen un tamaño explícito se denominan tipos **string** acotados.

El tamaño máximo de cualquier tipo **string** es 65 535 caracteres.

Los datos cuyo tipo es un tipo definido utilizando el constructor **string** tomarán por valor una cadena de cero o más caracteres.

Las variables de un tipo **string** que no tengan un valor inicial explícito se inicializarán a una cadena nula (cadena de cero caracteres).

#### 8.1.2.3 Tipos **array**

Los tipos **array** se definirán por:

– su tamaño;

– su tipo de elemento.

El tamaño será un valor unsigned short. Representa el número exacto de elementos de la matriz (*array*).

El tipo de elemento puede ser cualquier tipo primitivo, construido o predefinido, excepto **void** y **object reference**. Se hará referencia al tipo de elemento utilizando su identificador de tipo. Las definiciones de tipo **array** no conducirán a una recursión infinita.

NOTA – En consecuencia, el identificador de tipo **array** puede estar contenido (anidado) dentro de la definición de tipo solamente debajo de un constructor **union**.

Los datos cuyo tipo es un tipo definido utilizando el constructor **array** tomarán por valor una lista ordenada de valores del tipo de elemento, especificándose la longitud de la lista por el tamaño de la matriz.

Las variables de un tipo **array** que no tengan un valor inicial explícito se inicializarán a una lista de elementos cuyo valor inicial se determina por el tipo de elemento.

#### 8.1.2.4 Tipos **structure**

Los tipos **structure** se definirán por una lista ordenada de 1 a 256 tipos de elemento.

Los tipos de elemento pueden ser cualquier tipo primitivo, construido o predefinido, excepto **void** y **object reference**. Los tipos de elemento serán referenciados por sus identificadores de tipo. Las definiciones de tipo **structure** no conducirán a una recursión infinita.

NOTA – En consecuencia, el identificador de tipo **structure** puede estar contenido (anidado) dentro de la definición de tipo solamente debajo de un constructor **union**.

Los datos cuyo tipo es un tipo definido utilizando el constructor **structure** tomarán su valor como una lista ordenada de valores del tipo de elemento que corresponde a su rango en la definición de tipo.

Las variables de un tipo **structure** que no tengan un valor inicial explícito se inicializarán a una lista de elementos cuyo valor inicial se determina por su tipo de elemento.

#### 8.1.2.5 Tipos **union**

Los tipos **union** se definirán por una lista ordenada de tipos de elemento.

En un tipo **union** no habrá más de 256 posibilidades de elección (tipos de elemento).

Los tipos de elemento pueden ser cualquier tipo primitivo, construido o predefinido, excepto **void** y **object reference**. Los tipos de elemento serán referenciados por sus identificadores de tipo.

Los datos cuyo tipo es un tipo definido utilizando el constructor **union** tomarán por valor:

– un número entero que representa el índice (comenzando por 0) en la lista de posibilidades de elección;

– un valor del tipo de elemento cuyo rango en la definición de tipo es el índice antes mencionado.

Las variables de un tipo **union** que no tengan un valor inicial explícito tendrán un valor inicial no definido.

### 8.2 Datos

La MHEG-SIR define tres modalidades de datos:

– valores inmediatos (véase 8.2.1);

– constantes (véase 8.2.2);

– variables (véase 8.2.3).

Todos los datos utilizados por un guión intercambiado son de un determinado tipo de datos, predefinido o declarado.

Dos valores de datos serán iguales únicamente si:

– son del mismo tipo, es decir, tienen el mismo identificador de tipo;

– si son de un tipo primitivo, son idénticos;

– si son de un tipo **structure**, **sequence** o **array**, cada elemento de una de las listas es igual al elemento del mismo rango de la otra lista;

– si son de un tipo **union**, sus rótulos (*tags*) son idénticos y sus valores son iguales entre sí.

En consecuencia,

– los valores de un tipo **string** no serán comparados con valores de una **sequence** (secuencia) de tipo **character** (carácter);

– los valores de un tipo **sequence** acotada no serán comparados con valores de otro tipo **sequence** acotada ni con valores de un tipo **sequence** no acotada, pues tienen identificadores de tipo diferentes;

– los valores de un tipo **string** acotada no se compararán con valores de otro tipo **string** acotada ni con valores de un tipo **string** no acotada, pues tienen identificadores de tipo diferentes.

Todas las variables y constantes son referenciadas de una manera única e inequívoca por su identificador de datos.

### 8.2.1 Valores inmediatos

Los valores inmediatos son datos que no están declarados dentro del guión intercambiado y, por tanto, sólo pueden utilizarse "inmediatamente", es decir, en el mismo momento en que se encuentran. Un valor inmediato puede encontrarse en un guión intercambiado:

– como un valor constante;

– como el valor inicial de una variable;

– como el operando de una instrucción **push immediate** (**PUSHI**).

Además, se utilizan valores inmediatos en el curso de la ejecución de un guión a través de la pila de parámetros, como parámetros de instrucciones o funciones.

A menos que el contexto imponga otras restricciones, los valores inmediatos pueden ser de cualquier tipo excepto **void**.

La codificación de valores de datos en un guión intercambiado se define en el anexo A. Esta Recomendación no impone requisitos sobre la manera en que los motores MHEG-3 representan valores de datos de un tipo determinado.

### 8.2.2 Constantes

Las constantes serán declaradas dentro del guión intercambiado y definidas por:

– un tipo de datos;

– un valor de este tipo de datos.

Las constantes pueden ser de cualquier tipo, excepto:

– **object reference**;

– **data identifier**;

– **void**.

Las constantes tienen un alcance global y pueden ser referenciadas utilizando su identificador de tipo a través del guión intercambiado.

No puede haber más de 4096 constantes declaradas en un guión intercambiado.

### 8.2.3 Variables

Las variables se declararán dentro del guión intercambiado y se definirán por:

– un tipo de datos;

– facultativamente, un valor de este tipo de datos, que se tomará como el valor inicial de esta variable.

Las variables pueden ser de cualquier tipo excepto **void**.

Las variables son referenciadas utilizando su identificador de tipo. Una referencia a una variable puede utilizarse con una de las semánticas siguientes:

– semántica de "miembro situado a la derecha": produce el mismo efecto que si, en lugar de la variable, se asigna su valor;

– semántica de "miembro situado a la izquierda": indica que a esta variable hay que asignarle un valor de datos.

En este último caso, el valor que se asignará a la variable puede ser un valor inmediato (incluido un valor calculado), el valor de una constante o el valor de una variable (incluido el valor futuro de un parámetro de salida de una función).

La MHEG-SIR define tres modalidades de variables:

– variables globales (véase 8.2.3.1);

– variables locales (véase 8.2.3.2);

– variables dinámicas (véase 8.2.3.3).

#### 8.2.3.1 Variables globales

Las variables globales tienen un alcance global que abarca la totalidad del guión intercambiado. Pueden ser referenciadas mediante su identificador de datos a partir de cualquier rutina o variable. Se les puede asignar un nuevo valor en cualquier momento durante la ejecución del guión rt.

No habrá más de 28 672 variables globales declaradas en un guión intercambiado.

#### 8.2.3.2 Variables locales

Las variables locales tienen un alcance local que está limitado a la ejecución del código de la rutina a que pertenecen. Pueden ser referenciadas mediante su identificador de datos solamente dentro del código de esta rutina.

Hay dos modalidades de variables locales:

– variables locales que se declaran en la declaración de rutina como parte de la declaración de la variable local;

– parámetros efectivos de la rutina, hayan sido pasados por valor o por referencia, que se declaran en la declaración de rutina como parte de la firma de la rutina.

No habrá más de 256 variables locales declaradas en cada rutina de un guión intercambiado.

#### 8.2.3.3 Variables dinámicas

Las variables dinámicas tienen un alcance dinámico que se extiende desde el instante en que son creadas mediante una instrucción **allocate (ALLOC)** hasta el instante en que son liberadas mediante

una instrucción **free (FREE)**. En el momento de la creación de las variables dinámicas, el intérprete de guión les da un identificador de datos. Pueden ser referenciadas mediante su identificador de datos en cualquier momento, en el curso de la ejecución del guión. Sin embargo, como el identificador de datos de una variable sólo se conoce en la fase de ejecución, sólo puede utilizarse como un valor de parámetro en la pila de parámetros o como un valor de variable, y no como el operando de una instrucción.

No podrá utilizar más de 32 512 variables dinámicas en un momento dado durante la ejecución de un guión rt.

### 8.3 Funciones

La MHEG-SIR define tres modalidades de funciones:

– rutinas (véase 8.3.1);

– servicios (véase 8.3.2);

– funciones predefinidas (véase 8.3.3).

Todas las funciones tendrán una firma (o prototipo) que está constituido por:

– un tipo de valor de retorno;

– una lista ordenada de parámetros formales definidos por el tipo y el modo de paso de los parámetros.

Todas las funciones son referenciadas de una manera única e inequívoca mediante su identificador de función.

Las funciones pueden ser **synchronous** (síncronas) o **asynchronous** (asíncronas). Cuando se invoca una función síncrona, el llamante espera hasta la conclusión de la ejecución de la función y, por tanto, podrá obtener su resultado. Cuando se invoca una función asíncrona, el llamante sólo espera la recepción del acuse de recibo de la petición, después de lo cual reanuda la ejecución y no espera hasta la conclusión de la función.

En consecuencia, las funciones asíncronas no deben tener parámetros de salida ni valor de retorno. Las rutinas son siempre síncronas.

#### 8.3.1 Rutinas

Las rutinas son funciones internas de guiones intercambiados.

Las rutinas se declararán dentro del guión intercambiado. Constarán de:

– una firma;

– variables locales;

– código de programa.

No podrá haber más de 4096 rutinas declaradas en un guión intercambiado.

La ejecución de una rutina puede ser iniciada:

– por una instrucción **call (CALL)**, siendo el identificador de función de la rutina el operando de la instrucción;

– al recibirse una excepción durante una instrucción **external call (XCALL)**, donde el identificador de mensaje de la excepción recibida corresponde al identificador de función de la rutina según la tabla de definiciones de manejadores;

– al examinar la cola de mensajes recibidos, tanto cuando no se está ejecutando ninguna rutina, como cuando se encuentra una instrucción **yield (YIELD)**, donde el identificador de mensaje

del mensaje recibido corresponde al identificador de función de la rutina según la tabla de definiciones de manejadores o es una operación **run** de la MHEG-3 API dirigida a la rutina.

Los parámetros pueden pasarse a las rutinas por uno de los modos siguientes:

– por **value** (valor): se pasa a la rutina un valor del tipo de parámetro;

– por **reference** (referencia): se pasa a la rutina un identificador de datos que hace referencia a una variable global, variable dinámica o constante cuyo tipo es el mismo que el tipo del parámetro pasado a la rutina.

En ambos casos, el valor del parámetro pasado se vuelve el valor de la variable local cuyo índice corresponde al índice del parámetro. La variable local que corresponde a un parámetro pasado por referencia será del tipo **data identifier**.

Los identificadores de datos para variables locales no se pasarán por referencia.

### 8.3.2 Servicios

Los servicios son funciones externas proporcionadas por el entorno de ejecución, que pueden ser invocados por un guión intercambiado.

Los servicios serán declarados dentro del guión intercambiado, como parte de una declaración de lote, por:

– su firma;

– su nombre de operación global IDL.

No habrá más de 256 servicios declarados en cada lote de un guión intercambiado.

No habrá más de 192 lotes declarados en un guión intercambiado.

Un servicio puede ser invocado por una instrucción **external call (XCALL)**.

Los parámetros pueden pasarse a los servicios por uno de los modos siguientes:

– **in**: se pasa al servicio un identificador de datos que hace referencia a una variable o constante cuyo tipo es el mismo que el tipo del parámetro;

– **inout**: se pasa al servicio un identificador de datos que hace referencia a una variable cuyo tipo es el mismo que el tipo del parámetro; al retornar el servicio, la variable se actualiza con su nuevo valor;

– **out**: es igual que **inout**, pero el servicio no utiliza el valor de la variable.

### 8.3.3 Funciones predefinidas

Las funciones predefinidas corresponden a las operaciones de la interfaz del motor MHEG-3.

Las funciones predefinidas tienen identificadores de función predefinidos y, por tanto, no se declararán dentro de un guión intercambiado.

Dado que esta Recomendación no está específicamente ligada a una de las Recomendaciones UIT-T de la serie T.170 (y partes de ISO/CEI 13522), las operaciones MHEG API utilizadas para manipular objetos de MHEG no están definidas explícitamente. Sin embargo, esta Recomendación específica el procedimiento que deberá utilizarse para definir una interfaz MHEG API y para especificar la correspondencia de operaciones de esta MHEG API con identificadores de funciones predefinidas. Esto se describe en el anexo E.

Además, esta Recomendación define la MHEG-3 API, es decir, la interfaz que los motores MHEG-3 deben proporcionar para la manipulación de guiones. Esta interfaz se describe en la cláusula 15 y el anexo F. Esta interfaz puede utilizarse desde el interior de guiones y, por tanto, se ha hecho

corresponder con identificadores de funciones predefinidas. La lista de estas funciones predefinidas y sus identificadores se indica en el anexo C.

Se puede invocar funciones predefinidas y se les puede pasar parámetros utilizando el mismo mecanismo empleado para los servicios.

## 8.4 Mensajes

La MHEG-SIR define dos modalidades de mensajes:

– excepciones de lotes (véase 8.4.1);

– mensajes predefinidos (véase 8.4.2).

Todos los mensajes tendrán una firma (o prototipo) constituida por una lista ordenada de parámetros (miembros) formales definidos por su tipo.

Todos los mensajes son referenciados de una manera única e inequívoca por su identificador de mensaje.

### 8.4.1 Excepciones de lote

Las excepciones de lote se envían a un guión rt por el entorno de ejecución, como consecuencia de la invocación de un servicio por este guión rt.

Las excepciones de lote serán declaradas dentro del guión intercambiado, como parte de una declaración de lote, por:

– su firma;

– su nombre de excepción global IDL.

No habrá más de 256 excepciones declaradas en cada lote de un guión intercambiado.

### 8.4.2 Mensajes predefinidos

Los mensajes predefinidos enviados a un guión rt pueden ser uno de los siguientes:

– una excepción de la interfaz del motor MHEG-3 (es decir, la MHEG API), señalada por el motor MHEG-3 como consecuencia de la invocación de una función predefinida por el guión rt;

– un mensaje predefinido enviado como consecuencia de la invocación de una operación de la MHEG-3 API dirigida al guión rt;

– La excepción **InstructionExecutionError**, que se señala como consecuencia de un error producido durante la ejecución de una instrucción del guión rt.

Los mensajes predefinidos tienen identificadores de mensajes predefinidos y, por tanto, no serán declarados dentro de un guión intercambiado.

Puesto que esta Recomendación no está específicamente ligada a una de las partes de ISO/CEI 13522, las excepciones MHEG API no están definidas explícitamente. Sin embargo, esta Recomendación especifica el procedimiento que deberá utilizarse para definir una MHEG API y para especificar la correspondencia de excepciones de esta MHEG API con identificadores de mensajes predefinidos. Esto se describe en el anexo E.

El anexo C especifica cómo la **InstructionExecutionError** y los mensajes resultantes de operaciones MHEG-3 API se harán corresponder con identificadores de mensajes predefinidos.

## 8.5 Instrucciones

La parte de código de programa de rutinas consta de una secuencia de instrucciones. A diferencia del resto de un guión intercambiado, que se trata después de la preparación del guión, las instrucciones sólo hay que tratarlas después de creado un guión rt, cuando se activa la rutina a que pertenecen.

Una instrucción consistirá en un código de operación (código op) seguido de cero o más operandos. El número, tipo y codificación de los operandos está determinado íntegramente por el código op.

En general, los operandos son parte integrante de la instrucción, mientras que los valores de los parámetros se toman de la pila de parámetros.

El funcionamiento de la unidad de ejecución de instrucción se describe en la cláusula 9, y la semántica precisa de cada instrucción se describe en la cláusula 13.

## 8.6 Identificadores

Se utilizan identificadores para hacer referencia a entidades MHEG-SIR (es decir, tipos, datos, funciones y mensajes) de una manera inequívoca, a través de guiones intercambiados.

### 8.6.1 Identificadores de tipos

Los identificadores de tipo (TID, *type identifiers*) se codificarán en dos octetos de la manera siguiente:

– los tipos primitivos y los tipos predefinidos tendrán los TID predefinidos especificados en el anexo C;

– los tipos declarados cuyos índices (comenzando por 0) en la tabla de declaraciones de tipos es X, tendrán (X + 4000h) como TID.

En consecuencia:

– los TID entre 0 y 3FFFh referenciarán tipos predefinidos;

– los TID entre 4000h y 7FFFh referenciarán tipos declarados.

### 8.6.2 Identificadores de datos

Los identificadores de datos (DID, *data identifiers*) se codificarán en dos octetos de la manera siguiente:

– las constantes cuyo índice (comenzando por 0) en la tabla de declaraciones de constantes es X tendrán X como DID;

– las variables globales cuyo índice (comenzando por 0) en la tabla de declaraciones de variables globales es X tendrán (X + 1000h) como DID;

– las variables locales cuyo índice (comenzando por 0) en la tabla de declaraciones de variables locales es X tendrán (X + 8000h) como DID;

– las variables dinámicas tendrán DID que comenzarán por 8100h. El procedimiento para atribuir identificadores de datos a variables dinámicas no se especifica; los motores MHEG-3 pueden, por tanto, tener diferentes esquemas de atribución de identificadores.

En consecuencia:

– los DID entre 0 y 0FFFh referenciarán constantes;

– los DID entre 1000h y 7FFFh referenciarán variables globales;

– los DID entre 8000h y 80FFh referenciarán variables locales;

– los DID entre 8100h y FFFFh referenciarán variables dinámicas.

### 8.6.3 Identificadores de funciones

Los identificadores de funciones (FID, *function identifiers*) se codificarán en dos octetos de la manera siguiente:

– las rutinas cuyo índice (comenzando por 0) en la tabla de declaraciones de rutinas es X tendrán X como FID;

– las funciones predefinidas cuyo índice (comenzando por 0) en la tabla de funciones predefinidas es X tendrán (X + 1000h) como FID;

– los servicios cuyo índice (comenzando por 0) en una declaración de lote es X y cuyo índice de lote en la tabla de declaraciones de lotes es Y (comenzando por 0) tendrán (((Y + 64) << 8) + X) como FID.

En consecuencia:

– los FID entre 0 y 0FFFh referenciarán rutinas;

– los FID entre 1000h y 3FFFh referenciarán funciones predefinidas;

– los FID entre 4000h y FFFFh referenciarán servicios.

### 8.6.4 Identificadores de mensajes

Los identificadores de mensajes (MID, *message identifiers*) se codificarán en dos octetos de la manera siguiente:

– los mensajes predefinidos cuyo índice (comenzando por 0) en la tabla de mensajes predefinidos es X tendrán X como MID;

– las excepciones cuyo índice (comenzando por 0) en una declaración de lote es X y cuyo índice de lote (comenzando por 0) en la tabla de declaraciones de lotes es Y tendrán (((Y + 64) << 8) + X) como MID.

En consecuencia:

– los MID entre 0 y 3FFFh referenciarán mensajes predefinidos;

– los MID entre 4000h y FFFFh referenciarán excepciones de lotes.

## 9 La máquina virtual MHEG-SIR

Esta cláusula presenta la máquina virtual MHEG-SIR, es decir, el modelo de ejecución para el código MHEG-SIR.

### 9.1 Estructura de la máquina virtual MHEG-SIR

La máquina virtual MHEG-SIR es un conjunto de componentes lógicos, abstractos. La descripción de la máquina virtual MHEG-SIR tiene por finalidad aclarar la semántica operacional del código MHEG-SIR.

Un motor MHEG-3 tendrá el mismo comportamiento de interpretación para código MHEG-SIR que la máquina virtual descrita. Interpretará declaraciones e instrucciones MHEG-SIR de modo que produzcan efectos externos similares en todos los aspectos.

Sin embargo, esto no implica que se impongan requisitos a la tecnología u organización que puedan efectivamente utilizarse para implementar un motor MHEG-3. Un intérprete de guión dado no tiene que ser diseñado en la forma descrita para la máquina virtual, siempre que proporcione una funcionalidad equivalente.

La máquina virtual MHEG-SIR consta de:

– áreas de memoria (véase 9.3);

– unidades de procesamiento (véase 9.5).

Algunas áreas de memoria están asociadas con un guión mh por lo que son compartidas por todos los guiones rt creados a partir de aquél. Otras áreas de memoria están asociadas con cada guión rt.

La unidades de procesamiento se aplican a un solo guión rt. Sin embargo, un motor MHEG-3 puede ejecutar varios guiones rt al mismo tiempo. En este caso, mantendrá un contexto de ejecución separado para cada guión rt activo.

NOTA – En otras palabras, la máquina virtual MHEG-SIR es una máquina de un solo hilo de ejecución (*single-threaded*). Para obtener aplicaciones de muchos hilos (*multi-threaded*) hay que asociar cada hilo con un guión rt distinto.

## 9.2 Estructuras y notaciones

### 9.2.1 Tabla

Una tabla **T** está constituida por una matriz de entradas **T[i]** homogéneas a las que se puede ganar acceso mediante su índice **i**. Estas entradas tienen la misma estructura, pero no necesariamente el mismo tamaño. Las entradas constan de uno o varios campos **fld**. Algunas entradas pueden estar vacías. Los índices son identificadores MHEG-SIR, es decir, valores numéricos consecutivos comprendidos en una gama dada, que no comienza necesariamente por 0 en una tabla dada. El mecanismo de acceso subyacente (indexación secuencial, acceso directo, codificación hash, etc.) no se especifica. En la notación se utilizan los siguientes primitivos para expresar la manipulación de una tabla **T**:

– **T[i]** para ganar acceso a la entrada **i**;

– **T[i] = VAL** para asignar el valor **VAL** a la entrada **i**;

– **T[i].fld** para ganar acceso al campo **fld** de la entrada **i**;

– **T[i].fld = VAL** para asignar el valor **VAL** al campo **fld** de la entrada **i**.

### 9.2.2 Pila

Una pila consiste en una matriz de elementos homogéneos. Los elementos se insertan en la posición más alta (la cima) de la pila. En un momento dado cualquiera sólo se puede ganar acceso al elemento situado en la cima de la pila (es decir, al último elemento insertado). Cuando este elemento se retira de la pila, se pierde, y el elemento siguiente pasa a la cima de la pila. En la notación se utilizan los siguientes primitivos para expresar manipulación de la pila de llamadas **CS**:

– **CS.push(F)**: inserta la trama **F** en la cima de la pila, incrementa el registro de puntero de trama (FP);

– **CS.pop()**: decrementa FP, retira la trama de la cima de la pila, de lo cual retorna esta trama;

– **CS[FP]**: retorna el valor de la trama situada en la cima de la pila.

Al igual que en las tablas, se utiliza la notación "." para ganar acceso a campos de elementos en la pila.

### 9.2.3 Pila de parámetros

La pila de parámetros es un caso especial, porque ésta es una pila (no tipificada) constituida por un solo octeto, que se utiliza para almacenar valores tipificados. En la notación se utilizan los siguientes primitivos para expresar la manipulación de la pila de parámetros **PS**, donde **tid** es el identificador de tipo de un tipo primitivo, como se indica en el cuadro 1:

– **PS.push(VAL)**: inserta el valor **VAL** en la cima de la pila;

– **PS.pop(tid)**: retira el valor situado en la cima de la pila, cuyo identificador de tipo es **tid**, después de lo cual retorna este valor;

– **PS[SP](tid)**: retorna el valor en la cima de la pila, cuyo identificador de tipo es **tid**.

### 9.2.4 Cola

Una cola está constituida por una matriz de elementos homogéneos. Los elementos se insertan al final de la cola. En un momento dado cualquiera, sólo se puede ganar acceso al elemento que está al principio de la cola (al primer elemento insertado). Cuando este elemento es retirado de la cola, se pierde, y el elemento siguiente viene a situarse al principio de la cola. En esta notación se utilizan los siguientes primitivos para expresar la manipulación de la cola de mensajes **MQ**:

– **MQ.insert(M)**: inserta el valor de mensaje **M** al final de la cola;

– **MQ.remove()**: incrementa el registro de puntero de cola (QP, *queue pointer*), retira el elemento que está al principio de la cola, y retorna este elemento;

– **MQ[QP]**: retorna el valor del elemento que está al principio de la cola.

Al igual que en las tablas, se utiliza la notación "." para ganar acceso a campos de elementos en la cola.

### 9.2.5 Representación de datos

La representación de las estructuras y de los datos es independiente de la implementación. Los intérpretes de guión pueden representar cada valor de un tipo de datos con un número mínimo de octetos, pero no están obligados a seguir esta regla. En el cuadro 2 se indica este número mínimo de octetos.

**Cuadro 2/T.173 – Número mínimo de octetos para la representación de los valores de los tipos de datos**

| Tipo | Número mínimo de octetos para la representación del valor del tipo |
|---|---|
| octet | 1 |
| short | 2 |
| long | 4 |
| unsigned short | 2 |
| unsigned long | 4 |
| float | 4 |
| double | 8 |
| boolean | 1 |
| character | 2 (1 en el caso de juegos de caracteres restringidos) |
| data identifier | 2 |
| object reference | Depende de la implementación |
| string | Tamaño de carácter x (longitud de cadena + 1) |
| sequence | Tamaño de tipo de elemento x longitud de secuencia (número efectivo de elementos) + 2 |
| array | Tamaño de tipo de elemento x tamaño de matriz |
| structure | Suma de los tamaños de los tipos de elemento |

**Cuadro 2/T.173 – Número mínimo de octetos para la representación de los valores de los tipos de datos** *(fin)*

| Tipo | Número mínimo de octetos para la representación del valor del tipo |
|---|---|
| union | Tamaño del tipo de elemento de mayor tamaño + 1 |
| type identifier | 2 |
| function identifier | 2 |
| message identifier | 2 |
| package identifier | 1 |

NOTA – En esta notación no se distingue entre valores de longitud fija y valores de longitud variable. Los intérpretes de guión pueden almacenar valores de longitud variable en la región de la memoria designada por heap. Se utiliza **VT[i].val** para ganar acceso al valor de una variable aunque dicho valor pudiera estar, en realidad, almacenado en la tabla de variables como un asa en el montón (*heap*).

Cuando **VAL** es un valor de un tipo construido, la notación utilizada para el acceso a sus elementos es la siguiente:

– **VAL.tag**: rótulo de una unión;

– **VAL.val**: valor de una unión;

– **VAL[n]**: valor del n-ésimo elemento de una secuencia, cadena, estructura o matriz;

– **VAL.lg**: longitud efectiva de una secuencia o cadena.

Las semánticas de ejecución se expresan utilizando una sintaxis similar a la del lenguaje de programación C. Las expresiones escritas entra comillas simples indican el valor correspondiente, por ejemplo "void" indica TID de valor 0.

### 9.3 Áreas de memoria

En la máquina virtual MHEG-SIR se utilizan áreas de memoria para contener toda la información que se necesita para interpretar un determinado guión intercambiado.

Se puede asociar áreas de memoria con un guión mh (véase 9.3.1) o con un guión rt (véase 9.3.2).

#### 9.3.1 Áreas de memoria de guión mh

Las áreas de memoria de guión mh deben llenarse completamente en el momento de la carga del programa, es decir, durante la inicialización de un guión mh. Estas áreas podrán ser utilizadas por todos los guiones rt creados a partir de este guión mh. Las áreas de memoria de guión mh no se modificarán en la fase de ejecución y subsistirán mientras no se destruya el guión mh, a menos que se especifique otra cosa (por ejemplo, para la tabla de definiciones de lotes). Las áreas de memoria de guión mh comprenden:

– áreas de datos (véase 9.3.1.1);

– áreas de código (véase 9.3.1.2).

##### 9.3.1.1 Áreas de datos

Las áreas de datos se utilizan para almacenar las definiciones y valores de los datos globales del guión. Las áreas de datos comprenden:

– la tabla de definiciones de tipos (TT, *type definition table*) (véase 9.3.1.1.1);

– la tabla de constantes (CT, *constant table*) (véase 9.3.1.1.2);
– la tabla de definiciones de variables globales (GT, *global variable definition table*) (véase 9.3.1.1.3).

#### 9.3.1.1.1 Tabla de definiciones de tipo

La tabla de definiciones de tipos hace corresponder todos los tipos definidos del guión, representados por identificadores de tipo, con su descripción:

– **TT[TID].val**: descripción del tipo.

NOTA – La representación utilizada para la descripción del tipo no se especifica; sin embargo, debe permitir verificar fácilmente si un valor pertenece a un tipo.

#### 9.3.1.1.2 Tabla de constantes

La tabla de constantes hace corresponder todas las constantes del guión, representadas por identificadores de datos, con su tipo y valor:

– **CT[DID].TID**: tipo de la constante (expresado como un identificador de tipo);
– **CT[DID].val**: valor de la constante (que depende de su tipo).

#### 9.3.1.1.3 Tabla de definiciones de variables globales

La tabla de definiciones de variables globales hace corresponder todas las variables globales del guión, representadas por identificadores de datos, con su tipo y su valor inicial:

– **GT[DID].TID**: tipo de la variable global (expresado como un identificador de tipo);
– **GT[DID].val**: valor inicial de la variable global (que depende de su tipo).

### 9.3.1.2 Áreas de código

Se utilizan áreas de código para almacenar las direcciones y el código de programa de las funciones del guión. Las áreas de código comprenden:

– la tabla de definiciones de rutinas (RT, *routine definition table*) (véase 9.3.1.2.1);
– la tabla de definiciones de lotes (PT, *package definition table*) (véase 9.3.1.2.2);
– la tabla de definiciones de servicios (ST, *service definition table*) (véase 9.3.1.2.3);
– la tabla de definiciones de excepciones (XT, *exception definition table*) (véase 9.3.1.2.4);
– la tabla de definiciones de manejadores (HT, *handler definition table*) (véase 9.3.1.2.5);
– el área de código de programa, constituida por la secuencia de instrucciones de cada rutina (véase 9.3.1.2.6).

#### 9.3.1.2.1 Tabla de definiciones de rutinas

La tabla de definiciones de rutinas hace corresponder todas las rutinas del guión, representadas por identificadores de funciones, con su descripción de firma, su declaración de variables locales y su código de programa:

a) **RT[FID].TID**: tipo de valor de retorno (expresado como un identificador);
b) **RT[FID].nbp**: número de parámetros;
c) **RT[FID].sig**: descripción de la firma, donde:
   1) **RT[FID].sig[i].TID** es el tipo (expresado como un identificador de tipo) del i-ésimo parámetro;
   2) **RT[FID].sig[i].mod** es el modo de paso de los parámetros (**value** o **reference**) del i-ésimo parámetro;

d) **RT[FID].LT**: declaración de las variables locales de la rutina (cuyos **nbp** primeros elementos son los parámetros efectivos de la rutina);

e) **RT[FID].IP**: puntero a la primera instrucción en el código de la rutina.

Las variables locales utilizadas para contener parámetros pasados por **reference** tendrán **data identifier** como tipo, mientras que las variables locales utilizadas para contener parámetros pasados por **value** tendrán el mismo tipo que figura en la descripción de la firma para el parámetro correspondiente.

#### 9.3.1.2.2 Tabla de definiciones de lotes

La tabla de definiciones de lotes hace corresponder todos los lotes definidos del guión, representados por identificadores de lotes (PID, *package identifiers*) declarados por la tabla de declaraciones de lotes de la MHEG-SIR, con nombres de lotes e información adicional:

– **PT[PID].name**: nombre del lote;

– **PT[PID].nbf**: número de servicios en el lote;

– **PT[PID].nbm**: número de excepciones definidas por el lote;

– **PT[PID].sts**: estado actual del lote [**unchecked** (no comprobado), **available** (disponible), **ready** (listo), **opened** (abierto)];

– **PT[PID].or**: referencia de objeto inicial del lote.

Un lote está inicialmente en el estado **unchecked** (no comprobado). Pasa al estado **available** (disponible) cuando el procedimiento **package availability** (disponibilidad de lote) haya sido realizado con éxito. Después pasa al estado **ready** (listo) cuando se ha efectuado con éxito el procedimiento **package load** (carga de lote). Finalmente, pasa al estado **opened** (abierto) cuando hay una referencia de objeto inicial válida al lote, almacenada en el campo **PT[PID].or**, para uso por ulteriores invocaciones de servicio.

Como excepción a la regla indicada en 9.3.1,

– Los campos **PT[PID].sts** pueden ser modificados durante la ejecución, cada vez que cambia el estado de un lote;

– Los campos **PT[PID].or** pueden ser modificados durante la ejecución cuando se carga un lote.

#### 9.3.1.2.3 Tabla de definiciones de servicios

La tabla de definiciones de servicios hace corresponder todos los servicios externos definidos del guión, representados por identificadores de funciones MHEG-SIR, con su descripción de firma y con su nombre de operación global IDL:

a) **ST[FID].TID**: tipo de valor de retorno (expresado como un identificador de tipo);

b) **ST[FID].syn**: modo de llamada **synchronous** (síncrono), **asynchronous** (asíncrono);

c) **ST[FID].nbp**: número de parámetros;

d) **ST[FID].sig**: descripción de la firma donde:

   1) **ST[FID].sig[i].TID** es el tipo (expresado por un identificador de tipo) del i-ésimo parámetro;

   2) **ST[FID].sig[i].mod** es el modo de paso de los parámetros (**in, inout** o **out**) del i-ésimo parámetro;

e) **ST[FID].name**: el nombre global IDL de la operación invocada por el servicio.

La especificación de correspondencia específica de plataforma IDL se utilizará para establecer la correspondencia de **ST[MID].name** con un nombre específico de la plataforma.

#### 9.3.1.2.4 Tabla de definiciones de excepciones

La tabla de definiciones de excepciones hace corresponder todos los mensajes definidos del guión intercambiado, representados por identificadores de mensajes, con su descripción de firma y con su nombre de excepción global IDL:

– **XT[MID].name**: el nombre global IDL de la excepción que ocasiona el mensaje;

– **XT[MID].nbm**: número de parámetros;

– **XT[MID].sig**: descripción de la firma, donde **XT[MID].sig[i].TID** es el tipo (expresado como un identificador de tipo) del i-ésimo miembro.

La especificación de correspondencia específica de plataforma IDL se utiliza para establecer la correspondencia de **XT[MID].name** con un nombre específico de plataforma.

#### 9.3.1.2.5 Tabla de definiciones de manejadores

La tabla de definiciones de manejadores hace corresponder mensajes, representados por identificadores de mensajes, con rutinas representadas por identificadores de funciones:

– **HT[MID].FID**: identificador de rutina que habrá de invocarse para el tratamiento del mensaje.

Si un mensaje corresponde a una rutina en la tabla de manejadores (*handler table*), la firma de esta rutina debe concordar con la firma de este mensaje. La concordancia de las firmas se comprobará durante la carga, y se rechazarán las entradas no concordantes.

La tabla de definiciones de manejadores es utilizada por la unidad de ejecución de guión rt. Cuando la unidad de ejecución de guión rt retira un mensaje de la cola de mensajes, invoca la rutina que corresponde al mensaje, con los parámetros del mensaje como sus parámetros.

#### 9.3.1.2.6 Area de código de programa

Una instrucción está constituida por un código de operación (código op) de un octeto seguido de cero a tres octetos para operandos. El código op determina completamente el número y la longitud de sus operandos, de acuerdo con la tabla de instrucciones. Tanto los códigos op como los operandos se codifican de una manera optimizada para facilitar la conmutación.

NOTA – Un intérprete de guión (especialmente en máquinas de 32 bits) puede alinear las instrucciones en la fase de carga, es decir, insertar octetos de relleno de modo que cada instrucción esté representada por cuatro octetos; esto facilita la incrementación del puntero de instrucción. Otra posible solución es que el intérprete de instrucciones deje las instrucciones empacadas (*packed*), y determinar en la fase de ejecución el número de octetos a incrementar.

### 9.3.2 Areas de memoria de guión rt

Las áreas de memoria de guión rt se inicializan cuando se crea el guión rt y pueden ser modificadas durante su ejecución. Las áreas de memoria de guión rt comprenden:

– áreas de memoria dinámicas (véase 9.3.2.1);

– registros (véase 9.3.2.2).

#### 9.3.2.1 Areas de memoria dinámica

Las áreas de memoria dinámica se utilizan para representar los datos y el contexto de ejecución actual del guión rt.

Las áreas de memoria dinámica comprenden:

– la tabla de variables (VT, *variable table*) (véase 9.3.2.1.1);

– la pila de llamadas (CS, *call stack*) (véase 9.3.2.1.2);

– la pila de parámetros (PS, *parameter stack*) (véase 9.3.2.1.3);
– la cola de mensajes (MQ, *message queue*) (véase 9.3.2.1.4);
– la memoria montón (*heap*) (véase 9.3.2.1.5).

#### 9.3.2.1.1 Tabla de variables

La tabla de variables hace corresponder las variables del guión rt, representadas por identificadores de datos, con su tipo y su valor actual:

– **VT[DID].TID**: tipo de la variable (expresado como un identificador de tipo);
– **VT[DID].val**: valor actual de la variable (que depende de su tipo).

La tabla de variables se inicializa cuando se crea el guión rt. Consta de dos subtablas:

– una copia de la tabla de variables globales asociada con el guión mh;
– la tabla de variables locales de la rutina que se está ejecutando en ese momento.

Los campos **VT[DID].val** se modifican cada vez que se asigna una variable al ejecutarse una instrucción de asignación de variable.

Cuando la rutina actual es una nueva rutina (tras la ejecución de una instrucción **CALL**, **RET** o **YIELD**), la tabla de variables locales de la antigua rutina se almacena, y se substituye en la VT por la tabla de variables locales de la nueva rutina. La primera entrada de una tabla de variables locales está formada por los parámetros pasados a la función.

#### 9.3.2.1.2 Pila de llamadas

La pila de llamadas se utiliza para almacenar el contexto de invocación actual.

La pila de llamadas es una matriz de tramas de llamada. Cada trama de llamada corresponde al contexto en el momento de la invocación de una función activa (rutina, función externa o acción MHEG). Las tramas se almacenarán en la pila CS en el orden de invocación. La trama que está en la cima de la pila, si hay alguna, describirá el contexto de ejecución de la rutina que invocó la función que se está ejecutando en ese momento.

Cada trama consta de los siguientes elementos:

– **CS[i].FID**: identificador de función del llamante;
– **CS[i].IP**: puntero a la instrucción que habrá de retornarse después de retornada la función actual;
– **CS[i].LT**: tabla de variables locales del llamante (en el momento de la invocación);
– **CS[i].SP**: puntero a la cima de la pila de parámetros (en el momento de la invocación).

El campo LT de una trama de llamada tendrá la estructura de una tabla de variables:

– **CS[i].LT[DID].TID**: identificador de tipo de la variable cuyo identificador es **DID**;
– **CS[i].LT[DID].val**: valor de la variable.

La pila de llamadas es modificada por ciertas instrucciones de flujo de control. Inicialmente, la pila de llamadas deberá estar vacía. Cuando se invoca una función, se inserta en la pila de llamadas una trama que describe esta llamada. Cuando se retorna de una función, esta trama será sacada de la pila de llamadas. La dirección de la trama situada en la cima de la pila de llamadas estará almacenada en todo momento en el registro de puntero de trama (FP).

#### 9.3.2.1.3 Pila de parámetros

La pila de parámetros se utiliza para almacenar los parámetros y valores de retorno de las instrucciones. La pila de parámetros es una matriz de valores de datos. El tipo del valor de datos se determina por la secuencia de operaciones como resultado de la cual se inserta el valor en la pila.

La pila de parámetros la utiliza la unidad de ejecución de instrucción MHEG-SIR. Inicialmente, la pila de parámetros deberá estar vacía. Esta pila es modificada por la mayor parte de las instrucciones (operadores aritméticos, operadores lógicos, operadores de comparación, manipulación de pila, asignación de variable, saltos condicionales, llamadas). Cuando se ejecuta una instrucción, ésta tomará sus parámetros sacándolos de la pila de parámetros e insertará su valor de retorno en la pila de parámetros. La dirección de la trama que ocupa la cima de la pila de parámetros estará almacenada en todo momento en el registro de puntero de pila (SP, *stack pointer*).

#### 9.3.2.1.4 Cola de mensajes

La cola de mensajes se utiliza para almacenar en memoria tampón los mensajes que recibe el intérprete de guión. Cada ítem en la cola consta de los siguientes elementos:

– **MQ[i].MID**: identificador de mensaje;

– **MQ[i].LT**: lista de parámetros del mensaje.

El campo LT de un ítem en la cola de mensajes tendrá la estructura de una tabla de variables:

– **MQ[i].LT[j].TID**: identificador de tipo del j-ésimo parámetro;

– **MQ[i].LT[j].val**: valor del j-ésimo parámetro.

El intérprete de guión insertará asíncronamente los mensajes en la cola de mensajes a medida que éstos vayan siendo generados en el entorno externo. La cola de mensajes será procesada por la unidad de ejecución del guión rt cuando se dé una de estas dos situaciones:

– el guión rt no está funcionando, es decir, no se está ejecutando actualmente ninguna rutina;

– se encuentra una instrucción **YIELD**.

El principio de la cola de mensajes (es decir, el mensaje que está al principio de la cola y será próximo que se sacará de la cola) estará almacenado en todo momento en el registro de puntero de cola (QP). Inicialmente la cola de mensajes deberá estar vacía.

#### 9.3.2.1.5 Memoria montón (*heap*)

La región de la memoria designada por montón (*heap*) se utiliza para almacenar variables dinámicas, representadas por identificadores de datos, como su tipo y su valor actual:

– **VT[DID].TID**: tipo de la variable (expresado como un identificador de tipo);

– **VT[DID].val**: valor actual de la variable (que depende de su tipo).

Las variables dinámicas son referenciadas por asas de un tipo opaco, cuya representación no se especifica. Los identificadores de datos se han hecho corresponder internamente con estas asas, de manera que se gana acceso a las variables dinámicas de la misma manera que a otras variables.

Los campos **VT[DID].val** se modifican cada vez que se asigna una variable al ejecutarse una instrucción de asignación de variable.

Incumbe a la aplicación la atribución (*allocation*) y la anulación de la atribución (*de-allocation*) explícitas de variables dinámicas mediante las instrucciones **ALLOC** y **FREE**.

NOTA – Los intérpretes de guiones pueden utilizar también el montón para almacenar los valores de variables globales o locales de tipo longitud variable. En este caso se almacena en la tabla un asa en el montón, en vez del dato propiamente dicho.

#### 9.3.2.2 Registros

Los registros (*registers*) contienen estados específicos de la máquina virtual y tienen que ser modificados frecuentemente durante la ejecución de un guión rt.

La máquina virtual MHEG-SIR mantiene los siguientes registros:

– registro de puntero de instrucción (IP, *instruction pointer*) o contador de programa (véase 9.3.2.2.1);
– registro de puntero de trama (FP, *frame pointer*) (véase 9.3.2.2.5);
– registro de puntero de pila (SP) (véase 9.3.2.2.4);
– registro de puntero de cola (QP) (véase 9.3.2.2.6);
– registro de instrucción (IR, *instruction register*) (véase 9.3.2.2.2);
– registro de error (ER, *error register*) (véase 9.3.2.2.3);
– registro de función (FR, *function register*) (véase 9.3.2.2.7).

La representación de los datos contenidos en un registro de puntero no está especificada. Todos los registros se iniciarán a un valor nulo cuya representación no está especificada.

##### 9.3.2.2.1 Registro de puntero de instrucción

El registro IP apunta a la próxima instrucción que habrá de ejecutarse dentro del código de programa de una rutina. Este registro será modificado por la unidad de ejecución del guión rt y la unidad de ejecución de instrucción MHEG-SIR como parte de la ejecución de instrucciones.

##### 9.3.2.2.2 Registro de instrucción

El registro IR contiene el código para la instrucción que se está ejecutando en ese momento. Será actualizado por la unidad de ejecución de guión rt cada vez que se cargue una nueva instrucción, y la unidad de ejecución de instrucción MHEG-SIR ganará acceso a este registro.

NOTA – No es necesario que el registro IR tenga una longitud superior a cuatro octetos; sin embargo, el tamaño que efectivamente pueda tener no está especificado.

##### 9.3.2.2.3 Registro de error

El registro ER contiene el código del último error encontrado durante la ejecución de una instrucción. Este registro será actualizado por la unidad de ejecución de instrucción MHEG-SIR cada vez que encuentre un error. El valor nulo indica que hasta ese momento no se ha encontrado ningún error durante la ejecución del guión rt.

Los códigos de error están predefinidos. Los códigos de error señalados por cada instrucción se definen en la cláusula 13.

Cuando se señala un error durante la ejecución de una instrucción, se fijará el registro ER a un valor no nulo y se señalará la excepción **InstructionExecutionError**. Como resultado de esto se inserta el correspondiente mensaje en la cola de mensajes.

##### 9.3.2.2.4 Registro de puntero de pila

El registro de puntero de pila (SP) apunta a la cima de la pila de parámetros. El valor de este registro se actualizará por la unidad de ejecución de instrucción MHEG-SIR de la manera siguiente:

– se incrementará cada vez que se inserta un dato en la pila;
– se decrementará cada vez que se saca un dato de la pila de parámetros.

#### 9.3.2.2.5 Registro de puntero de trama

El registro de puntero de trama (FP) apunta a la trama situada en la cima de la pila de llamadas. El valor de este registro será actualizado por la unidad de ejecución de instrucción MHEG-SIR de la manera siguiente:

– se incrementará cada vez que se llama a una función;

– se decrementará cada vez que se retorna de una función.

#### 9.3.2.2.6 Registro de puntero de cola

El registro de puntero de cola (QP) apunta al próximo mensaje que habrá de sacarse de la cola de mensajes. El intérprete de guión decrementará el valor de este registro cada vez que se retira un mensaje de la cola.

#### 9.3.2.2.7 Registro de función

El registro de función (FR) contiene el FID de la función que se está ejecutando en ese momento. El intérprete de guión actualizará el valor de este registro cada vez que se llama a una función o se retorna de una función.

## 9.4 Estados de los guiones

### 9.4.1 Estados de guión mh

El estado de un guión mh será **available** (disponible) o **not available** (no disponible).

#### 9.4.1.1 No disponible

El estado de un guión mh deberá ser **not available** (no disponible) si se encuentra en uno de estos dos casos:

– la inicialización de guión mh (esto es, el efecto de la operación **prepare** de la MHEG-3 API) no se ha realizado sobre este guión mh;

– la destrucción de guión mh (es decir, el efecto de la operación **destroy** de la MHEG-3 API) se ha pedido sobre este guión mh.

#### 9.4.1.2 Disponible

El estado de un guión mh será **available** (disponible) si la inicialización de guión mh se ha realizado con éxito sobre este guión mh y si todavía no se ha solicitado la destrucción de guión mh.

Esto implica que:

– el guión intercambiado ha sido analizado y las áreas de memoria de guión mh han sido debidamente completadas, en consecuencia;

– los lotes referenciados en el guión mh están disponibles y han sido cargados de acuerdo con el procedimiento **package load** (carga de lote).

### 9.4.2 Estados de guión rt

El estado de un guión rt será uno de los siguientes: **not ready** (no listo), **ready** (listo), **running** (en funcionamiento), **erroneous** (erróneo).

#### 9.4.2.1 No listo

El estado de un guión rt será **not ready** (no listo) si se encuentra en uno de estos dos casos:

– la inicialización de guión rt (es decir, el efecto de la operación **new** de la MHEG-3 API) no se ha realizado sobre este guión rt;

– la destrucción de guión rt (es decir, el efecto de la operación **delete** de la MHEG-3 API) se ha pedido sobre este guión rt.

El estado de un guión rt es inicialmente **not ready**. Por otra parte, pasa a **not ready** cuando se invoca una operación **delete** sobre este guión rt.

#### 9.4.2.2 Listo

El estado de un guión rt será **ready** (listo) si se cumplen todas las condiciones siguientes:

– la inicialización de guión rt se ha realizado con éxito sobre este guión mh;

– la destrucción de guión rt no se ha pedido todavía sobre este guión rt;

– el registro IP se fija a "null" es decir, no se está ejecutando ninguna rutina en este momento;

– el registro ER se fija a "null".

Esto implica que la pila de llamadas, la cola de mensajes y la pila de parámetros están vacías.

Sin embargo, los valores de las variables globales no tienen que ser los mismos que los valores iniciales; cuando un guión rt no tiene más ninguna instrucción por ejecutar, y no hay mensajes en la pila de mensajes, retorna al estado **ready**.

El estado de un guión rt conmuta:

– de **not ready** a **ready**, cuando se invoca una operación **new** sobre este guión rt;

– de **running** a **ready**, cuando la unidad de ejecución de guión rt ya no tiene más instrucciones por ejecutar, o como resultado de la invocación de una operación **stop** o **reinit**;

– de **erroneous** a **ready**, como resultado de la invocación de una operación **stop** o **reinit**.

#### 9.4.2.3 En funcionamiento

El estado de un guión rt será **running** (en funcionamiento) si se cumplen todas las condiciones siguientes:

– la inicialización de guión rt se ha efectuado sin error sobre este guión mh;

– la destrucción de guión rt no se ha pedido todavía sobre este guión rt;

– el registro IP no contiene el valor "null", es decir, se está ejecutando una rutina en este momento;

– el registro ER está fijado a "null".

El estado de un guión rt conmuta de **ready** a **running** cuando hay un mensaje en la cola de mensajes y se activa la unidad de ejecución de guión rt. Esto puede producirse como resultado de la invocación de una operación **run**.

#### 9.4.2.4 Erróneo

El estado de un guión rt será **erroneous** (erróneo) si el registro ER no contiene el valor "null", es decir, si se ha producido un error durante la ejecución del guión rt.

El estado de un guión rt conmuta de **running** a **erroneous** cuando la unidad de ejecución de instrucción de guión rt señala un error de ejecución de instrucción.

## 9.5 Unidades de procesamiento

Esta subcláusula describe el flujo de control de la máquina virtual MHEG-SIR y la semántica de instrucciones.

Para los fines de la descripción de la máquina virtual, se supone que el proceso principal del intérprete de guión funciona en paralelo con todas las unidades de ejecución de guión rt activas. La calendarización de las diferentes tareas no está especificada.

### 9.5.1 Recepción de mensajes

El proceso principal del intérprete de guión recibe y trata eventos. Si no hay ningún evento por tratar, dicho proceso está en reposo (*idle*). Los eventos recibidos por el intérprete de guión pueden ser:

– invocaciones de operaciones MHEG-3 API;

– mensajes que corresponden a la aparición de una excepción señalada como resultado de la invocación de un servicio o de una función predefinida.

#### 9.5.1.1 Operaciones MHEG-3 API

Las operaciones MHEG-3 API pueden ser invocadas por una unidad de ejecución de guión rt, por otro componente del motor MHEG-3 o por procesos externos con respecto al motor MHEG-3.

Cuando se invoca una operación MHEG-3 API, el proceso principal actuará como se especifica por la semántica de la MHEG-3 API, descrita en la cláusula 15.

#### 9.5.1.2 Excepción externa

Cuando un mensaje proveniente de un intérprete de acción (excepción MHEG API) o del entorno de ejecución está destinado a un guión rt, entonces, si este mensaje corresponde realmente a una excepción señalada por la MHEG API o el entorno de ejecución como consecuencia de la invocación de una operación resultante de una instrucción **XCALL** por este guión rt, el proceso principal analizará los parámetros de la excepción y construirá una estructura de mensaje que estará constituida por el identificador de mensaje de la excepción seguido de sus miembros efectivos (comenzando por la referencia de objeto del objeto de origen). Entonces:

– si la excepción es el resultado de la invocación de una operación síncrona que se está ejecutando en ese momento, el proceso principal pedirá a la unidad de ejecución de guión rt que termine la instrucción **XCALL** (para lo cual saca su trama de la pila de llamadas) sin buscar parámetros de salida ni un valor de retorno, y que, inmediatamente después de esto, inicie la rutina que corresponde al identificador de mensaje de la excepción, con los miembros de la excepción como los parámetros reales de la rutina; esto produce el mismo efecto que si esta rutina hubiera sido invocada por una instrucción **CALL**;

– si la excepción es el resultado de la invocación de una operación síncrona terminada anteriormente (sea con éxito o no), el proceso principal no tendrá en cuenta la excepción;

– si la excepción es el resultado de la invocación de una operación asíncrona, el proceso principal insertará el mensaje construido en la cola de mensajes del guión rt de destino.

#### 9.5.1.3 Excepción **InstructionExecutionError**

Cuando en la ejecución de una instrucción se señala la excepción interna **InstructionExecutionError**, el proceso principal construirá una estructura de mensaje que estará formada por el identificador de mensaje que corresponde a esta excepción, seguido por un miembro fijado al valor del registro de error, y la insertará en la cola de mensajes del guión rt en cuya ejecución se señaló la excepción.

#### 9.5.1.4 Excepción MHEG-3 API

Cuando se retorna a un guión rt una excepción resultante de la invocación de una operación MHEG-3 API, el proceso principal construirá una estructura de mensaje que estará formada por el identificador de mensaje que corresponde a la excepción, seguido de sus miembros, y la insertará en la cola de mensajes del guión rt que invocó la operación.

### 9.5.2 Inicialización de guión mh

Cuando se invoca la operación **prepare** de la MHEG-3 API, el intérprete de guión ganará acceso al tren (*stream*) o fichero (*file*) por medio del identificador de sistema proporcionado y analizará el guión. Seguidamente, el intérprete de guión:

– analizará la parte de declaraciones e inicializará las tablas CT, GT, TT, RT, ST, PT, XT, HT y el campo **RT[i].LT** para cada rutina **i**; esto incluye las verificaciones apropiadas [verificación del manejador, procedimiento **package availability** (disponibilidad de lote)];

– analizará la estructura de la parte de instrucciones para llenar el área de código de programa de cada rutina;

– aplicará el procedimiento **package load** (carga de lote), estableciendo enlaces estáticos con lotes de acuerdo con la especificación de correspondencia de plataforma;

– se pone el guión mh en el estado **available**.

NOTA – La especificación de correspondencia de plataforma deberá definir la semántica de la carga de lotes. El motor MHEG-3 puede encargarse de optimizar su estrategia de gestión de recursos, por ejemplo, descargando lotes temporalmente para liberar memoria, o cargando lotes solamente cuando se crean guiones rt o incluso cuando se invocan servicios.

### 9.5.3 Inicialización de guión rt

Cuando se invoca la operación **new** MHEG-3 API, el intérprete de guión creará un contexto para el guión rt de destino, es decir, el intérprete de guión:

– inicializará las áreas de memoria dinámica;

– inicializará todos los registros al valor nulo;

– creará una unidad de ejecución de guión rt para el guión rt;

– pondrá el guión rt en el estado **ready**.

### 9.5.4 Unidad de ejecución de guión rt

Cuando la unidad de ejecución de guión rt está activada, y si no se le pide que detenga el guión rt actual, realizará lo siguiente:

```
rt-script-execution-unit ()
{
FID fid = 'null';
if (IP == 'null')                              // no next instruction
{
        while (fid == 'null')
        {
                if (QP == 'null') then exit;                    // return
                fid= HT[MQ[QP].MID].FID;            // find handler for message
                if (fid != 'null') then                 // handler found
                {
                        CS.push({IP, FR, SP, 'null'});      // stack routine call
                        FR = fid;
                        IP = RT[FR].IP;      // branch to start of routine
                }
                MQ.remove();                    // remove message
        }
}
// endif

while (IP != 'null'):
{
```

```
        IR = *IP++;                 // load next instruction and increment program counter
        instruction-execution-unit();       // call the MHEG-SIR instruction execution unit
}
// endwhile
return;                                     // return to script interpreter
}
```

### 9.5.5 Unidad de ejecución de instrucción MHEG-SIR

Cuando haya sido llamada por la unidad de ejecución guión rt, la unidad de ejecución de instrucción MHEG-SIR de un guión rt decodificará el código op contenido en el primer octeto del registro IR, después interpretará la instrucción que corresponde a este código op como se especifica en la cláusula 13, y por último retornará.

La unidad de ejecución de instrucción saca de la pila de parámetros los parámetros que se utilizan para ejecutar la instrucción (si existen). Seguidamente, inserta en la pila de parámetros los parámetros obtenidos como resultado de la instrucción (si existen).

El cuadro 3 recapitula los efectos de las instrucciones sobre los diversos elementos de la máquina virtual MHEG-SIR, en la forma definida por esta cláusula.

## 10 Disposiciones para el acceso al entorno de ejecución

Esta cláusula describe los mecanismos definidos por esta Recomendación para hacer posible que los guiones rt ganen acceso a funciones externas proporcionadas por el entorno de ejecución en la plataforma de ejecución, e intercambien datos con dichas funciones.

### 10.1 Modelo general

La interfaz proporcionada por el soporte lógico externo en el entorno de ejecución debe declararse en el guión intercambiado como parte de su declaración de lote, de modo que el intérprete de guión sepa como ganar acceso a esta interfaz cuando el guión la invoque.

Una declaración de lote describe un conjunto de servicios (es decir, funciones externas) por su firma, es decir, el tipo y el modo de paso de cada parámetro.

La MHEG-SIR especifica como debe expresarse en guiones intercambiados la manera de invocar funciones externas, pasar parámetros, obtener valores de retorno y tratar excepciones.

Esta Recomendación especifica asimismo cómo estas expresiones deberán ser interpretadas por los motores MHEG-3.

Esta Recomendación trata también el intercambio (es decir, llamada de función, paso de parámetros, obtención de valor de retorno y tratamiento de excepciones) entre un motor MHEG-3 y el entorno de ejecución. Con este fin, la presente Recomendación especifica cómo un soporte lógico externo debe proporcionar a los motores MHEG-3 el acceso a estas funciones. Tal convenio, denominado especificación de correspondencia de plataforma, depende de la plataforma de ejecución.

Las especificaciones de correspondencia de plataforma conformes con las disposiciones de esta Recomendación deben registrarse para garantizar la interoperabilidad de los servicios del entorno de ejecución con cualquier motor MHEG-3 conforme, en esta plataforma. Si existe, para la plataforma, una especificación de correspondencia de plataforma, los motores MHEG-3 deberán ser conformes con esta especificación de correspondencia de plataforma para poder ganar acceso a servicios del entorno de ejecución.

Las implementaciones de motores MHEG-3 indicarán, en su documento de conformidad, la especificación o especificaciones de correspondencia de plataforma con las que son conformes.

NOTA – Si el soporte lógico existente no cumple con la especificación de correspondencia de plataforma y es necesario que los guiones MHEG-SIR ganen acceso al mismo, podrá ser incorporado en una interfaz que traduzca sus propios convenios de interfaz a los convenios de interfaz de la especificación de correspondencia de plataforma.

### 10.2 Declaración de interfaces IDL

La interfaz de soporte lógico externo destinado a ser utilizado por un guión intercambiado puede contener:

– declaraciones de operaciones;

– declaraciones de excepciones;

– declaraciones de tipos.

Los tipos deberán declararse en la declaración de tipo de este guión intercambiado.

Las operaciones y excepciones se declararán en la declaración de lote de este guión intercambiado. A esta declaración de lote se asignará un identificador de lote y estará constituida por:

– el nombre del lote;

– un conjunto de descripciones de servicios;

– un conjunto de descripciones de excepciones.

A las descripciones de servicios se asignará un identificador de función y estarán constituidas por:

– el nombre de la operación;

– la firma de la función, es decir, el tipo y el modo de paso de cada parámetro y el tipo del valor de retorno.

A las descripciones de excepciones se asignará un identificador de mensaje y estarán constituidas por:

– el nombre de la excepción;

– la firma de la excepción, es decir, el tipo de cada miembro.

Los guiones MHEG-SIR utilizan identificadores (de lotes, de tipos, de funciones) para hacer referencia a tipos y funciones; un identificador de función para una operación externa puede construirse a partir de un identificador de lote y del índice de la declaración de servicio en este lote, mientras que un identificador de mensaje para una excepción externa puede construirse a partir de un identificador de lote y del índice de la declaración de excepción en este lote.

El intérprete de guión utilizará nombres (de lotes, de operaciones, de excepciones) para enlazar con la implementación efectiva del soporte lógico externo.

Una declaración de lote MHEG-SIR se sitúa en el mismo nivel de abstracción que una especificación IDL. Esta Recomendación define las reglas para establecer la correspondencia de una especificación IDL con una declaración de lote. La cláusula 14 especifica:

– cómo una descripción de tipo de datos IDL deberá corresponder a una descripción de tipo de datos MHEG-SIR;

– cómo una descripción de operación IDL deberá corresponder a una descripción de servicio MHEG-SIR;

– cómo una descripción de excepción IDL deberá corresponder a una descripción de excepción MHEG-SIR.

### 10.3 Invocación de operaciones externas en un programa MHEG-SIR

Un servicio descrito en una declaración de lote se invocará a partir de un programa MHEG-SIR de la manera siguiente:

– las variables de tipos esperados que corresponden al valor de retorno (si existe) y a cada parámetro se declararán dentro del guión intercambiado (excepto la referencia de objeto del objeto de origen, que será implícita).

– el programa asignará las variables que corresponden a parámetros de entrada o de entrada/salida;

– el programa insertará en la pila los identificadores de datos de todas estas variables en el orden de derecha a izquierda [el identificador de la variable que corresponde al valor de retorno se inserta primero, después los parámetros reales, y por último la referencia de objeto (parámetro implícito)] de la operación deseada;

– el programa invocará la operación utilizando una instrucción **external call (XCALL)** con el identificador de función de la operación invocada como operando;

– el programa se servirá de los resultados de la función, para lo cual utilizará las variables correspondientes al valor de retorno, a los parámetros **inout** y a los parámetros **out**.

### 10.4 Tratamiento de excepciones externas en un programa MHEG-SIR

Una excepción descrita en una declaración de lote deberá ser tratada por un programa MHEG-SIR de la manera siguiente:

– las variables de tipos esperados que corresponden a cada miembro deberán declararse dentro del guión intercambiado (salvo la referencia de objeto del objeto de origen, que deberá ser implícita);

– una rutina cuyos parámetros corresponden a los miembros de la excepción deberá declararse dentro de la parte de declaración de rutina del guión intercambiado;

– la relación de correspondencia entre los identificadores de esta rutina de tratamiento y la excepción se declararán en la parte de declaración de manejador del guión intercambiado.

### 10.5 Invocación de operaciones externas por un motor MHEG-3

Cuando un guión intercambiado expresa la invocación de una operación como se describe en 10.3, el intérprete de guión se comportará como se describe por la semántica de la instrucción **XCALL** en la cláusula 13. Como parte de este funcionamiento, interpretará los mecanismos descritos en 10.3 al traducirlos en los mecanismos del entorno de ejecución, definidos por las especificaciones de correspondencia de plataforma.

NOTA – Por ejemplo, un identificador de variable insertado en la pila como un parámetro de servicio puede ser traducido por un motor MHEG-3, o bien en un valor, o en una dirección de memoria real, que habrá de pasarse al soporte lógico externo que proporciona el servicio.

### 10.6 Tratamiento de excepciones externas por un motor MHEG-3

Cuando un servicio externo señala una excepción, esto tiene por consecuencia que se transmita un mensaje al motor MHEG-3 de acuerdo con el mecanismo de acceso al entorno de ejecución definido por las especificaciones de correspondencia de plataforma.

El intérprete de guión se comportará como se describe en 9.5.1.2.

### 10.7 Especificaciones de correspondencia de plataforma

Una especificación de correspondencia de plataforma contendrá lo siguiente:

– la descripción de la plataforma a que se aplica la especificación;

– el procedimiento de **package availability** (disponibilidad de lote) que los motores MHEG-3 aplicarán para verificar la disponibilidad de un lote dado en el entorno de ejecución;

– el procedimiento de **package load** (carga de lote) que los motores MHEG-3 aplicarán para hacer que las operaciones de un determinado lote sean accesibles por un guión rt;

– el procedimiento de **package unload** (descarga de lote) que los motores MHEG-3 aplicarán para descargar un lote;

– el procedimiento de **operation invocation** (invocación de operación) que los motores MHEG-3 aplicarán para invocar una determinada operación;

– las reglas de **data encoding** (codificación de datos) que los motores MHEG-3 deberán utilizar para codificar el valor de parámetros **in** o **inout** de una operación y de codificar el valor de parámetros **out** o **inout** de una operación o de miembros de una excepción;

– los procedimientos de **parameter passing** (paso de parámetros) que los motores MHEG-3 deberán utilizar para pasar parámetros **in**, **inout** y **out** a una operación;

– el procedimiento de **return value retrieval** (extracción de valor de retorno) que los motores MHEG-3 deberán utilizar para extraer el valor de retorno de una operación;

– el procedimiento de **exception retrieval** (extracción de excepción) que los motores MHEG-3 deberán utilizar para extraer excepciones señaladas por una operación.

El contenido de una especificación de correspondencia de plataforma se define en el anexo D.

## 11 Disposiciones para la manipulación de objetos MHEG

Esta cláusula describe los mecanismos definidos por la presente Recomendación para hacer posible que los guiones rt manipulen objetos MHEG.

### 11.1 Invocación de acciones MHEG

Se utiliza la representación MHEG-SIR para expresar la invocación de acciones MHEG, definidas por la interfaz MHEG API.

La interfaz MHEG API se define mediante el IDL (lenguaje de definición de interfaz). La correspondencia de una definición IDL con una declaración de lote MHEG-SIR y una declaración de tipo se define en la cláusula 14. Sin embargo, se considera que el lote MHEG API es un lote predefinido. Por tanto, su declaración no deberá incluirse explícitamente en guiones intercambiados. El mecanismo de correspondencia es similar al mecanismo de declaración de función externa descrito en la cláusula 10, con la salvedad de que los tipos y operaciones IDL definidas por la MHEG API no serán declaradas como parte del código MHEG-SIR, sino que se tratarán como tipos predefinidos y funciones externas predefinidas.

El mecanismo utilizado para invocar una acción MHEG es similar al utilizado para la invocación de un servicio proporcionado por el entorno de ejecución. Para esto se utiliza una instrucción **XCALL**. Los tipos definidos en el lote MHEG API son referenciados por un identificador de tipo predefinido. Las funciones descritas en el lote MHEG API son referenciadas mediante un identificador de función predefinida.

### 11.1.1 Envío de mensajes a otros guiones

Se considera que el lote MHEG-3 API es un lote predefinido. Dentro de un guión intercambiado, se puede dirigir eficientemente mensajes a otros guiones utilizando las funciones predefinidas que corresponden a operaciones MHEG-3 API. Por tanto, un guión rt puede pasar parámetros a otro guión rt, y recibir parámetros y llamar rutinas de otro guión rt.

NOTA – Esto puede utilizarse para aplicar el concepto de guiones de "library" (biblioteca) o "utility" (utilidad). Puede utilizarse también para sincronizar guiones rt.

### 11.1.2 Intercambio de información con objetos MHEG

El intercambio de información entre un guión rt y otras entidades MHEG (incluidos otros guiones rt) puede expresarse utilizando operaciones MHEG API que corresponden con las acciones MHEG "set data" (fijar datos) y "get data" (obtener datos). Se puede utilizar objetos de contenido que incorporan valores genéricos para constituir un área de memoria compartida entre objetos MHEG.

La espera de una señal proveniente de otro objeto puede traducirse por un bucle que incluye una llamada a la operación MHEG API que corresponde a la acción MHEG "get data" hasta que se extrae el valor esperado.

La generación de una señal puede traducirse por una llamada a la operación MHEG API que corresponde a la acción MHEG "set data".

NOTA – En lo que respecta al intercambio de información entre guiones rt, se recomienda la utilización del mecanismo descrito en 11.1.1.

## 11.2 Recepción de mensajes MHEG

Se utiliza la MHEG-SIR para expresar el tratamiento de mensajes que se producen como resultado de acciones MHEG. Estos mensajes pueden ser de:

– operaciones **run** de la MHEG-3 API;

– excepciones MHEG API.

### 11.2.1 Operaciones **run** de la MHEG-3 API

Las acciones "set parameters" y "run" de MHEG que pueden ser dirigidas a un guión rt deben dar por resultado las operaciones **setParameter** y **run** de la MHEG-3 API. La invocación de la operación **run** tiene por consecuencia la inserción de un mensaje en la cola de mensajes del guión rt con lo siguiente:

– como identificador de mensaje, un identificador de mensaje predefinido que corresponde al identificador de rutina de la rutina de destino;

– como miembros, los parámetros anteriormente fijados por la operación **setParameter**.

### 11.2.2 Excepciones MHEG API

Se considera que las excepciones MHEG API son mensajes que se envían al intérprete de guión como resultado de la invocación de una operación MHEG API. Estas excepciones tienen identificadores de mensajes predefinidos. El intérprete de guión procesará estos mensajes de la misma manera que procesaría una excepción proveniente del entorno de ejecución, como se describe en 9.5.1.2.

# 12 Declaraciones MHEG-SIR

Esta cláusula define la estructura de guiones intercambiados. También especifica la manera en que la máquina virtual trata el análisis (*parsing*) de un guión intercambiado.

Se utilizan los siguientes convenios de notación:

– los tipos no terminales se escriben como texto normal;
– los tipos terminales se escriben con mayúscula;
– los valores enumerados se encierran entre comillas simples;
– ":=" indica una definición;
– "|" indica una elección en una producción;
– "*" indica cero o más apariciones del tipo precedente;
– "+" indica una o más apariciones del tipo precedente;
– "?" indica cero o una aparición del tipo precedente (tipo facultativo).

NOTA – La gramática completa de los guiones intercambiados se describe en el apéndice I.

Un guión intercambiado consistirá en:

– una secuencia de declaraciones de tipos;
– una secuencia de declaraciones de constantes;
– una secuencia de declaraciones de variables globales;
– una secuencia de declaraciones de lotes;
– una secuencia de declaraciones de manejadores de mensajes;
– una secuencia de declaraciones de rutinas.

**InterchangedScript ::= TypeDeclaration***
**ConstantDeclaration***
**VariableDeclaration***
**PackageDeclaration***
**HandlerDeclaration***
**RoutineDeclaration***

## 12.1 Declaración de tipo

Se utilizan declaraciones de tipos para describir los tipos del guión intercambiado.

Una declaración de tipo consistirá en:

– un identificador de tipo (facultativo);
– una descripción de tipo.

**TypeDeclaration ::= TypeIdentifier?**
**TypeDescription**

### 12.1.1 Identificador de tipo

Se utilizan identificadores de tipos para hacer referencia a la descripción de tipo en la totalidad del guión intercambiado.

El identificador de tipo deberá ser un número entero positivo comprendido en la gama permitida para los tipos declarados. Corresponderá al número máximo de tipos predefinidos incrementado por el índice (comenzando por 0) de la declaración en la parte de declaraciones de tipos.

Si no se proporciona el identificador de tipo, deberá ser calculado por el analizador de guión.

**TypeIdentifier ::= INTEGER**

### 12.1.2 Descripción de tipo

Las descripciones de tipos describen la estructura de un tipo declarado.

La descripción de tipo será una de las descripciones siguientes:

– una descripción de string (cadena);
– una descripción de sequence (secuencia);
– una descripción de array (matriz);
– una descripción de structure (estructura);
– una descripción de union (unión).

```
TypeDescription     ::=  SequenceDescription
                    |    StringDescription
                    |    ArrayDescription
                    |    StructureDescription
                    |    UnionDescription
```

#### 12.1.2.1 Descripción de string

Una descripción de string (cadena) consistirá en un número entero (facultativo).

```
StringDescription   ::=  INTEGER?          // String (max) size
```

El número entero representa el tamaño máximo de la cadena; si no se proporciona, la cadena será una cadena no acotada.

#### 12.1.2.2 Descripción de sequence

Una descripción de sequence (secuencia) consistirá en:

– un número entero (facultativo);
– un identificador de tipo.

```
SequenceDescription     ::=  INTEGER?          // Sequence (max) size
                             TypeIdentifier
```

El número entero representa el tamaño máximo de la secuencia; si no se proporciona, la secuencia será una secuencia no acotada.

El identificador de tipo representa el tipo de elemento de la secuencia.

#### 12.1.2.3 Descripción de array

Una descripción de array (matriz) consistirá en:

– un número entero;
– un identificador de tipo.

```
ArrayDescription    ::=  INTEGER           // Array size
                         TypeIdentifier
```

El número entero representa el tamaño de la matriz.

El identificador de tipo representa el tipo de elemento de la matriz.

#### 12.1.2.4 Descripción de structure

Una descripción de structure (estructura) consistirá en una secuencia de identificadores de tipos.

```
StructureDescription    ::=  TypeIdentifier+
```

Cada identificador de tipo representa el tipo de uno de los campos de la estructura.

#### 12.1.2.5 Descripción de union

Una descripción de union (unión) consiste en una secuencia de uno o más identificadores de tipos.

**UnionDescription ::= TypeIdentifier+**

Cada identificador de tipo representa el tipo de una de las posibilidades de elección de la unión.

## 12.2 Declaración de constante

Las declaraciones de constantes se utilizan para describir los tipos y valores de las constantes del guión intercambiado.

Una declaración de constante consistirá en:

– un identificador de datos (facultativo);

– un identificador de tipo;

– un valor de constante.

**ConstantDeclaration ::= DataIdentifier?**
**TypeIdentifier**
**ConstantValue**

### 12.2.1 Identificador de datos

Se utilizan identificadores de datos para hacer referencia a datos en la totalidad del guión intercambiado.

El identificador de datos será un número entero positivo comprendido en la gama permitida para las constantes. Corresponderá al índice (comenzando por 0) de la declaración en la parte de declaraciones de constantes.

Si no se proporciona el identificador de tipo, será calculado por el analizador de guión.

**DataIdentifier ::= INTEGER**

### 12.2.2 Identificador de tipo

El identificador de tipo representa el tipo a que pertenece el valor de la constante.

### 12.2.3 Valor (de) constante

El valor (de) constante representa el valor a que corresponde la constante en la totalidad del guión.

Si el tipo de la constante es un tipo primitivo o **string** (cadena), el valor de constante consistirá en un valor inmediato expresado en este tipo.

Si el tipo de la constante es un tipo **sequence** (secuencia), el valor de constante consistirá en una secuencia de valores de constante cuya longitud es inferior o igual al tamaño del tipo **sequence** y cuyo tipo es el tipo de elemento de la descripción de la secuencia.

Si el tipo de la constante es un tipo **array** (matriz), el valor de constante consistirá en una secuencia de valores de constante, cuya longitud es igual al tamaño del tipo **array** y cuyo tipo es el tipo de elemento de la descripción de la matriz.

Si el tipo de la constante es un tipo **structure** (estructura), el valor de constante consistirá en una secuencia de valores de constante, cuya longitud es igual al número de elementos en el tipo **structure**; cada uno de estos valores será del mismo tipo que el tipo de elemento correspondiente en la descripción de estructura.

Si el tipo de la constante es un tipo **union** (unión), el valor de constante consistirá en un número entero que representa el índice (comenzando por 0) de la posibilidad de elección en el tipo **union** y un valor de constante cuyo tipo es el tipo de elemento del rango correspondiente en la descripción de unión.

```
ConstantValue       ::=  BOOLEAN
                    |    OCTET
                    |    INTEGER    // all numeric types
                    |    REAL       // float or double
                    |    STRING     // character or string
                    |    DataIdentifier
                    |    ConstantValue*    // sequence, array or structure
                    |    UnionValue

UnionValue          ::=  INTEGER    // Tag index
                         ConstantValue
```

## 12.3 Declaración de variables globales

Se utilizan declaraciones de variables globales para describir los tipos y los valores iniciales de las variables globales del guión intercambiado.

Una declaración de variable global consistirá en:

– un identificador de datos (facultativo);

– un identificador de tipo;

– una referencia de constante (facultativa).

```
VariableDeclaration    ::=  DataIdentifier?
                            TypeIdentifier
                            ConstantReference? // Initial value
```

### 12.3.1 Identificador de datos

Se utilizan identificadores de datos para hacer referencia a datos en la totalidad del guión intercambiado.

El identificador de datos será un número entero positivo comprendido en la gama permitida para variables globales. Corresponderá al número máximo de constantes incrementado por el índice (comenzando por 0) de la declaración en la parte de declaraciones de variables globales.

Si no se proporciona el identificador de datos, será calculado por el analizador de guión.

### 12.3.2 Identificador de tipo

El identificador de tipo representa el tipo a que pertenece el valor de la variable global.

### 12.3.3 Referencia de constante

La referencia de constante representa el valor inicial de la variable global.

La referencia de constante será:

– o bien un identificador de datos que hace referencia a una constante;

– o un valor (de) constante como los descrito en 12.2.3.

En cualquiera de estos casos, el valor a que hace referencia la referencia de constante será del tipo de la variable global.

Si no se proporciona la referencia de constante, el intérprete de guión asignará a la variable global un valor por defecto si su tipo lo permite. De lo contrario, seguirá sin estar definida hasta que sea asignada por una instrucción.

```
ConstantReference    ::=  DataIdentifier
                     |    ConstantValue
```

## 12.4 Declaración de lote

Se utilizan declaraciones de lotes para describir los servicios y excepciones externos utilizados por el guión intercambiado.

Una declaración de lote consistirá en:

– un identificador de lote (facultativo);

– una cadena que representa el nombre del lote;

– una secuencia de descripciones de servicios;

– una secuencia de descripciones de excepciones.

```
PackageDeclaration    ::=  PackageIdentifier?
                           VisibleString        // Package name
                           ServiceDescription*
                           ExceptionDescription*
```

### 12.4.1 Identificador de lote

Se utilizan identificadores de lotes para hacer referencia a lotes en la totalidad del guión intercambiado.

El identificador de lote será un número entero positivo comprendido en la gama permitida para los lotes. Corresponderá al índice (comenzando por 0) de la declaración en la parte de declaraciones de lotes.

Si no se proporciona el identificador de lote, será calculado por el analizador de guión.

```
PackageIdentifier    ::=  INTEGER
```

### 12.4.2 Nombre

El intérprete de guión utiliza un nombre de lote para ganar acceso al lote en el entorno de ejecución, de acuerdo con el procedimiento **package availability** (disponibilidad de lote) descrito por la especificación de correspondencia de plataforma.

### 12.4.3 Descripción de servicio

Las descripciones de servicio describen prototipos de funciones externas.

Una descripción de servicio consistirá en:

– un identificador de función (facultativo);

– una cadena que representa el nombre de la operación;

– un modo de llamada (facultativo);

– un identificador de tipo (facultativo);

– una secuencia de descripciones de parámetros.

```
ServiceDescription    ::=  FunctionIdentifier?
                           VisibleString?       // IDL global name
                           CallingMode?
```

**TypeIdentifier? // return value**
**ServiceParameterDescription***

#### 12.4.3.1 Identificador de función

Se utilizan identificadores de funciones para hacer referencia a funciones en la totalidad del guión intercambiado.

El identificador de función será un número entero positivo comprendido en la gama permitida para servicios. Corresponderá al número máximo de rutinas más el número máximo de funciones predefinidas más el identificador de lote multiplicado por 256, incrementado por el índice (comenzando por 0) del servicio en la declaración de lote.

Si no se proporciona el identificador de función, deberá ser calculado por el analizador de guión.

**FunctionIdentifier ::= INTEGER**

#### 12.4.3.2 Nombre

El intérprete de guión utiliza el nombre de operación para ganar acceso a la operación en el entorno de ejecución, de acuerdo con el procedimiento **operation invocation** (invocación de operación) descrito por la especificación de correspondencia de plataforma.

#### 12.4.3.3 Modo de llamada

El modo de llamada representa la forma en que se invoca la operación.

El modo de llamada será "synchronous" (síncrono) o "asynchronous" (asíncrono).

Si no se especifica el valor, el modo de llamada será "synchronous".

**CallingMode ::= 'SYNCHRONOUS' | 'ASYNCHRONOUS'**

#### 12.4.3.4 Identificador de tipo

El identificador de tipo representa el tipo del valor de retorno del servicio.

Si no se especifica el identificador de tipo, deberá interpretarse como un tipo **void**, es decir, la función no tendrá valor de retorno.

Si el modo de llamada de la operación es "asynchronous", el identificador de tipo será "void" o no estará especificado.

#### 12.4.3.5 Descripción de parámetro

Se utilizan descripciones de parámetros para especificar el tipo y el modo de paso de parámetros de servicio.

Una descripción de parámetro consistirá en:

– un modo de paso;

– un identificador de tipo.

**ServiceParameterDescription ::= ServicePassingMode?**
**TypeIdentifier**

##### 12.4.3.5.1 Modo de paso

El modo de paso indica si el valor del parámetro en el momento de la invocación del servicio es utilizado por el servicio (parámetro de entrada) y si este parámetro es modificado por el servicio para uso por el llamante (parámetro de salida).

El modo de paso será uno de los siguientes: "in", "inout" o "out".

Si no se especifica el modo de paso de un parámetro, se interpretará que es un parámetro **in**.

NOTA – El parámetro referencia de objeto es implícito, por lo que no se especifica como parte de la declaración. Se trata como un parámetro **in**.

Si el modo de llamada de la operación es "asynchronous", el modo de paso será "in" o no estará especificado.

**ServicePassingMode ::= 'IN' | 'OUT' | 'INOUT'**

#### 12.4.3.5.2 Identificador de tipo

El identificador de tipo representa el tipo del parámetro de servicio considerado.

### 12.4.4 Descripción de excepción

Las descripciones de excepciones describen prototipos de excepciones que pueden ser señaladas durante la ejecución de funciones externas.

Una descripción de excepción consistirá en:

– un identificador de mensaje (facultativo);

– una cadena que representa el nombre de la excepción;

– una secuencia de identificadores de tipo que representan los miembros de la excepción.

```
ExceptionDescription    ::=    MessageIdentifier?
                               VisibleString?      //IDL exception global name
                               TypeIdentifier*     //Parameter types
```

#### 12.4.4.1 Identificador de mensaje

Se utilizan identificadores de mensajes para hacer referencia a mensajes en la totalidad del guión intercambiado.

El identificador de mensaje será un número entero positivo comprendido en la gama permitida para las excepciones. Corresponderá al número máximo de mensajes predefinidos más el identificador de paquete multiplicado por 256, incrementado por el índice (comenzando por 0) de la excepción en la declaración del lote.

Si no se proporciona el identificador de mensaje, deberá ser calculado por el analizador de guión.

**MessageIdentifier ::= INTEGER**

#### 12.4.4.2 Nombre

El intérprete de guión utiliza un nombre de excepción para extraer la excepción del entorno de ejecución de acuerdo con el procedimiento **exception retrieval** (extracción de excepción) descrito en la especificación de correspondencia de plataforma.

#### 12.4.4.3 Descripción de parámetro

Cada parámetro del mensaje corresponde a un miembro de la excepción. Se describe por su identificador de tipo.

## 12.5 Declaración de manejador (handler declaration)

Se utilizan declaraciones de manejadores para asociar un mensaje con la función que lo maneja.

Una declaración de manejador consta de:

– un identificador de mensaje;

– un identificador de función.

```
HandlerDeclaration      ::=     MessageIdentifier
                                FunctionIdentifier
```

### 12.5.1 Identificador de mensaje

El identificador de mensaje indica el mensaje que habrá de tratarse.

El identificador de mensaje será un número entero positivo comprendido en la gama completa autorizada para mensajes, que representa un mensaje predefinido o una excepción.

### 12.5.2 Identificador de función

El identificador de función indica la función que habrá de iniciarse cuando se retira el mensaje de la cola de mensajes.

El identificador de función será un número entero positivo comprendido en la gama completa permitida para las funciones, que representa una rutina, una función predefinida o un servicio.

La descripción de los tipos de parámetros formales para la función será la misma que para el mensaje, por lo que la función puede invocarse con los parámetros reales del mensaje como sus parámetros. Si las firmas no concuerdan, el analizador de guión rechazará el manejador.

## 12.6 Declaración de rutina

Se utilizan declaraciones de rutinas para describir la estructura y el código de programa de las funciones internas del guión intercambiado.

Una declaración de rutina consistirá en:

– un identificador de función (facultativo);

– un identificador de tipo (facultativo);

– una secuencia de descripciones de parámetros;

– una secuencia de declaraciones de variables locales;

– un código de programa MHEG-SIR.

```
RoutineDeclaration      ::=     FunctionIdentifier?
                                TypeIdentifier?             // for return value
                                RoutineParameterDescription*
                                LocalVariableDeclaration*
                                OCTET STRING                // program code
```

### 12.6.1 Identificador de función

El identificador de función será un número entero positivo comprendido en la gama permitida para las rutinas. Corresponderá al índice (comenzando por 0) de la rutina en la parte de declaraciones de rutinas.

Si no se proporciona el identificador de función, será calculado por el analizador de guión.

### 12.6.2 Identificador de tipo

El identificador de tipo representa el tipo de valor de retorno de la rutina.

Si no se especifica el identificador de tipo deberá interpretarse como un tipo **void**, es decir, la función no tendrá valor de retorno.

### 12.6.3 Descripción de parámetro

Se utilizan descripciones de parámetros para especificar el tipo y el modo de paso de los parámetros de las rutinas.

Una descripción de parámetros consistirá en:

– un modo de paso (facultativo);

– un identificador de tipo.

**RoutineParameterDescription ::= RoutinePassingMode?**
**TypeIdentifier**

#### 12.6.3.1 Modo de paso

El modo de paso indica si el parámetro deberá pasarse a la rutina utilizando su **value** (valor) (parámetro de entrada) o una **reference** (referencia) a la variable que contiene su valor (parámetro de entrada/salida).

El modo de paso será uno de los siguientes: "value" o "reference".

**RoutinePassingMode ::= 'VALUE' | 'REFERENCE'**

#### 12.6.3.2 Identificador de tipo

El identificador de tipo representa el tipo de parámetro de la rutina considerada.

### 12.6.4 Declaración de variable local

Se utilizan declaraciones de variables locales para describir los tipos y valores iniciales de variables cuyo alcance está limitado a una ejecución de una rutina.

Una declaración de variable local tendrá la misma estructura que una declaración de variable global, como se define en 12.3. Consistirá en:

– un identificador de datos (facultativo);

– un identificador de tipo;

– una referencia de constante (facultativa).

#### 12.6.4.1 Identificador de datos

El identificador de datos será un número entero positivo comprendido en la gama permitida para las variables locales. Corresponderá al número máximo de constantes más el número máximo de variables globales incrementado por el índice (comenzando por 0) de la declaración en las declaraciones de variables locales de la rutina, incrementado por el número de parámetros formales de la rutina.

Si no se proporciona el identificador de datos, deberá ser calculado por el analizador de guión.

#### 12.6.4.2 Identificador de tipo

El identificador de tipo representa el tipo a que pertenece el valor de la variable local.

#### 12.6.4.3 Referencia de constante

La referencia de constante representa el valor inicial de la variable local.

La referencia de constante será:

– o bien un identificador de datos que hace referencia a una constante;

– o un valor constante como se define en 12.2.3.

Tanto en uno como en otro caso, el valor a que hace referencia esta referencia de constante será del tipo de la variable local.

Si no se proporciona la referencia de constante, el intérprete de guión asignará a la variable local un valor por defecto si su tipo lo permite. De lo contrario, permanecerá sin ser definido hasta que sea asignado por una instrucción.

### 12.6.5 Código de programa

El código de programa consiste en una secuencia de instrucciones de la rutina, prevista para ser ejecutada por el intérprete de guión cuando se inicie la rutina. La sintaxis y la semántica de las instrucciones MHEG-SIR se describen en la cláusula 13.

La última instrucción de una rutina deberá ser una instrucción **RET**.

## 13 Instrucciones MHEG-SIR

Esta cláusula define la semántica de las instrucciones MHEG-SIR.

### 13.1 Metodología de presentación

Cada instrucción se define en la sección correspondiente por el siguiente conjunto de entradas:

| | |
|---|---|
| Descripción breve: | Una breve descripción de la semántica de la instrucción. |
| Sinopsis: | **Nemónico Operando1 ... OperandoN** |
| Operandos: | Una descripción de los tipos y de la semántica de cada operando asociado a la instrucción (si los hubiere). |
| Pila: | Una representación del efecto que produce la instrucción sobre la pila de parámetros, por ejemplo<br>**..., Parámetro1, Parámetro2 ⇨ ..., Resultado** |
| Tipos: | Una lista de los tipos de parámetros a que se aplica la instrucción (si es una instrucción por plantilla). |
| Parámetros: | Una descripción de la semántica de cada elemento de la pila de parámetros que es sacado de la pila (*popped*), insertado en la pila (*pushed*) o afectado de cualquier otra forma por la instrucción (si existe). |
| Efecto: | Una especificación textual de la manera en que ha de interpretarse la semántica de la instrucción. |
| Especificación formal: | Una especificación formal de la manera en que ha de interpretarse la semántica de la instrucción, expresada mediante la notación descrita en esta subcláusula. |
| Errores: | Una lista de los errores que pueden ser señalados durante la ejecución de una instrucción. |

### 13.1.1 Condiciones de error

La semántica de la instrucción, tal como se describe en la especificación formal, se aplicará solamente si los operandos son válidos. De lo contrario se señalará una excepción **InstructionExecutionError** y el registro de error se fijará a **InvalidOperand**. El resultado de la ejecución de la instrucción no está especificado.

Cuando se examina la pila de parámetros, es decir, en el caso de una primitiva **PS.pop** o **PS[SP]**, si la pila de parámetros no contiene un número suficiente de parámetros, se señalará una excepción **InstructionExecutionError** y el registro de error se fijará a **StackUnderflow**. El estado resultante de la pila de parámetros no está especificado.

Si el resultado de una operación aritmética cae en una gama que excede la del tipo de destino, las operaciones aritméticas señalarán una excepción **InstructionExecutionError** y el registro de error se fijará a **ArithmeticOverflow** o **DivisionByZero**, según el caso.

Si un identificador no hace referencia a una entidad válida (tipo, dato, función, mensaje o lote), entonces, cuando se gana acceso (por ejemplo, utilizando **DT[i]**) a su valor en la tabla correspondiente, se señalará una excepción **InstructionExecutionError** y el registro de error se fijará a **InvalidIdentifier**.

Si el registro IP está fijado a un puntero no válido, se señalará la excepción **InstructionExecutionError** y el registro de error se fijará a **JumpOutOfRange**.

Cuando se intenta atribuir (*allocate*) una variable dinámica y la atribución (*allocation*) es imposible por falta de espacio de memoria o de identificadores de datos, la primitiva **new()** señalará una excepción **InstructionExecutionError** y fijará el registro de error a **AllocationFailed**.

La señalización de otras condiciones de error se especifica explícitamente en 13.3. Los valores de los códigos de error se definen en el anexo C.

### 13.1.2 Especificación formal

La entrada "especificación formal" de una descripción de instrucción da una notación formal concisa del efecto que la unidad de ejecución de instrucción deberá producir cuando interpreta la instrucción; sin embargo, como esta especificación se expresa como una secuencia de operaciones, puede haber otros métodos que den el mismo resultado, por lo que esta especificación formal no requiere que la unidad de ejecución de instrucción funcione como se ha expresado, siempre que el efecto sea el mismo.

Los casos de error descritos en 13.1.1 son implícitos y no se indican explícitamente en la especificación formal. Los otros casos de error se indican explícitamente.

Para especificar la semántica de una instrucción de una manera formal se utiliza una sintaxis similar a la del lenguaje de programación C. Esta sintaxis utiliza las notaciones y conceptos definidos en las cláusulas 8 y 9, y además las notaciones siguientes:

– notación de tabla de datos (DT, *data table*) (véase 13.1.3);
– notación de instrucción por plantilla (*template instruction*) (véase 3.1.4);
– primitivas (véase 13.1.5).

### 13.1.3 Notación de tabla de datos

La notación **DT(i)**, donde **i** es un identificador de datos, corresponde a:

– la entrada cuya clase es **i** en la tabla de constantes, si **i** es el identificador de datos o una constante;

– la entrada cuya clave es **i** en la tabla de variables globales, si **i** es el identificador de datos o una variable global;
– la entrada cuya clave es **i** en la tabla de variables locales de la rutina que se está ejecutando en ese momento, si **i** es el identificador de datos o una variable local;
– la variable dinámica cuya asa (*handle*) corresponde a **i**, si **i** es el identificador de datos o una variable dinámica.

Esta macro se puede expresar como sigue:

```
#define DT(i) (i < 4096) ? CT[i] : VT[i])
```

### 13.1.4 Notación de instrucción por plantilla

Cierto número de instrucciones (por ejemplo instrucciones aritméticas o lógicas) actúan sobre valores de un tipo dado y producen un resultado del mismo tipo. Se utiliza la notación **<T>** para denotar una instrucción por plantilla. **<Nemónico>_<T>** representa todas las instrucciones **<Nemónico>**, sustituyéndose **<T>** por la letra que representa el tipo de cualquier primitiva a que es aplicable la instrucción, descrito por la entrada "Tipos" en la descripción de instrucción.

NOTA – Las operaciones sobre tipos mixtos deben tratarse insertando explícitamente instrucciones de conversión de tipo en la secuencia de instrucciones.

### 13.1.5 Primitivas

En la especificación formal de instrucciones se utilizan las siguientes notaciones primitivas:

– **DID new(tid)**: atribuye una variable dinámica del tipo identificado por **tid**;
– **void raise(exc)**: señala una excepción **InstructionExecutionError** y fija el registro de error (ER) al código de error **exc**;
– **void delete(did)**: libera la variable dinámica identificada por **did**;
– **int sizeof(tid)**: retorna el tamaño de valores identificados por **tid**, expresado en las mismas unidades que aquéllas a las que apunta el puntero PS;
– **type(<T>)**: macro que se sustituirá por el nombre de tipo en el lenguaje C.

## 13.2 Clasificación de las instrucciones MHEG-SIR

Las instrucciones MHEG-SIR pueden ser agrupadas en categorías atendiendo al efecto que producen en el flujo de control, en las tablas de variables o en la pila de parámetros, y de acuerdo con los tipos de los parámetros tomados de la pila que ellas aceptan:

a) Instrucciones que afectan al flujo de control:
   1) instrucciones de salto incondicional: **JMP, LJMP**;
   2) instrucciones de salto condicional: **JT, JF, LJT, LJF**;
   3) invocaciones de funciones: **CALL, XCALL**;
   4) instrucciones diversas de flujo de control: **RET, YIELD**.

b) Instrucciones que no afectan al flujo de control:
   1) modificadores de variables complejas: **SET, SETC**;
   2) operadores aritméticos que actúan sobre variables: **INC, DEC**;
   3) instrucciones para sacar elementos de la pila: **POPR, POP, POPC**;
   4) instrucciones de gestión de la memoria: **ALLOC, FREE**.

c) Instrucciones que no afectan al flujo de control ni a las variables, pero sí a la pila de parámetros:

1) instrucciones de conversión: **CVT**;
2) operadores aritméticos: **ADD, SUB, MUL, DIV, REM, NEG**;
3) operadores lógicos: **AND, OR, XOR, NOT**;
4) operadores de desplazamiento lógicos: **SHIFT**;
5) operadores de comparación: **EQ, GT, LT, EQR**;
6) accesores a datos complejos: **GET, GETC**;
7) instrucciones diversas de manipulación de pila: **PUSHI, PUSHR, PUSH, DUP, GETOR**.

d) Instrucciones que no producen ningún efecto: **NOP**.

NOTA – La mayor parte de las instrucciones actúan sobre valores de tipos primitivos. Sólo las siguientes instrucciones se utilizan para manipular valores construidos: **EQR, GET, GETC, SET, SETC, ALLOC, FREE, CALL, XCALL**.

Los efectos de las instrucciones se recapitulan en el cuadro 3. Las operaciones se indican en orden canónico, es decir, por número de código op ascendente. Algunos nemónicos representan instrucciones por plantilla, por lo que tienen un sufijo que representa su tipo.

**Cuadro 3/T.173 – Sinopsis de las instrucciones MHEG-SIR y sus efectos**

| Nemónicos | Ref. | Código op (hexadec.) | Tamaño del código op. | Tipo del código op. | Tipos de parámetros | Efecto en la pila PS | Efecto en la tabla VT | Efecto en el flujo de control |
|---|---|---|---|---|---|---|---|---|
| **NOP** | 13.3.1 | 00 | 0 | | | | | |
| **YIELD** | 13.3.2 | 02 | 0 | | | | | x |
| **RET** | 13.3.3 | 03 | 0 | | | 0\|1 ⇨ 0\|1 | | x |
| **FREE** | 13.3.4 | 08 | 0 | | | 1 ⇨ 0 | x | |
| **NOT_<T>** | 13.3.5 | 10-13 | 0 | | BOWU | 1 ⇨ 1 | | |
| **OR_<T>** | 13.3.6 | 14-17 | 0 | | BOWU | 2 ⇨ 1 | | |
| **XOR_<T>** | 13.3.7 | 18-1B | 0 | | BOWU | 2 ⇨ 1 | | |
| **AND_<T>** | 13.3.8 | 1C-1F | 0 | | BOWU | 2 ⇨ 1 | | |
| **EQR** | 13.3.9 | 20 | 0 | | | 2 ⇨ 1 | | |
| **EQ_<T>** | 13.3.10 | 21-2B | 0 | | OSLWUFDBCIR | 2 ⇨ 1 | | |
| **LT_<T>** | 13.3.11 | 30-37 | 0 | | COSLWUFD | 2 ⇨ 1 | | |
| **GT_<T>** | 13.3.12 | 38-3F | 0 | | COSLWUFD | 2 ⇨ 1 | | |
| **ADD_<T>** | 13.3.13 | 40-47 | 0 | | OSLWUFD | 2 ⇨ 1 | | |
| **SUB_<T>** | 13.3.14 | 48-4F | 0 | | OSLWUFD | 2 ⇨ 1 | | |
| **MUL_<T>** | 13.3.15 | 50-57 | 0 | | OSLWUFD | 2 ⇨ 1 | | |
| **DIV_<T>** | 13.3.16 | 58-5F | 0 | | OSLWUFD | 2 ⇨ 1 | | |
| **NEG_<T>** | 13.3.17 | 62-67 | 0 | | SLFD | 1 ⇨ 1 | | |
| **REM_<T>** | 13.3.18 | 79-7D | 0 | | OSLWU | 2 ⇨ 1 | | |
| **DUP_<T>** | 13.3.19 | 81-8B | 0 | | OSLWUFDBCIR | 1 ⇨ 2 | | |
| **CVT_<TT>** | 13.3.20 | 94-BE | 0 | | OSLWUFDBC | 1 ⇨ 1 | | |
| **JT** | 13.3.21 | C0 | 1 | offset | | 1 ⇨ 0 | | x |
| **JF** | 13.3.22 | C1 | 1 | offset | | 1 ⇨ 0 | | x |
| **JMP** | 13.3.23 | C2 | 1 | offset | | | | x |
| **SHIFT_<T>** | 13.3.24 | C5-C7 | 1 | offset | OWU | 1 ⇨ 1 | | |
| **GETOR** | 13.3.25 | C9 | 1 | PID | | 0 ⇨ 1 | | |
| **LJT** | 13.3.26 | D0 | 2 | offset | | 1 ⇨ 0 | | x |

**Cuadro 3/T.173 – Sinopsis de las instrucciones MHEG-SIR y sus efectos** *(fin)*

| Nemónicos | Ref. | Código op (hexadec.) | Tamaño del código op. | Tipo del código op. | Tipos de parámetros | Efecto en la pila PS | Efecto en la tabla VT | Efecto en el flujo de control |
|---|---|---|---|---|---|---|---|---|
| **LJF** | 13.3.27 | D1 | 2 | offset | | 1 ⇨ 0 | | x |
| **LJMP** | 13.3.28 | D2 | 2 | offset | | | | x |
| **CALL** | 13.3.29 | D4 | 2 | FID | | n ⇨ 0\|1 | | x |
| **XCALL** | 13.3.30 | D6 | 2 | FID | | n ⇨ 0\|1 | | x |
| **PUSH** | 13.3.31 | E0 | 2 | DID | | 0 ⇨ 1 | | |
| **PUSHR** | 13.3.32 | E1 | 2 | DID | | 0 ⇨ 1 | | |
| **PUSHI** | 13.3.33 | E3 | 2 | value | | 0 ⇨ 1 | | |
| **POP** | 13.3.34 | E4 | 2 | DID | | 1 ⇨ 0 | x | |
| **POPR** | 13.3.35 | E5 | 2 | DID | | 1 ⇨ 0 | x | |
| **POPC** | 13.3.36 | E6 | 2 | DID | | 1 ⇨ 0 | x | |
| **ALLOC** | 13.3.37 | E8 | 2 | TID | | 0 ⇨ 1 | x | |
| **INC** | 13.3.38 | EA | 2 | DID | | 1 ⇨ 0 | x | |
| **DEC** | 13.3.39 | EB | 2 | DID | | 1 ⇨ 0 | x | |
| **GET** | 13.3.40 | F0 | 3 | DID, idx | | idx ⇨ 1 | | |
| **GETC** | 13.3.41 | F2 | 3 | DID, idx | | idx+1 ⇨ 0 | x | |
| **SET** | 13.3.42 | F4 | 3 | DID, idx | | idx+1 ⇨ 0 | x | |
| **SETC** | 13.3.43 | F6 | 3 | DID, idx | | idx+1 ⇨ 0 | x | |

## 13.3 Descripción de las instrucciones

### 13.3.1 Ninguna operación (no operation)

Descripción breve: No hace nada.

Sinopsis: **NOP**

Operandos: Ninguno.

Tipos: No aplicable.

Parámetros: Ninguno.

Pila: **... ⇨ ...**

Efecto: Ninguno.

Especificación formal: **0;**

Errores:

### 13.3.2 Producir (yield)

Descripción breve: Trata mensajes pendientes.

Sinopsis: **YIELD**

Operandos: Ninguno.

Pila: **... ⇨ ...**

Tipos: No aplicable.

Parámetros: Ninguno.

Efecto: Si hay un mensaje pendiente en la cola de mensajes, lo trata mediante una llamada a la rutina correspondiente. Una vez retornada la rutina, repite el proceso, y así sucesivamente hasta que la cola de mensajes esté vacía.

Especificación formal:

```
while (QP != 'null')
{
        FID fid = HT[MQ[QP].MID].FID;
        if (fid == 'null') then   raise('HandlerNotFound');
        else
        {
            CS.push({IP-1, FR, SP, LT});
            // IP-1: allows to re-iterate the YIELD          instruction
            FR = fid;
            IP = RT[FR].IP;
            LT = MQ[QP].LT;
        }
        MQ.remove();
}
```

Errores: **HandlerNotFound**

### 13.3.3 Retornar (return)

Descripción breve: Retorna al llamante.

Sinopsis: **RET**

Operandos: Ninguno.

Pila: **..., (Val) ⇨ ..., (Val)**

Tipos: No aplicable.

Parámetros: Si la firma de la rutina actual tiene un valor de retorno, **Val** se interpretará como del tipo de este valor de retorno.

Si no lo tiene, no se considerará ningún parámetro de la pila.

Efecto: Retorna a la rutina llamante. Saca un elemento de la pila de llamada y restablece el contexto de la trama precedente. Si la rutina actual tiene un valor de retorno, comprueba que hay un valor del mismo tipo en la cima de la pila de parámetros. Si no hay una rutina llamante a la cual retornar, detiene el proceso y regresa al estado **ready**.

Especificación formal:

```
if (sizeof(RT[FR].TID) != (SP - CS[FP].SP))
        then raise('InvalidReturnValue');
IP = CS[FP].IP;
FR = CS[FP].FR;
LT = CS[FP].LT;
CS.pop();
```

Errores: **InvalidReturnValue**

### 13.3.4 Liberar (free)

| | |
|---|---|
| Descripción breve: | Libera variable dinámica. |
| Sinopsis: | **FREE** |
| Operandos: | Ninguno. |
| Pila: | **..., Did ⇨ ...** |
| Tipos: | No aplicable. |
| Parámetros: | **Did** se interpretará como un identificador de datos. |
| Efecto: | Comprueba que **Did** es el identificador de datos de una variable dinámica. Libera la memoria dinámica asociada con **Did**, e invalida el identificador de datos. |
| Especificación formal: | **if (did < 8100h) then raise('InvalidParameter');**<br>**delete(VT[PS.pop('data identifier')]);** |
| Errores: | **StackUnderflow**<br>**InvalidIdentifier** |

### 13.3.5 No (not)

| | |
|---|---|
| Descripción breve: | Negación lógica. |
| Sinopsis: | **NOT_<T>** |
| Operandos: | Ninguno. |
| Pila: | **..., Val ⇨ ..., Neg** |
| Tipos: | **Booleano** o cualquier tipo entero sin signo (B, O, W U). |
| Parámetros: | **Val** se interpretará como de tipo <**T**>.<br>**Neg** será de tipo <**T**>. |
| Efecto: | Sustituye el elemento en la cima de la pila de parámetros por su negación lógica si <**T**> es **B** y, si es de otro tipo, por su negación bit por bit (es decir, por su complemento de uno):<br>**Neg = ~Val** |
| Especificación formal: | **type(<T>) buf = PS.pop(<T>);**<br>**if (<T> == 'boolean') then PS.push(! buf);**<br>**else PS.push(~ buf);** |
| Errores: | **StackUnderflow** |

### 13.3.6 O (or)

| | |
|---|---|
| Descripción breve: | Disyunción lógica. |
| Sinopsis: | **OR_<T>** |
| Operandos: | Ninguno. |
| Pila: | **..., Val1, Val2 ⇨ ..., Disj** |
| Tipos: | **Booleano** o cualquier tipo entero sin signo (B, O, W U). |
| Parámetros: | **Val1** y **Val2** se interpretarán como de tipo <**T**>.<br>**Disj** será de tipo <**T**>. |

Efecto: Sustituye los dos elementos que ocupan las dos posiciones más altas de la pila de parámetros por su disyunción lógica si **<T>** es **B** y, si es de otro tipo, por su disyunción bit por bit:

**Disj = Val1 | Val2**

Especificación formal:
```
type(<T>) buf = PS.pop(<T>);
if (<T> == 'boolean') then buf = buf || PS.pop('boolean');
else buf |= PS.pop(<T>);
PS.push(buf);
```

Errores: **StackUnderflow**

### 13.3.7 O exclusivo (exclusive or)

Descripción breve: Exclusión lógica.

Sinopsis: **XOR_<T>**

Operandos: Ninguno.

Pila: **..., Val1, Val2 ⇨ ..., Excl**

Tipos: **Booleano** o cualquier tipo entero sin signo (B, O, W U).

Parámetros: **Val1** y **Val2** se interpretarán como de tipo **<T>**.

**Excl** será de tipo **<T>**.

Efecto: Sustituye los dos elementos que ocupan las dos posiciones más altas de la pila de parámetros por su exclusión lógica si **<T>** es **B** y, si es otro tipo, por su exclusión bit por bit:

**Excl = Val1 ^ Val2**

Especificación formal:
```
type(<T>) buf = PS.pop(<T>);
if (<T> == 'boolean') then buf = (buf != PS.pop('boolean'));
else buf ^= PS.pop(<T>);
PS.push(buf);
```

Errores: **StackUnderflow**

### 13.3.8 Y (and)

Descripción breve: Conjunción lógica.

Sinopsis: **AND_<T>**

Operandos: Ninguno.

Pila: **..., Val1, Val2 ⇨ ..., Conj**

Tipos: **Booleano** o cualquier tipo entero sin signo (B, O, W U).

Parámetros: **Val1** y **Val2** se interpretarán como de tipo **<T>**.

**Conj** será de tipo **<T>**.

Efecto: Sustituye los dos elementos que ocupan las dos posiciones más altas de la pila de parámetros por su conjunción lógica si **<T>** es **B** y, si es otro tipo, por su conjunción bit por bit:

**Conj = Val1 & Val2**

Especificación formal:
```
type(<T>) buf = PS.pop(<T>);
if (<T> == 'boolean') then buf = buf && PS.pop('boolean');
else buf &= PS.pop(<T>);
PS.push(buf);
```

Errores: **StackUnderflow**

### 13.3.9 Referencia de igual (equal reference)

Descripción breve: Compara valores construidos.

Sinopsis: **EQR**

Operandos: Ninguno.

Pila: **..., Did1, Did2 ⇨ ..., Bool**

Tipos: No aplicable.

Parámetros: **Did1** y **Did2** se interpretarán como de tipo **data identifier**.

**Bool** será de tipo **boolean**.

Efecto: Comprueba que **Did1** y **Did2** identifican datos del mismo tipo. Retorna "true" si los datos identificados por **Did1** y **Did2** son iguales (véase 8.2) "false" en caso contrario:

**Bool = (DT(Did1) == DT(Did2))**

Especificación formal:
```
DID did2 = PS.pop('data identifier');
DID did1 = PS.pop('data identifier');
if (DT(did1).tid != DT(did2).tid) then raise('TypeMismatch');
if (DT(did1).val == DT(did2).val) then PS.push('true');
else PS.push('false')
```

Errores: **TypeMismatch**
**StackUnderflow**
**InvalidIdentifier**

### 13.3.10 Igual (equal)

Descripción breve: Igualdad.

Sinopsis: **EQ_<T>**

Operandos: Ninguno.

Pila: **..., Val1, Val2 ⇨ ..., Comp**

Tipos: Cualquier tipo primitivo excepto **void** (O, S, L, W, U, F, D, B, C, I, R).

Parámetros: **Val1** y **Val2** se interpretarán como de tipo **<T>**.

**Comp** será de tipo **boolean**.

Efecto: Sustituye los dos elementos que ocupan las dos posiciones más altas de la pila por "true" si son iguales y por "false" en caso contrario:

**Comp = (Val1 == Val2)**

Especificación formal:
```
type(<T>) buf = PS.pop(<T>);
if (buf == PS.pop<T>) then PS.push('true');
else PS.push('false');
```

Errores: **StackUnderflow**

### 13.3.11 Menor que (less than)

| | |
|---|---|
| Descripción breve: | Inferioridad estricta. |
| Sinopsis: | **LT_<T>** |
| Operandos: | Ninguno. |
| Pila: | **..., Val1, Val2 ⇨ ..., Comp** |
| Tipos: | **Character** o cualquier tipo numérico (C, O, S, L, W, U, F, D). |
| Parámetros: | **Val1** y **Val2** se interpretarán como de tipo <**T**>.<br>**Comp** será de tipo **boolean**. |
| Efecto: | Sustituye los dos elementos que ocupan las dos posiciones más altas de la pila por "true" si el elemento en la cima es mayor que el siguiente y por "false" en caso contrario:<br>**Comp = (Val1 < Val2)**<br>Para comparar caracteres se utilizará el orden numérico. |
| Especificación formal: | **type(<T>) buf = PS.pop(<T>);**<br>**if (PS.pop<T> < buf) then PS.push('true');**<br>**else PS.push('false');** |
| Errores: | **StackUnderflow** |

### 13.3.12 Mayor que (greater than)

| | |
|---|---|
| Descripción breve: | Superioridad estricta. |
| Sinopsis: | **GT_<T>** |
| Operandos: | Ninguno. |
| Pila: | **..., Val1, Val2 ⇨ ..., Comp** |
| Tipos: | **Character** o cualquier tipo numérico (C, O, S, L, W, U, F, D). |
| Parámetros: | **Val1** y **Val2** se interpretarán como de tipo <**T**>.<br>**Comp** será de tipo **boolean**. |
| Efecto: | Sustituye los dos elementos que ocupan las dos posiciones más altas de la pila por "true" si el elemento en la cima es menor que el siguiente y por "false" en caso contrario:<br>**Comp = (Val1 > Val2)**<br>Para comparar caracteres se utilizará el orden numérico. |
| Especificación formal: | **type(<T>) buf = PS.pop(<T>);**<br>**if (PS.pop<T> > buf) then PS.push('true');**<br>**else PS.push('false');** |
| Errores: | **StackUnderflow** |

### 13.3.13 Sumar (o adicionar) (add)

| | |
|---|---|
| Descripción breve: | Suma (o adición) aritmética. |
| Sinopsis: | **ADD_<T>** |
| Operandos: | Ninguno. |
| Pila: | **..., Num1, Num2 ⇨ ..., Sum** |

Tipos: Cualquier tipo numérico (O, S, L, W, U, F, D).

Parámetros: **Num1** y **Num2** se interpretarán como de tipo <**T**>.

**Sum** será de tipo <**T**>.

Efecto: Sustituye los dos elementos que ocupan las dos posiciones más altas de la pila por su suma:

**Sum = Num1 + Num2**

Especificación formal:
```
type(<T>) buf = PS.pop(<T>);
buf += PS.pop(<T>);
PS.push(buf);
```

Errores: **StackUnderflow**
**ArithmeticOverflow**

### 13.3.14 Restar (o sustraer) (substract)

Descripción breve: Resta (o sustracción) aritmética.

Sinopsis: **SUB_<T>**

Operandos: Ninguno.

Pila: **..., Num1, Num2 ⇨ ..., Diff**

Tipos: Cualquier tipo numérico (O, S, L, W, U, F, D).

Parámetros: **Num1** y **Num2** se interpretarán como de tipo <**T**>.

**Diff** será de tipo <**T**>.

Efecto: Sustituye los dos elementos que ocupan las dos posiciones más altas de la pila por su diferencia:

**Diff = Num1 – Num2**

Especificación formal:
```
type(<T>) buf = PS.pop(<T>);
buf = PS.pop(<T>) –buf;
PS.push(buf);
```

Errores: **StackUnderflow**
**ArithmeticOverflow**

### 13.3.15 Multiplicar (multiply)

Descripción breve: Multiplicación aritmética.

Sinopsis: **MUL_<T>**

Operandos: Ninguno.

Pila: **..., Num1, Num2 ⇨ ..., Prod**

Tipos: Cualquier tipo numérico (O, S, L, W, U, F, D).

Parámetros: **Num1** y **Num2** se interpretarán como de tipo <**T**>.

**Prod** será de tipo <**T**>.

Efecto: Sustituye los dos elementos que ocupan las dos posiciones más altas de la pila por su producto:

**Prod = Num1 * Num2**

| | |
|---|---|
| Especificación formal: | **type(<T>) buf = PS.pop(<T>);**<br>**buf *= PS.pop(<T>);**<br>**PS.push(buf);** |
| Errores: | **StackUnderflow**<br>**ArithmeticOverflow** |

### 13.3.16 Dividir (divide)

| | |
|---|---|
| Descripción breve: | División aritmética. |
| Sinopsis: | **DIV_<T>** |
| Operandos: | Ninguno. |
| Pila: | **..., Num1, Num2 ⇨ ..., Quot** |
| Tipos: | Cualquier tipo numérico (O, S, L, W, U, F, D). |
| Parámetros: | **Num1** y **Num2** se interpretarán como de tipo <**T**>.<br>**Quot** será de tipo <**T**>. |
| Efecto: | Sustituye los dos elementos que ocupan las dos posiciones más altas de la pila por su cociente:<br>**Quot = Num1 / Num2** |
| Especificación formal: | **type(<T>) buf = PS.pop(<T>);**<br>**buf = PS.pop(<T>) / buf;**<br>**PS.push(buf);** |
| Errores: | **StackUnderflow**<br>**DivisionByZero** |

### 13.3.17 Negar (negate)

| | |
|---|---|
| Descripción breve: | Cambio de signo. |
| Sinopsis: | **NEG_<T>** |
| Operandos: | Ninguno. |
| Pila: | **..., Num ⇨ ..., Opp** |
| Tipos: | Cualquier tipo numérico con signo (S, L, F, D). |
| Parámetros: | **Num** se interpretará como de tipo <**T**>.<br>**Opp** será de tipo <**T**>. |
| Efecto: | Sustituye el elemento en la cima de la pila por su valor con el signo opuesto:<br>**Opp = –Num1** |
| Especificación formal: | **type(<T>) buf = PS.pop<T>;**<br>**PS.push(–buf);** |
| Errores: | **StackUnderflow** |

### 13.3.18 Residuo (remainder)

| | |
|---|---|
| Descripción breve: | Residuo aritmético. |
| Sinopsis: | **REM_<T>** |

| | |
|---|---|
| Operandos: | Ninguno. |
| Pila: | **..., Num1, Num2 ⇨ ..., Rem** |
| Tipos: | Cualquier tipo entero (O, S, L, W, U). |
| Parámetros: | **Num1** y **Num2** se interpretarán como de tipo <**T**>. |
| | **Rem** será de tipo <**T**>. |
| Efecto: | Sustituye los dos elementos que ocupan las dos posiciones más altas de la pila por su residuo:<br>**Rem = Num1 % Num2** |
| Especificación formal: | **type(<T>) buf = PS.pop(<T>);**<br>**buf = PS.pop(<T>) % buf;**<br>**PS.push(buf);** |
| Errores: | **StackUnderflow**<br>**DivisionByZero** |

### 13.3.19 Repetir (o duplicar) (duplicate)

| | |
|---|---|
| Descripción breve: | Repite el valor. |
| Sinopsis: | **DUP_<T>** |
| Operandos: | Ninguno. |
| Pila: | **..., Val ⇨ ..., Val, Val** |
| Tipos: | Cualquier tipo primitivo excepto **void** (O, S, L, W, U, F, D, B, C, I, R). |
| Parámetros: | **Val** se interpretará como de tipo <**T**>. |
| Efecto: | Repite el valor que está en la cima de la pila. |
| Especificación formal: | **type(<T>) buf = PS[SP](<T>);**<br>**PS.push(buf);** |
| Errores: | **StackUnderflow** |

### 13.3.20 Convertir (convert)

| | |
|---|---|
| Descripción breve: | Convierte valor. |
| Sinopsis: | **CVT_<T1><T2>** |
| Operandos: | Ninguno. |
| Pila: | **..., Val ⇨ ..., Res** |
| Tipos: | **Boolean**, **character** o cualquier tipo numérico (O, S, L, W, U, F, D, B, C); véanse las combinaciones permitidas en 13.4. |
| Parámetros: | **Val** se interpretará como de tipo <**T1**> (tipo de fuente). |
| | **Res** será de tipo <**T2**> (tipo de destino). |
| Efecto: | Sustituye el valor en la cima de la pila por un valor equivalente en el tipo de destino. Se aplican las reglas de conversión definidas en 13.4. |

| | |
|---|---|
| Especificación formal: | **type(<T2>) buf = (type(<T2>)) (PS.pop(<T1>));**<br>**PS.push(buf);** |
| Errores: | **StackUnderflow** |

### 13.3.21 Salto con verdadero (jump on true)

| | |
|---|---|
| Descripción breve: | Salto corto condicional "if". |
| Sinopsis: | **JT Off** |
| Operandos: | **Off** será un desplazamiento (offset) con signo representada por un octeto (en notación de complemento de dos) que especifica el número de instrucciones que habrán de saltarse hacia adelante o hacia atrás en la rutina actual. |
| Pila: | **..., Test ⇨ ...** |
| Tipos: | No aplicable. |
| Parámetros: | **Test** se interpretará como de tipo **boolean**. |
| Efecto: | Si el valor en la cima de la pila es "true" entonces:<br>si **Off** es positivo, se saltan **Off** instrucciones hacia adelante, y,<br>si **Off** es negativo, se saltan -**Off** instrucciones hacia atrás. |
| Especificación formal: | **if (PS.pop('boolean')) then IP += Off;** |
| Errores: | **StackUnderflow**<br>**JumpOutOfRange** |

### 13.3.22 Salto con falso (jump on false)

| | |
|---|---|
| Descripción breve: | Salto corto condicional "else". |
| Sinopsis: | **JF Off** |
| Operandos: | **Off** será un desplazamiento con signo representada por un octeto (en notación de complemento de dos) que especifica el número de instrucciones que habrán de saltarse hacia adelante o hacia atrás en la rutina actual. |
| Pila: | **..., Test ⇨ ...** |
| Tipos: | No aplicable. |
| Parámetros: | **Test** se interpretará como de tipo **boolean**. |
| Efecto: | Si el valor en la cima de la pila es "false" entonces:<br>si **Off** es positivo, se saltan **Off** instrucciones hacia adelante, y,<br>si **Off** es negativo, se saltan -**Off** instrucciones hacia atrás. |
| Especificación formal: | **if ! (PS.pop('boolean')) then IP += Off;** |
| Errores: | **StackUnderflow**<br>**JumpOutOfRange** |

### 13.3.23 Salto (jump)

| | |
|---|---|
| Descripción breve: | Salto corto incondicional. |
| Sinopsis: | **JMP Off** |
| Operandos: | **Off** será un desplazamiento con signo representada por un octeto (en notación de complemento de dos) que especifica el número de instrucciones que habrán de saltarse hacia adelante o hacia atrás en la rutina actual. |
| Pila: | **...** ⇨ **...** |
| Tipos: | No aplicable. |
| Parámetros: | Ninguno. |
| Efecto: | Si **Off** es positivo, se saltan **Off** instrucciones hacia adelante;<br>si **Off** es negativo, se saltan -**Off** instrucciones hacia atrás. |
| Especificación formal: | **IP += Off;** |
| Errores: | **JumpOutOfRange** |

### 13.3.24 Desplazar (shift)

| | |
|---|---|
| Descripción breve: | Desplazamiento lógico. |
| Sinopsis: | **SHIFT_<T> Off** |
| Operandos: | **Off** será un desplazamiento con signo representado por un octeto (en notación de complemento de dos) que especifica el número de posiciones de bit que habrá que desplazar el parámetro hacia la izquierda o hacia la derecha. |
| Pila: | **..., Val** ⇨ **..., Pwr** |
| Tipos: | Cualquier tipo entero sin signo (O, W, U). |
| Parámetros: | **Val** se interpretará como de tipo <**T**>.<br>**Pwr** será de tipo <**T**>. |
| Efecto: | Se sustituye el elemento en la cima de la pila por su valor desplazado **Off** bits hacia la derecha si **Off** es positivo, u -**Off** bits hacia la izquierda si **Off** es negativo. Si **Off** está fuera de gama, es resultado no está especificado. |
| Especificación formal: | **type(<T>) buf = PS.pop(<T>);**<br>**if (Off >=0) then buf >>= Off;**<br>**else if (buf < 0)**<br>**buf = –(–buf << –Off);**<br>**else buf <<= –Off;**<br>**PS.push(buf);** |
| Errores: | **StackUnderflow**<br>**ShiftOutOfRange** |

### 13.3.25 Obtener referencia de objeto (get object reference)

| | |
|---|---|
| Descripción breve: | Obtiene la referencia al objeto inicial de un lote. |
| Sinopsis: | **GETOR Pid** |

| | |
|---|---|
| Operandos: | **Pid** será la representación en un octeto de un identificador de lote que especifica el lote a que habrá de ganarse acceso. |
| Pila: | **... ⇨ ..., Obref** |
| Tipos: | No aplicable. |
| Parámetros: | **Obref** será de tipo **object reference**. |
| Efecto: | Extrae una referencia de objeto al objeto inicial del lote. |
| Especificación formal: | **if (PT[PID].sts = 'not available') then raise('BadPackageStatus');**<br>**PS.push(PT[PID].or);** |
| Errores: | **InvalidIdentifier**<br>**BadPackageStatus** |

### 13.3.26 Salto largo con verdadero (long jump on true)

| | |
|---|---|
| Descripción breve: | Salto largo condicional "if". |
| Sinopsis: | **LJT Off** |
| Operandos: | **Off** será un desplazamiento con signo representado por dos octetos (en notación de complemento de dos) que especifica el número de instrucciones que habrán de saltarse hacia adelante o hacia atrás en la rutina actual. |
| Pila: | **..., Test ⇨ ...** |
| Tipos: | No aplicable. |
| Parámetros: | **Test** se interpretará como de tipo **boolean**. |
| Efecto: | Si el elemento en la cima de la pila es "true", entonces:<br>si **Off** es positivo, se saltan **Off** instrucciones hacia adelante, y,<br>si **Off** es negativo, se saltan -**Off** instrucciones hacia atrás. |
| Especificación formal: | **if (PS.pop('boolean')) then IP += Off;** |
| Errores: | **StackUnderflow**<br>**JumpOutOfRange** |

### 13.3.27 Salto largo con falso (long jump on false)

| | |
|---|---|
| Descripción breve: | Salto largo condicional "else". |
| Sinopsis: | **LJF Off** |
| Operandos: | **Off** será un desplazamiento con signo representado por dos octetos (en notación de complemento de dos) que especifica el número de instrucciones que habrán de saltarse hacia adelante o hacia atrás en la rutina actual. |
| Pila: | **..., Test ⇨ ...** |
| Tipos: | No aplicable. |
| Parámetros: | **Test** se interpretará como de tipo **boolean**. |
| Efecto: | Si el elemento en la cima de la pila es "false", entonces: |

si **Off** es positivo, se saltan **Off** instrucciones hacia adelante, y,

si **Off** es negativo, se saltan -**Off** instrucciones hacia atrás.

Especificación formal: **if ! (PS.pop('boolean')) then IP += Off;**

Errores: **StackUnderflow**
**JumpOutOfRange**

### 13.3.28 Salto largo (long jump)

Descripción breve: Salto largo incondicional.

Sinopsis: **LJMP Off**

Operandos: **Off** será un desplazamiento con signo representado por dos octetos (en notación de complemento de dos) que especifica el número de instrucciones que habrán de saltarse hacia adelante o hacia atrás en la rutina actual.

Pila: **... ⇨ ...**

Tipos: No aplicable.

Parámetros: Ninguno.

Efecto: Si **Off** es positivo, se saltan **Off** instrucciones hacia adelante;

si **Off** es negativo, se saltan -**Off** instrucciones hacia atrás.

Especificación formal: **IP += Off;**

Errores: **JumpOutOfRange**

### 13.3.29 Llamada (call)

Descripción breve: Llama rutina.

Sinopsis: **CALL Fid**

Operandos: **Fid** será la representación en dos octetos de un identificador de función que especifica la rutina que ha de invocarse.

Pila: **..., ParN, ... , Par1 ⇨ ...**

Tipos: No aplicable.

Parámetros: **Par1**, ..., **ParN** (donde N es el número de parámetros de la rutina) son los parámetros reales de la rutina. Se interpretarán como del mismo tipo que los parámetros formales de la rutina cuando se pasan por valor, y se interpretarán como de tipo **data identifier** y harán referencia a una variable del mismo tipo de los parámetros formales cuando se pasan por referencia.

Efecto: Saca los elementos que ocupan las posiciones más altas de la pila e invoca la rutina especificada por Fid, con estos elementos como parámetros reales. En el caso de parámetros pasados por referencia, comprueba que el identificador de datos no hace referencia a una variable local y apunta a datos del mismo tipo que el de la firma. Inserta una trama en la pila de llamadas con el contexto actual. Inicializa la tabla de variables locales para la rutina. Fija el puntero de instrucciones a la primera instrucción de la rutina.

Especificación formal:

```
TID tid;
CS.push(IP, FR, SP, LT);
FR = Fid;
LT = RT[Fid].LT;
for (short i = 0; i < RT[Fid].nbp; i--;)
{
        switch (RT[Fid].sig[i].mod)
        {
        case 'value':
            tid = RT[Fid].sig[i].TID;
            break;
        case 'reference':
            tid == 'data identifier';
            if (8000h <= PS[SP](tid) < 8100h)
            then raise('InvalidParameter');
            if (RT[Fid].sig[i].TID !=
            DT(PS[SP](tid)).TID)
            then raise('TypeMismatch');
            break;
        };
        LT[I+0x8000].val = PS.pop(tid);
};
IP = RT[Fid].IP;
```

Errores:
**InvalidIdentifier**
**StackUnderflow**
**TypeMismatch**
**InvalidParameter**

### 13.3.30 Llamada externa (call)

Descripción breve: Llama función externa.

Sinopsis: **XCALL Fid**

Operandos: **Fid** será la representación en dos octetos de un identificador de función que especifica el servicio o función predefinida que ha de invocarse.

Pila: **..., ParN, ... , Par1 ⇨ ..., (Ret)**

Tipos: No aplicable.

Parámetros: **Par1** se interpretará como de tipo **object reference**. Indica la referencia de la instancia IDL a que se aplica la operación.

**Par2**, ..., **ParN** (donde N es el número de parámetros de la función, más 1) son los parámetros reales de la función. Cualquiera que sea el modo de paso, los parámetros se interpretarán como de tipo **data identifier**.

Si la función tiene un tipo de valor de retorno diferente de **void**, **Ret** será de este tipo.

Efecto: Comprueba que los parámetros hacen referencia a datos del mismo tipo que los de la firma de la función. Saca los elementos que ocupan las posiciones más altas de la pila e invoca la función externa especificada por Fid, con estos elementos como parámetros reales. Inserta una trama en la pila de llamadas con el contexto actual. Pasa parámetros a la función externa e invoca esta función. Si la invocación es asíncrona, saca un

elemento de la pila de llamadas tan pronto como se acusa recibo de la petición. Si la invocación es síncrona, espera hasta la conclusión de la petición. Si se señala una excepción, activa el manejador de la excepción. De lo contrario, extrae los parámetros de salida y el valor de retorno de la función, inserta el valor de retorno en la pila de parámetros y saca una trama de la pila de llamadas.

Especificación formal:

```
DID buf[ST[Fid].nbp];
Object obref = PS.pop('object reference');
for (short i = 0; i < ST[Fid].nbp; i++;)
{
        if (ST[Fid].sig[i].TID != DT(PS[SP]('data identifier').TID))

        then raise('TypeMismatch');
        buf[I] = PS.pop('data identifier');
};
CS.push(IP, FR, SP, LT);
FR = Fid;
LT[0].tid = 'object reference';
LT[0].val = obref;
for (short i = 1;i < ST[Fid].nbp; i++}
        LT[i].TID = 'data identifier';
        LT[i].val = buf[i];
}
short Pid = (Fid>>8)–64;
if (PT[Pid].sts != 'available') then raise('BadPackageStatus');
_open_package(PT[Pid].name);

// open a context to invoke the service
_pass_in_parameter(LT[0]);
// according to the platform mapping specification procedure
for (short i=1; i<ST[Fid].nbp; i++;)
        switch(ST[Fid].sig[i].mod)
        {
        case 'in': _pass_in_parameter (LT[i]);
        case 'out': _pass_out_parameter (LT[i]);
        case 'inout': _pass_inout_parameter (LT[i]);
        };

if (ST[Fid].mod == 'asynchronous') then
{
        _invoke_operation(PT[Pid].name, ST[Fid].name);
        {IP, FR, SP, LT} = CS.pop();
}
else
{
```

```
                result = _invoke_operation(PT[Pid].name, ST[Fid].name);
                if (result == 'ok')          then
                {
                    _retrieve_out_parameter();
                    {IP, FR, SP, LT} = CS.pop();
                    if (ST[Fid].TID != 'void') then
                        PS.push(_retrieve_return_value());
                }
                else  // the result is an exception formatted as a message
                {
                    FR = HT[result.MID];
                    LT = result.LT;

                    IP = RT[FT].IP;
                }
        }

        _close_package(PT[Pid].name);
        // close service invocation context
```

Errores: **InvalidIdentifier**
**StackUnderflow**
**TypeMismatch**
**BadPackageStatus**
**InvalidObjectReference**

### 13.3.31 Insertar (en una pila) (push)

Descripción breve: Inserta un valor de datos en una pila.

Sinopsis: **PUSH Did**

Operandos: **Did** será la representación en dos octetos de un identificador de datos que contiene el valor que se va a insertar en la pila.

Pila: **... ⇨ ..., Val**

Tipos: No aplicable.

Parámetros: **Val** será del mismo tipo que el de la constante o variable identificada por **Did**.

Efecto: Comprueba que **Did** identifica una constante o variable de un tipo primitivo. Inserta en la pila de parámetros el valor de la constante o variable cuyo identificador de datos es **Did**.

Especificación formal:
```
if (DT(Did).tid > 'object reference') then raise('InvalidType');
PS.push(DT(Did).val);
```

Errores: **InvalidIdentifier**
**InvalidType**

### 13.3.32 Insertar referencia (en una pila) (push reference)

Descripción breve: Inserta un identificador de datos en una pila.

Sinopsis: **PUSHR Did**

Operandos: **Did** será la representación en dos octetos de un identificador de datos que se va a insertar en la pila.

Pila: **... ⇨ ..., Val**

Tipos: No aplicable.

| | |
|---|---|
| Parámetros: | **Val** será de tipo **data identifier**. |
| Efecto: | Inserta **Did** en la pila de parámetros. |
| Especificación formal: | **PS.push(Did);** |
| Errores: | Ninguno. |

### 13.3.33 Insertar valor inmediato (en una pila) (push immediate)

| | |
|---|---|
| Descripción breve: | Inserta un número entero de tipo short en una pila. |
| Sinopsis: | **PUSHI Int** |
| Operandos: | **Int** será la representación en dos octetos de un valor entero short con signo que se va a insertar en la pila. |
| Pila: | **... ⇨ ..., Val** |
| Tipos: | No aplicable. |
| Parámetros: | **Val** será de tipo **short**. |
| Efecto: | Inserta **Int** en la pila de parámetros. |
| Especificación formal: | **PS.push(Int);** |
| Errores: | Ninguno. |

### 13.3.34 Sacar (de una pila) (pop)

| | |
|---|---|
| Descripción breve: | Saca un valor de una pila y lo asigna a una variable. |
| Sinopsis: | **POP Did** |
| Operandos: | **Did** será la representación en dos octetos del identificador de datos de la variable a que se va a asignar el elemento que está en la cima de la pila. |
| Pila: | **..., Val ⇨ ...** |
| Tipos: | No aplicable. |
| Parámetros: | **Val** se interpretará como del tipo de la variable identificada por **Did**. |
| Efecto: | Comprueba que **Did** identifica una variable de un tipo primitivo. Saca **Val** de la pila de parámetros y lo asigna a la variable identificada por **Did**. |
| Especificación formal: | **TID tid = VT(Did).TID;**<br>**if (tid > 'object reference') then raise('InvalidType')**<br>**VT(Did).val = PS.pop(tid);** |
| Errores: | **InvalidIdentifier**<br>**StackUnderflow**<br>**InvalidType** |

### 13.3.35 Sacar referencia (de una pila) (pop reference)

| | |
|---|---|
| Descripción breve: | Saca un valor de una pila y lo asigna a la variable referenciada por una variable. |
| Sinopsis: | **POPR Did** |

| | |
|---|---|
| Operandos: | **Did** será la representación en dos octetos del identificador de datos de la variable de tipo **data identifier**, cuyo calor identifica la variable a la que se va a asignar el elemento que está en la cima de la pila. |
| Pila: | **..., Val ⇨ ...** |
| Tipos: | No aplicable. |
| Parámetros: | **Val** se interpretará como de tipo de **VT(Did).val**. |
| Efecto: | Comprueba que **Did** identifica una variable de un tipo **data identifier**. Saca **Val** de la pila de parámetros y lo asigna a la variable identificada por **Did**. |
| Especificación formal: | **TID tid = type(VT(Did).val.TID);**<br>**if (tid != 'data identifier') then raise('InvalidType');**<br>**VT(VT(Did).val).val = PS.pop(tid);** |
| Errores: | **InvalidIdentifier**<br>**StackUnderflow**<br>**InvalidType** |

### 13.3.36 Sacar contenido (de una pila) (pop contents)

| | |
|---|---|
| Descripción breve: | Saca una variable de una pila y asigna su valor a una variable. |
| Sinopsis: | **POPC Did1** |
| Operandos: | **Did1** será la representación en dos octetos del identificador de datos de la variable a la cual se habrá de asignar el valor de los datos identificados por el elemento que está en la cima de la pila. |
| Pila: | **..., Did2 ⇨ ...** |
| Tipos: | No aplicable. |
| Parámetros: | **Did2** se interpretará como del tipo **data identifier**. Los datos identificados por **Did2** se interpretarán como del tipo de los datos identificados por **Did1**. |
| Efecto: | Comprueba que **Did1** y **Did2** identifican datos del mismo tipo. Saca **Did2** de la pila de parámetros y asigna el valor de **Did2** a la variable identificada por **Did1**. |
| Especificación formal: | **DID did2 = PS.pop('data identifier');**<br>**if (VT(Did1).TID != DT(did2).TID) then raise('TypeMismatch');**<br>**VT(Did1).val = DT(did2).val;** |
| Errores: | **InvalidIdentifier**<br>**StackUnderflow**<br>**TypeMismatch** |

### 13.3.37 Atribuir (allocate)

| | |
|---|---|
| Descripción breve: | Crea variable dinámica. |
| Sinopsis: | **ALLOC Tid** |
| Operandos: | **Tid** será la representación en dos octetos del identificador de tipo que especifica el tipo de la variable dinámica que se va a atribuir. |
| Pila: | **..., ⇨ ..., Did** |
| Tipos: | No aplicable. |
| Parámetros: | **Did** será del tipo **data identifier**. |

| | |
|---|---|
| Efecto: | Genera una variable dinámica cuyo tipo se identifica por **Tid**. Inserta su identificador de datos en la pila de parámetros. |
| Especificación formal: | **DID did = new(Tid);**<br>**VT[did].val = 'null';  // default value for the type**<br>**VT[did].TID = Tid;**<br>**PS.push(did);** |
| Errores: | **AllocationFailed**<br>**InvalidIdentifier** |

### 13.3.38 Incrementar (increment)

| | |
|---|---|
| Descripción breve: | Incrementa variable. |
| Sinopsis: | **INC Did** |
| Operandos: | **Did** será la representación en dos octetos del identificador de datos de la variable que habrá de ser incrementada. |
| Pila: | **..., Val ⇨ ...,** |
| Tipos: | No aplicable. |
| Parámetros: | **Val** se interpretará como del tipo de la variable identificada por **Did**. |
| Efecto: | Comprueba que **Did** identifica una variable de tipo numérico. Saca un elemento de la pila de parámetros y utiliza su valor para incrementar en esa cantidad el valor de la variable identificada por **Did**. |
| Especificación formal: | **TID tid = VT(Did).TID;**<br>**if (VT(Did).TID > 'double') raise('InvalidType');**<br>**VT(Did).val += PS.pop(<T>);** |
| Errores: | **InvalidIdentifier**<br>**StackUnderflow**<br>**InvalidType**<br>**ArithmeticOverflow** |

### 13.3.39 Decrementar (decrement)

| | |
|---|---|
| Descripción breve: | Decrementa variable. |
| Sinopsis: | **DEC Did** |
| Operandos: | **Did** será la representación en dos octetos del identificador de datos de la variable que habrá de ser decrementada. La variable decrementada por **Did** será de un tipo numérico. |
| Pila: | **..., Val ⇨ ...,** |
| Tipos: | No aplicable. |
| Parámetros: | **Val** se interpretará como del tipo de la variable identificada por **Did**. |
| Efecto: | Comprueba que **Did** identifica una variable de tipo numérico. Saca un elemento de la pila de parámetros y utiliza su valor para decrementar en esa cantidad el valor de la variable identificada por **Did**. |
| Especificación formal: | **TID tid = VT(Did).TID;**<br>**if (VT(Did).TID > 'double') raise('InvalidType');**<br>**VT(Did).val -= PS.pop(<T>);** |

Errores: **InvalidIdentifier**
**StackUnderflow**
**InvalidType**
**ArithmeticOverflow**

### 13.3.40 Obtener (get)

Descripción breve: Obtiene valor de elemento de datos de tipo construido.

Sinopsis: **GET Did Lvl**

Operandos: **Did** será la representación en dos octetos del identificador de datos de los datos a que se va a ganar acceso.

**Lvl** será la cantidad sin signo expresada por un octeto que representa el número de niveles anidados que hay que atravesar para ganar acceso al valor buscado.

Pila: **..., Idx(Lvl), ..., Idx(1) ⇨ ..., Val**

Tipos: No aplicable.

Parámetros: **Idx(1)**, … **Idx(Lvl)** se interpretarán como de tipo **unsigned short**.

**Val** será del mismo tipo que el elemento a que se gana acceso.

Efecto: Sustituye una lista de índices en la pila de parámetros por el valor del elemento direccionado por los índices sacados de la pila, dentro de los datos de tipo construido identificados por **Did**:

**Val = DT(Did)[Idx(1),...,Idx(Lvl)]**

Comprueba que el elemento a que se va a ganar acceso es de un tipo primitivo. Si **Lvl** es igual a 0, ejecuta una instrucción **PUSH**.

Especificación formal:

```
void *buf = &DT(Did);
unsigned short idx;
for (;Lvl>0; Lvl--;)
{
        if (buf->TID <= 'object reference') then   raise('InvalidLevel');
        idx = PS.pop('unsigned short');
        if (buf->lg < idx) then raise         ('InvalidIndex');
        buf = &buf->.val[idx];
}
if (buf->TID > 'object reference') then raise('InvalidType');
PS.push(buf->val);
```

Errores: **InvalidIdentifier**
**InvalidLevel**
**StackUnderflow**
**InvalidIndex**
**InvalidType**

### 13.3.41 Obtener contenido (get contents)

Descripción breve: Fija el contenido de datos a un elemento de datos de tipo construido.

Sinopsis: **GETC Did1 Lvl**

Operandos: **Didl** será la representación en dos octetos del identificador de datos de los datos a que se va a ganar acceso.

**Lvl** será la cantidad sin signo expresada por un octeto que representa el número de niveles anidados que hay que atravesar para ganar acceso al elemento deseado.

Pila: **..., Did2, Idx(Lvl), ..., Idx(1) ⇨ ...,**

Tipos: No aplicable.

Parámetros: **Idx(1)**, ... **Idx(Lvl)** se interpretarán como de tipo **unsigned short.**

Efecto: Saca de la pila de parámetros una lista de índices y un identificador de datos **Did2**. Dentro de los datos de tipo construido identificados por **Did1**, asigna la variable identificada por **Did2** al elemento direccionado por la lista de índices sacada de la pila:

**VT(Did2).Val = DT(Did1)[Idx(1),...,Idx(Lvl)]**

Comprueba que el elemento a que se va a ganar acceso es del mismo tipo que los datos identificados por **Did2**.

Especificación formal:

```
void *buf = &DT(Did1);
unsigned short idx;
for (;Lvl>0; Lvl--;)
{
        if (buf->TID <= 'object reference') then   raise('InvalidLevel');
        idx = PS.pop('unsigned short');
        if (buf->lg < idx) then raise         ('InvalidIndex');
        buf = &buf->.val[idx];
}
DID did2 = PS.pop('data identifier');
if (VT(did2).TID != buf->TID) then raise('TypeMismatch');
VT(did2).val = buf->val;
```

Errores: **InvalidIdentifier**
**InvalidLevel**
**StackUnderflow**
**InvalidIndex**
**TypeMismatch**

### 13.3.42 Fijar (set)

Descripción breve: Fija un elemento de una variable de tipo construido a un valor.

Sinopsis: **SET Did Lvl**

Operandos: **Did** será la representación en dos octetos del identificador de datos de la variable que se va a modificar.

**Lvl** será la cantidad sin signo expresada por un octeto que representa el número de niveles anidados que hay que atravesar para ganar acceso al elemento que se va a modificar.

Pila: **..., Val, Idx(Lvl), ..., Idx(1) ⇨ ...,**

Tipos: No aplicable.

Parámetros: **Idx(1)**, ... **Idx(Lvl)** se interpretarán como de tipo **unsigned short.**

**Val** se interpretará como del mismo tipo que el elemento que se va a modificar.

Efecto: Saca de la pila de parámetros una lista de índices y un valor. Dentro de la variable estructurada identificada por **Did**, asigna el elemento direccionado por la lista de índices sacada de la pila al valor sacado de la pila:

**VT(Did)[Idx(1),...,Idx(Lvl)] = Val**

Comprueba que el elemento que se va a modificar es de un tipo primitivo. Si **Lvl** es igual a 0, se ejecuta una instrucción **POP**.

Especificación formal:

```
void *buf = &VT(Did);
unsigned short idx;
for (;Lvl>0; Lvl--;)
{
        if (buf->TID <= 'object reference') then   raise('InvalidLevel');
        idx = PS.pop('unsigned short');
        if (buf->lg < idx) then raise         ('InvalidIndex');
        buf = &buf->.val[idx];
}
if (buf->TID > 'object reference') then raise('InvalidType');
buf->val = PS.pop(buf->TID);
```

Errores:
**Invalid Identifier**
**InvalidLevel**
**StackUnderflow**
**InvalidIndex**
**InvalidType**

### 13.3.43 Fijar contenido (set contents)

Descripción breve: Fija un elemento de variable de tipo construido a un contenido de datos.

Sinopsis: **SETC Did1 Lvl**

Operandos: **Did1** será la representación en dos octetos del identificador de datos de la variable que se va a modificar.

**Lvl** será la cantidad sin signo expresada por un octeto que representa el número de niveles anidados que hay que atravesar para ganar acceso al elemento que se va a modificar.

Pila: **..., Did2, Idx(Lvl), ..., Idx(1)** ⇨ **...,**

Tipos: No aplicable.

Parámetros: **Idx(1)**, ... **Idx(Lvl)** se interpretarán como de tipo **unsigned short.**

**Did2** se interpretará como de tipo **data identifier**.

Efecto: Saca de la pila de parámetros una lista de índices y un identificador de datos. Dentro de la variable estructurada identificada por **Did1**, asigna el elemento direccionado por la lista de índices sacada de la pila al valor identificado por los datos sacados de la pila:

**VT(Did1)[Idx(1),...,Idx(Lvl)] = DT(Did2).**

Comprueba que **Did2** identifica un dato del tipo del elemento que se va a modificar. Si **Lvl** es igual a 0, ejecuta una instrucción **POPC**.

Especificación formal:

```
void *buf = VT(Did1);
unsigned short idx;
for (;Lvl>0; Lvl--;)
{
        if (buf->TID <= 'object reference') then   raise('InvalidLevel');
        idx = PS.pop('unsigned short');
        if (buf->lg < idx) then raise        ('InvalidIndex');
        buf = &buf->.val[idx];
}
DID did2 = PS.pop('data identifier');
if (DT(did2).TID != buf->TID) then raise('TypeMismatch');
buf->val = DT(did2).val;
```

Errores:

**InvalidIdentifier**
**InvalidLevel**
**StackUnderflow**
**InvalidIndex**
**TypeMismatch**

## 13.4 Reglas de conversión de tipo

Esta subcláusula define las reglas que deberán aplicarse cuando se utiliza una instrucción **convert (CVT)** para convertir un valor de la pila de parámetros (en consecuencia, un valor de un tipo primitivo) de un tipo de fuente a un tipo de destino.

Los valores de los tipos **data identifier** y **object reference** no se convertirán a valores de otros tipos, ni a la inversa.

En lo que respecta a los otros tipos primitivos, no todas las conversiones de tipos están permitidas; sin embargo, un valor de cualquiera de estos tipos puede convertirse en un valor de cualquiera de los otros tipos mediante una secuencia de conversiones.

El cuadro 4 muestra las conversiones de tipo permitidas, así como el número de la subcláusula en que se definen:

**Cuadro 4/T.173 – Conversiones de tipos**

| Fuente/destino | O | S | L | W | U | F | D | B | C |
|---|---|---|---|---|---|---|---|---|---|
| **O** | N/A | 13.4.2.2 | N/A | 13.4.2.2 | N/A | N/A | N/A | 13.4.4 | N/A |
| **S** | N/A | N/A | 13.4.2.3 | 13.4.1 | 13.4.3 | N/A | N/A | 13.4.4 | N/A |
| **L** | N/A | 13.4.5 | N/A | N/A | 13.4.1 | 13.4.2.3 | N/A | 13.4.4 | N/A |
| **W** | 13.4.5 | 13.4.1 | 14.4.2.3 | N/A | 13.4.2.3 | N/A | N/A | 13.4.4 | 13.4.1 |
| **U** | N/A | N/A | 13.4.1 | 13.4.5 | N/A | 13.4.2.3 | N/A | 13.4.4 | N/A |
| **F** | N/A | N/A | 13.4.6 | N/A | 13.4.6 | N/A | 13.4.2.3 | N/A | N/A |
| **D** | N/A | N/A | N/A | N/A | N/A | 13.4.5 | N/A | N/A | N/A |
| **B** | 13.4.2.1 | 13.4.2.1 | N/A | N/A | N/A | N/A | N/A | N/A | N/A |
| **C** | N/A | N/A | N/A | 13.4.1 | N/A | N/A | N/A | N/A | N/A |
| N/A No aplicable. | | | | | | | | | |

### 13.4.1 Conversiones reversibles

Las siguientes conversiones no entrañan pérdida de información cuando se efectúan en sentido inverso:

– entre **unsigned short** y **character** (WC, CW);

– entre **short** y **unsigned short** (SW, WS);

– entre **long** y **unsigned long** (LU, UL).

Para todas estas conversiones, el resultado de la conversión será el valor del tipo de destino que tiene la misma representación en notación de complemento de dos que el valor de fuente.

### 13.4.2 Extensiones sin pérdida de información

Las siguientes conversiones extienden el valor de fuente sin causar pérdida de información:

– a partir de **boolean** (BO, BS) (véase 13.4.2.1);

– de **octet** a un tipo numérico (OS, OW) (véase 13.4.2.2);

– de un tipo numérico con signo a un tipo numérico con signo con una gama mayor (SL, LF, FD) (véase 13.4.2.3);

– de un tipo numérico sin signo a cualquier tipo numérico con unas gama mayor (WL, WU, UF) (véase 13.4.2.3).

#### 13.4.2.1 Conversiones a partir del tipo boolean

Si el valor del tipo **boolean** de fuente es falso, el valor del tipo de destino será 0.

Si el valor del tipo **boolean** de fuente es verdadero, el valor del tipo de destino será el valor que corresponda a todos los bits fijados a 1 (en notación de complemento de dos), es decir:

– 255 para un tipo de destino **octet**;

– –1 para un tipo de destino **short**.

#### 13.4.2.2 Conversiones del tipo octet a un tipo numérico

El valor en el tipo de destino será el valor del octeto.

#### 13.4.2.3 Conversiones sin pérdida de información de un tipo numérico a un tipo numérico de mayor gama

El valor en el tipo de destino será el mismo valor numérico que el valor en el tipo de fuente.

### 13.4.3 Extensiones con pérdida de información

La conversión de **short** a **unsigned long** (SU) se efectuará como sigue:

– si el valor de fuente es positivo o nulo, el valor de destino será el mismo valor numérico que el valor de fuente;

– si el valor de fuente es estrictamente negativo, el valor de destino no está especificado.

### 13.4.4 Truncamiento al tipo boolean

Los truncamientos de un tipo **octet** o de un tipo numérico a **boolean** (OB, SB, WB, LB, UB) se efectuarán como sigue:

– si el valor de fuente es 0, el valor de destino será "false";

– si el valor de fuente es diferente de 0, el valor de destino será "true".

### 13.4.5 Truncamiento entre tipos enteros o entre tipos coma flotante

Los truncamientos de un tipo entero a un tipo octet o entero (WO, LS, UW), o de un tipo coma flotante a otro tipo coma flotante (DF) se efectuará como sigue:

– si el valor de fuente está comprendido en la gama del tipo de destino, el valor de destino será el mismo valor numérico que el valor de fuente;
– de lo contrario, el valor de destino no está especificado.

### 13.4.6 Truncamiento de tipo coma flotante a entero

Los truncamientos de un tipo coma flotante a un tipo entero (FL, FU) se efectuará como sigue:

– primero, se trunca entre la parte entera y la decimal, con lo que se obtiene un valor entero (redondeo decreciente) y se desecha la parte decimal;
– después, al valor entero así obtenido se le aplican las reglas indicadas en 13.4.5.

## 14 Correspondencia de IDL a MHEG-SIR

Esta cláusula define cómo una especificación IDL deberá hacerse corresponder con las declaraciones de un guión intercambiado, cuando esta especificación IDL habrá de ser utilizada por el guión como un proveedor de servicio externo.

Esta cláusula define la correspondencia, con declaraciones MHEG-SIR, de:

– interfaces y módulos IDL;
– tipos IDL:
– constantes IDL;
– referencias a objetos IDL;
– operaciones IDL;
– atributos IDL;
– excepciones IDL.

### 14.1 Especificaciones IDL

Una especificación IDL deberá hacerse corresponder con una **PackageDeclaration** MHEG-SIR declarada como un componente de un componente **external-package-declarations** del **InterchangedScript**. El nombre de la especificación IDL se hará corresponder con el componente **name** de esta declaración de lote.

NOTA – Ejemplos de especificaciones IDL son MHEG API, MPEG/DSM-CC.

Si el número de operaciones o excepciones de una especificación IDL excede el tamaño de un lote, la especificación se dividirá en varios lotes que tendrán el mismo nombre, pero diferentes identificadores MHEG-SIR.

### 14.2 Interfaces y módulos IDL

Puesto que la declaración de lote es una organización no segmentada, no hay ni una correspondencia para un módulo IDL ni para una interfaz IDL. Sin embargo, para cada invocación de función que describa una operación IDL deberá proporcionarse, como un parámetro implícito, una referencia a la interfaz en la que se produce la incrustación (es decir, un parámetro de tipo **Object**).

### 14.3 Operaciones IDL

Una operación IDL se hará corresponder con un componente **services** MHEG-SIR de la declaración de lote que hace corresponder la especificación IDL a que pertenece la operación.

### 14.3.1 Nombre de operación

El nombre global de una operación IDL se hará corresponder con el componente **name** MHEG-SIR de esta descripción de servicio.

### 14.3.2 Parámetros de operación

Los parámetros de una operación IDL se harán corresponder con el componente **parameters-description** de la descripción de servicio. En una **ServiceParameterDescription**, cada tipo de parámetro IDL se hará corresponder con el componente **type** que identifica un tipo declarado de acuerdo con las reglas de correspondencia de tipos definidas en esta cláusula. El modo de paso, según la IDL, para un parámetro se hará corresponder con el componente **passing-mode** de la correspondiente descripción de parámetro de servicio MHEG-SIR.

Si la operación no tiene un parámetro de salida ni un valor de retorno y está diseñada específicamente para retornar varias excepciones en secuencia (por ejemplo, para fines de notificación), el valor de su componente **calling-mode** debe ser **asynchronous**. De lo contrario, el valor del componente **calling-mode** será **synchronous**.

Si está previsto que una operación semánticamente síncrona señale varias excepciones en secuencia, debe dividirse en dos operaciones MHEG-SIR: una **synchronous** y la otra **asynchronous**.

### 14.3.3 Parámetro implícito

Cuando una operación IDL se hace corresponder con una descripción de servicio MHEG-SIR, la instancia de objeto a que se aplica la operación deberá seguir siendo un parámetro implícito, es decir, no se expresará como parte de la firma del servicio.

NOTA – Sin embargo, al invocar la operación, este parámetro se proporciona como el parámetro real inicial, como si su **type** fuera **object reference** y su **passing-mode** fuera **in**.

### 14.3.4 Valor de retorno

El tipo valor de retorno de una operación IDL se hará corresponder con el componente **return-value-type** de la descripción de servicio.

## 14.4 Atributos IDL

Un atributo IDL se hará corresponder a dos descripciones de servicios dentro de una declaración de lote: un servicio "accesor", cuya función es obtener el valor del atributo, y un servicio "modificador", cuya función es fijar el valor del atributo.

### 14.4.1 Accesor

El lo que respecta al servicio "accesor", el nombre de atributo IDL global cuyo identificador final lleva el prefijo "get_" se hará corresponder con el componente **name** MHEG-SIR de la descripción de servicio. Un servicio "accesor" no tendrá un parámetro explícito. El tipo de atributo IDL se hará corresponder con el componente **return-value-type** de la descripción de servicio.

Ejemplo: En la MHEG-3 API, el atributo **routine_id** del objeto **RoutineInvocation** se hará corresponder con el nombre global IDL **MHEG_3::RoutineInvocation::get_RoutineId**.

### 14.4.2 Modificador

El lo que respecta al servicio "modificador", el nombre de atributo IDL global cuyo identificador final lleva el prefijo "set_" se hará corresponder con el componente **name** MHEG-SIR de la descripción de servicio. Un servicio modificador tendrá un parámetro con el modo de paso **in**, de tal

manera que el tipo de atributo IDL se hará corresponder con el componente **type** de la descripción de parámetros para este servicio. Un servicio "modificador" no tendrá valor de retorno.

### 14.4.3 Atributo lectura solamente

Si un atributo está definido como **readonly** (lectura solamente), el servicio "accesor" será proporcionado como parte de la declaración de lote.

## 14.5 Operaciones heredadas IDL

Las operaciones IDL se harán corresponder como si estuvieran definidas en la interfaz específica.

## 14.6 Excepciones IDL

Una excepción IDL se hará corresponder con un componente **exception-description** MHEG-SIR de la declaración de lote que hace corresponder la especificación IDL a que pertenece la excepción.

### 14.6.1 Nombre de excepción

El nombre global IDL de la excepción se hará corresponder con el componente **name** MHEG-SIR de esta descripción de excepción.

### 14.6.2 Miembros de excepción

Los miembros de una excepción IDL se harán corresponder con el componente **parameters-description** de esta descripción de excepción. En esta descripción de parámetros, cada tipo de miembro IDL se hará corresponder con el componente **type** que identifica un tipo declarado de acuerdo con las reglas de correspondencia de tipo definidas en esta cláusula.

### 14.6.3 Miembro implícito

Cuando una excepción IDL se ha hecho corresponder con una descripción de excepción MHEG-SIR, la instancia de objeto de la que se origina la excepción seguirá siendo un miembro implícito, es decir, no será expresada como parte de la firma de la excepción.

NOTA – Sin embargo, al señalarse la excepción, este miembro se proporciona como el miembro real inicial como si su type fuera "object reference".

## 14.7 Tipos IDL

Un tipo IDL se hará corresponder a una **TypeDeclaration** MHEG-SIR declarada como un componente del componente **type-declarations** del **InterchangedScript**. Una declaración de tipo tendrá un alcance global en el guión intercambiado.

Los tipos básicos y los constructores IDL se harán corresponder con los tipos primitivos y los constructores MHEG-SIR, como se recapitula en el cuadro 5.

**Cuadro 5/T.173 – Correspondencia de tipos**

| IDL | MHEG-SIR |
|---|---|
| void | void |
| octet | octet |
| short | short |
| unsigned short | unsigned short |
| long | long |

**Cuadro 5/T.173 – Correspondencia de tipos** *(fin)*

| IDL | MHEG-SIR |
|---|---|
| unsigned long | unsigned long |
| float | float |
| double | double |
| boolean | boolean |
| char | character |
| enum | unsigned long |
| string | string |
| sequence | sequence |
| array | array |
| struct | structure |
| union | union |
| (object) | object reference |
| any | data identifier (véase más adelante) |

### 14.7.1 Tipo **char**

El establecimiento de la correspondencia de tipos **char** IDL a tipos **character** MHEG-SIR implica la transcodificación de valores de ISO 8859-1 a ISO/CEI 10646-1.

### 14.7.2 Tipo **enum**

No es necesario conservar la comprobación de la gama de los valores **enum.**

### 14.7.3 Tipos construidos

Una definición de tipo IDL se hará corresponder con una **TypeDescription** MHEG-SIR. Si el tipo IDL es un tipo básico o si ya ha estado sujeta a otra declaración de tipo, esta declaración de tipo consistirá en un identificador de tipo. De lo contrario, se construirá de acuerdo con las siguientes reglas de correspondencia:

– un campo de **struct** IDL se hará corresponder con su rango en la descripción de **structure** MHEG-SIR; su nombre no será conservado;

– un valor de rótulo de **union** IDL se hará corresponder con su rango en la descripción de **union** MHEG-SIR; su nombre y valor no serán conservados de otra forma;

– un **array** IDL multidimensional se hará corresponder con un **array** MHEG-SIR cuyo tipo de elemento es **array**.

### 14.7.4 Tipo **any**

Un tipo **any** IDL se hará corresponder con **data identifier** MHEG-SIR a condición de que el tipo **any** se utilice con una clave asociada para determinar el tipo efectivo:

**struct { Key the_key; any value }**

donde **Key** es un tipo **string**, **numeric** o **enum**, cuyo valor determina completamente el tipo del campo **value**.

El mencionado tipo IDL se hará corresponder con una **structure** MHEG-SIR de dos elementos:

– un **unsigned short** que representa un TID válido dentro del guión, para la correspondencia de la clave;

– un **data identifier** que representa una variable del tipo identificado por el primer elemento y que contiene el valor.

No se garantiza que cualquier otro uso del tipo **any** conserve su semántica después de que se haya hecho corresponder con MHEG-SIR.

### 14.7.5 Restricciones a los tipos

Si dos tipos construidos IDL tienen la misma estructura, se harán corresponder con un solo tipo MHEG-SIR.

## 14.8 Constantes IDL

Las constantes IDL se harán corresponder con una **ConstantDeclaration** MHEG-SIR declarada como un componente del componente **constant-declarations** del **InterchangedScript**. Una declaración de constante tendrá un alcance global en el guión intercambiado.

# 15 La MHEG-3 API

Esta cláusula especifica la sintaxis y la semántica de la MHEG-3 API.

Los guiones intercambiados utilizarán la MHEG-3 API de acuerdo con la sintaxis de interfaz IDL definida en esta cláusula y en el anexo F.

Los intérpretes de guiones MHEG-SIR proporcionarán la MHEG-3 API de acuerdo con la sintaxis de interfaz IDL definida en esta cláusula y en el anexo F, con la semántica definida en esta cláusula. La invocación de las operación tendrá el efecto especificado en esta cláusula.

Todas las funciones predefinidas MHEG-SIR que correspondan con operaciones MHEG-3 API serán **synchronous**.

La definición MHEG-3 API consiste en un módulo IDL único denominado **MHEG_3**. Este módulo define tipos predefinidos, así como interfaces para tres excepciones y cuatro objetos; no hay relación de herencia entre los cuatro objetos.

## 15.1 Objeto ScriptInterpreter

El objeto **ScriptInterpreter** representa el intérprete de guión. Debe ser único. Se utiliza como una fábrica de objetos **MhScript**.

Para invocar operaciones sobre el objeto **ScriptInterpreter**, los guiones intercambiados utilizarán "null" como el valor del parámetro implícito de referencia de objeto.

### 15.1.1 Operación kill (eliminar)

**Sinopsis**

Interfaz: **ScriptInterpreter**

Operación: **kill**

Resultado: **void**

**Descripción**

La operación **kill** se utiliza para eliminar el objeto **ScriptInterpreter** y terminar el proceso de intérprete de guión.

Cuando se invoca la operación, el proceso principal invocará una operación **destroy** sobre todos los objetos **MhScript**, después de lo cual terminará el proceso de intérprete de guión.

A diferencia de las otras operaciones MHEG-3 API, esta operación no es una función predefinida MHEG-SIR. En consecuencia, no estará disponible para uso por guiones intercambiados MHEG-SIR.

### 15.1.2 Operación **prepare** (preparar)

#### Sinopsis

| | | |
|---|---|---|
| Interfaz: | **ScriptInterpreter** | |
| Operación: | **prepare** | |
| Resultado: | **MhScript** | |
| En: | **ContentReference** | **content_reference** |
| Excepción: | **InvalidParameter** | |
| Excepción: | **InvalidScript** | |
| Excepción: | **OperationFailed** | |

#### Descripción

La operación **prepare** se utiliza para crear un objeto **MhScript** a partir de un guión intercambiado y pedir al intérprete de guión que inicialice ese guión mh.

El parámetro **content_reference** especifica la ubicación del guión intercambiado. Consiste en dos cadenas: un identificador público y un identificador de sistema. Si una de estas cadenas tiene el valor nulo, no se tendrá en cuenta. Por lo menos uno de los valores de las dos cadenas tiene que ser diferente de nulo.

Cuando se invoca la operación, el proceso principal realizará las operaciones de inicialización del guión mh como se especifica en 9.5.2. Una vez efectuado esto, el estado del guión mh conmutará a **available** (disponible).

El resultado de la operación será una referencia de objeto al **Mhscript** creado.

Se señalará la excepción **InvalidParameter** si el parámetro **content_reference** no permite el acceso a un guión intercambiado. En este caso, el miembro **rank** será 1.

Se señalará la excepción **InvalidScript** si se detecta un enunciado ilegal (*illegal statement*) durante el análisis (*parsing*) del guión intercambiado. En este caso, el miembro **the_entity** representará el tipo de la primera entidad en la parte declaraciones en la que se detectó un error, mientras que el miembro **identifier** representará el identificador de esta entidad de la manera siguiente:

– un TID para una declaración de tipo;
– un DID para una declaración de constante o una declaración de variable;
– un FID para una declaración de servicio o una declaración de rutina;
– un MID para una declaración de excepción o una declaración de manejador;
– un PID para una declaración de lote.

La excepción **OperationFailed** se señalará si las operaciones de inicialización de guión mh no pueden concluirse aunque no se haya detectado ningún error en la sintaxis del guión intercambiado.

Cuando se señala una excepción, no se creará el objeto **MhScript**, y el estado del guión mh deberá seguir siendo **not available** (no disponible).

## 15.2 Objeto **MhScript**

El objeto **MhScript** representa un guión mh disponible. Se utiliza como una fábrica de objetos **RtScript**.

### 15.2.1 Operación destroy (destruir)

**Sinopsis**

Interfaz: **MhScript**

Operación: **destroy**

Resultado: **void**

**Descripción**

La operación **destroy** se utiliza para eliminar el objeto **MhScript** y destruir el correspondiente guión mh.

Cuando se invoca la operación, el proceso principal seguirá los pasos siguientes en el orden especificado:

– pone el guión mh de destino en el estado **not available** (no disponible);

– invoca una operación **delete** (suprimir) sobre todos los objetos **RtScript** existentes que hayan sido creados por este guión mh;

– ejecuta el procedimiento **package unload** (descargar lote) para todos los lotes;

– libera todas las áreas de memoria de guión mh asociadas al guión mh.

### 15.2.2 Operación **new** (nueva)

**Sinopsis**

Interfaz: **MhScript**

Operación: **new**

Resultado: **Rtscript**

Excepción: **OperationFailed**

**Descripción**

La operación **new** se utiliza para crear un objeto **RtScript** a partir del guión mh y pedir al intérprete de guión que inicialice ese guión rt.

Cuando se invoca la operación, el proceso principal realizará las operaciones de inicialización de guión rt como se especifica en 9.5.3. Tras una inicialización exitosa, el estado del guión rt será **ready** (listo).

El resultado de la operación será una referencia de objeto al **RtScript** creado.

Se señalará la excepción **OperationFailed** si las operaciones de inicialización de guión rt no pueden concluirse. En este caso, no se creará el objeto **RtScript**, y el estado del guión rt seguirá siendo **not ready** (no listo).

## 15.3 Objeto **RtScript**

El objeto **RtScript** representa un guión rt cuyo estado es **ready** (listo), **running** (en funcionamiento) o **erroneous** (erróneo). Se utiliza como una fábrica de objetos **RoutineInvocation**.

### 15.3.1 Operación **delete** (suprimir)

#### Sinopsis

Interfaz: **RtScript**

Operación: **delete**

Resultado: **void**

#### Descripción

La operación **delete** se utiliza para eliminar el objeto **RtScript** y destruir el correspondiente guión rt.

Cuando se invoca la operación, el proceso principal seguirá los pasos siguientes en el orden especificado:

– pone guión rt de destino en el estado **not ready** (no listo);

– invoca una operación **close** (cerrar) sobre todos los objetos **RoutineInvocation** que han sido creados por el guión rt;

– termina todas las unidades de procesamiento;

– libera todas las áreas de memoria de guión rt asociadas al guión rt.

### 15.3.2 Operación **setPriority** (fijar prioridad)

#### Sinopsis

Interfaz: **RtScript**

Operación: **setPriority**

Resultado: **void**

En: **unsigned short** **priority**

#### Descripción

La operación **setPriority** se utiliza para modificar la prioridad de calendarización (*scheduling*) asociada con el guión rt.

El parámetro **priority** especifica el nuevo valor de prioridad.

Cuando se invoca la operación, el proceso principal puede modificar consiguientemente la política de calendarización. El efecto preciso de esta operación no se especifica en esta Recomendación. Es posible que esta operación no produzca ningún efecto, lo que dependerá de la implementación. Sin embargo, si un guión rt tiene un valor de prioridad más bajo que el de otro guión rt, a la unidad de ejecución del primer guión rt no se le dará más tiempo de la unidad central de proceso (CPU) del computador que a la unidad de ejecución del segundo.

### 15.3.3 Operación **getPriority** (obtener prioridad)

#### Sinopsis

Interfaz: **RtScript**

Operación: **getPriority**

Resultado: **unsigned short**

#### Descripción

La operación **getPriority** se utiliza para extraer el valor actual de la prioridad de calendarización asociada con el guión rt. Si no se ha fijado explícitamente una prioridad a este guión rt, se utilizará el valor por defecto especificado por el intérprete de guión.

### 15.3.4 Operación **setData** (fijar datos)

**Sinopsis**

| | | |
|---|---|---|
| Interfaz: | **RtScript** | |
| Operación: | **setData** | |
| Resultado: | **void** | |
| En: | **DID** | **variable_id** |
| En: | **any** | **variable_value** |
| Excepción: | **InvalidParameter** | |
| Excepción: | **OperationFailed** | |

**Descripción**

La operación **setData** se utiliza para asignar un valor a una variable global o a una variable dinámica del guión rt.

El parámetro **variable_id** especifica el identificador de datos de los datos que habrán de ser modificados.

El parámetro **variable_value** especifica el valor que habrá de asignarse a la variable. El tipo del parámetro real está determinado por el tipo de la variable.

Cuando se invoca la operación, el proceso principal pedirá a la unidad de ejecución de guión rt que asigne la variable de destino al valor proporcionado.

Se señalará la excepción **InvalidParameter**:

– si el parámetro **variable_id** hace referencia a una constante, a una variable local, o a una constante o variable inexistente. En este caso, el miembro **rank** (rango) será 1;

– si el parámetro **variable_value** no es de un tipo IDL que concuerda con el tipo MHEG-SIR de la variable de destino. En este caso el miembro **rank** es 2.

Se señalará la excepción **OperationFailed** si el estado del guión rt es **running** o **erroneous**.

### 15.3.5 Operación **getData** (obtener datos)

**Sinopsis**

| | | |
|---|---|---|
| Interfaz: | **RtScript** | |
| Operación: | **getData** | |
| Resultado: | **any** | |
| En: | **DID** | **data_id** |
| Excepción: | **InvalidParameter** | |
| Excepción: | **OperationFailed** | |

**Descripción**

La operación **getData** se utiliza para extraer el valor actual de una constante o variable.

El parámetro **data_id** especifica el identificador (de datos) de los datos a que se habrá de ganar acceso.

Cuando se invoca la operación, la unidad de ejecución de guión rt retornará el valor actual de la constante o variable.

El resultado de la operación será el valor solicitado y será de un tipo IDL que concuerda con el tipo MHEG-SIR de la constante o variable de destino.

Se señalará la excepción **InvalidParameter** si el parámetro **data_id** hace referencia a una constante o variable inexistente. En este caso, el miembro **rank** será 1.

Se señalará la excepción **OperationFailed** si el estado del guión rt es **running** o **erroneous**.

### 15.3.6 Operación allocate (atribuir)

#### Sinopsis

| | | |
|---|---|---|
| Interfaz: | **RtScript** | |
| Operación: | **allocate** | |
| Resultado: | **DID** | |
| En: | **TID** | **variable_type_id** |
| Excepción: | **InvalidParameter** | |
| Excepción: | **OperationFailed** | |

#### Descripción

La operación **allocate** se utiliza para crear una variable dinámica de un tipo dado dentro del guión rt.

El parámetro **variable_type_id** especifica el identificador de tipo MHEG-SIR de la variable de destino, tal como está declarado en el guión rt.

Cuando se invoca la operación, la unidad de ejecución de guión rt funcionará como si estuviera ejecutando una instrucción **ALLOC** con **variable_type_id** como operando, es decir, reservará un espacio apropiado en la región del montón (*heap*) de la memoria, generará un nuevo DID, y lo retornará.

El resultado de la operación será el identificador de datos de la nueva variable dinámica.

Se señalará la excepción **InvalidParameter** si el valor del parámetro **variable_type_id** no es ni un tipo predefinido, ni un tipo declarado dentro del guión rt. En este caso, el miembro **rank** será 1.

Se señalará la excepción **OperationFailed** dondequiera que la instrucción **ALLOC** señalaría un error. En este caso, no se atribuirá la variable y no se modificará el registro de error.

### 15.3.7 Operación free (liberar)

#### Sinopsis

| | | |
|---|---|---|
| Interfaz: | **RtScript** | |
| Operación: | **free** | |
| Resultado: | **void** | |
| En: | **DID** | **variable_id** |
| Excepción: | **InvalidParameter** | |

#### Descripción

La operación **free** se utiliza para liberar una variable dinámica del guión rt.

El parámetro **variable_id** especifica el identificador de datos de la variable que se va a liberar.

Cuando se invoca la operación, la unidad de ejecución de guión rt funcionará como si estuviera ejecutando una instrucción **FREE** con **variable_id** como operando, es decir, liberará la variable dinámica e invalidará su identificador.

Se señalará la excepción **InvalidParameter** si el parámetro **variable_id** no hace referencia a una variable dinámica existente anteriormente atribuida a través de la MHEG-3 API. En este caso, el miembro **rank** será 1.

NOTA – Un intérprete de guión puede aplicar una política de atribución de identificadores de datos que permita distinguir fácilmente entre variables atribuidas a través de la MHEG-3 API y variables atribuidas mediante una instrucción, por ejemplo, según el rango a que pertenece su identificador de datos.

### 15.3.8 Operación stop (detener)

#### Sinopsis

Interfaz: **RtScript**

Operación: **stop**

Resultado: **void**

Excepción: **OperationFailed**

#### Descripción

La operación **stop** se utiliza para hacer que el guión rt vuelva al estado listo.

Cuando se invoca la operación, el intérprete de guión pedirá a la unidad de ejecución de guión rt que se detenga, que vacía la pila de llamadas, la cola de mensajes y la pila de parámetros, y que reponga (reset) todos los registros. Seguidamente, pondrá el guión rt en el estado **ready**. A diferencia de la operación **reInit** (reiniciar). Las variables globales y las dinámicas conservan sus valores.

Se señalará la excepción **OperationFailed** si la operación no pudo realizarse con éxito, por ejemplo si se han corrompido áreas de la memoria del guión rt como consecuencia de un error en la ejecución.

### 15.3.9 Operación reInit (reiniciar)

#### Sinopsis

Interfaz: **RtScript**

Operación: **reInit**

Resultado: **void**

Excepción: **OperationFailed**

#### Descripción

La operación **reInit** se utiliza para hacer que el guión rt vuelva a su estado inicial, es decir, al estado en que se encontraría inmediatamente después de una inicialización.

Cuando se invoca la operación, el intérprete de guión:

– terminará la unidad de ejecución de guión rt;

– liberará todas las variables dinámicas;

– fijará las variables globales a sus valores iniciales (que figuran en la tabla de definiciones de variables globales del guión mh);

– vaciará la pila de parámetros, la cola de mensajes y la pila de llamadas, liberando las tablas de variables locales;

– repondrá todos los registros;

– finalmente, pondrá el guión rt en el estado **ready**.

Se señalará la excepción **OperationFailed** si la operación no pudo realizarse con éxito, por ejemplo si se han corrompido áreas de la memoria del guión rt como consecuencia de un error en la ejecución.

### 15.3.10 Operación **getRtScriptStatus** (obtener estado de guión rt)

#### Sinopsis

| | |
|---|---|
| Interfaz: | **RtScript** |
| Operación: | **getRtScriptStatus** |
| Resultado: | **RtScriptStatus** |

#### Descripción

La operación **getRtScriptStatus** se utiliza para obtener el estado actual del guión rt.

El resultado de la operación será uno de los siguientes: **READY**, **RUNNING** o **ERRONEOUS**.

### 15.3.11 Operación **open** (abrir)

#### Sinopsis

| | | |
|---|---|---|
| Interfaz: | **RtScript** | |
| Operación: | **open** | |
| Resultado: | **RoutineInvocation** | |
| En: | **FID** | **routine_id** |
| Excepción: | **InvalidParameter** | |

#### Descripción

La operación **open** se utiliza para crear un objeto **RoutineInvocation** a partir del guión rt.

El parámetro **routine_id** especifica el identificador de función con el que está asociado el nuevo objeto **RoutineInvocation**.

El intérprete de guión puede optar por aplicar una de la políticas siguientes:

a) extraer la firma de la rutina de destino cuando se invoca la operación **open**, con el fin de comprobar "al vuelo", es decir, tan pronto como se invoca una operación **setParameter**, la validez de los parámetros pasados;

b) comprobar la validez de los parámetros solamente al invocar la operación **run**.

El resultado de la operación será una referencia de objeto a la **RoutineInvocation** creada.

Se señalará la excepción **InvalidParameter** si el parámetro **routine_id** no identifica una rutina válida del guión rt. En este caso, el miembro **rank** será 1.

## 15.4 Objeto **RoutineInvocation**

El objeto **RoutineInvocation** representa un contexto de invocación de una rutina. Este contexto de invocación se utiliza para pasar parámetros a una determinada rutina del guión rt, y pedir la ejecución de dicha rutina.

### 15.4.1 Operación **close** (cerrar)

#### Sinopsis

| | |
|---|---|
| Interfaz: | **RoutineInvocation** |
| Operación: | **close** |
| Resultado: | **void** |

### Descripción

La operación **close** se utiliza para eliminar el objeto **RoutineInvocation** y cerrar el correspondiente contexto de invocación de rutina.

### 15.4.2 Atributo routine_id (de lectura solamente)

### Sinopsis

| | | |
|---|---|---|
| Interfaz: | **RoutineInvocation** | |
| Atributo: | **FID** | **routine_id** |

### Descripción

El atributo **routine_id** es un atributo de lectura solamente que se fija cuando la operación **open** crea el objeto **RoutineInvocation**. Su valor será el identificador de función de la rutina direccionada por el objeto **RoutineInvocation**.

Los guiones intercambiados ganarán acceso al valor de este atributo utilizando la función predefinida **get_RoutineId**.

### 15.4.3 Operación setParameter (fijar parámetro)

### Sinopsis

| | | |
|---|---|---|
| Interfaz: | **RoutineInvocation** | |
| Operación: | **setParameter** | |
| Resultado: | **void** | |
| En: | **unsigned short** | **rank** |
| En: | **TID** | **parameter_type_id** |
| En: | **any** | **parameter_value** |
| Excepción: | **InvalidParameter** | |

### Descripción

La operación **setParameter** se utiliza para pasar el valor del parámetro de una rutina que habrá de ser utilizado por la próxima operación **run**.

El parámetro **rank** especifica, en la descripción de la firma de la rutina, el rango del parámetro pasado; el rango 0 indica el primer parámetro. Este parámetro corresponde por tanto al índice del parámetro en la tabla de variables locales de la rutina.

El parámetro **parameter_type_id** especifica el identificador de tipo MHEG-SIR del parámetro pasado, declarado en el guión rt.

El parámetro **parameter_value** especifica el valor del parámetro pasado. El tipo del valor lo determina el parámetro **parameter_type_id**.

Cuando se invoca la operación, el intérprete de guión deberá almacenar en memoria tampón el parámetro que será utilizado por la próxima operación **run** en esta rutina. Si el intérprete de guión opta por la política a) de 15.3.11, comprobará la validez de los parámetros **parameter_type_id** y **parameter_value** con respecto a la firma de la rutina.

Si el intérprete de guión opta por la política a) definida en 15.3.11, se señalará la excepción **InvalidParameter**:

– si el parámetro **rank** de la operación excede el número del último parámetro de la rutina. En este caso, el miembro **rank** de la excepción será 1;

– si el parámetro **parameter_type_id** no corresponde con el tipo de parámetro en la firma de la rutina. En este caso, el miembro **rank** de la excepción será 2;

– si el parámetro **parameter_value** no es de un tipo apropiado, esto es, un tipo IDL que concuerda con el tipo descrito por el parámetro **parameter_type_id**, cuando el modo de paso de parámetro es por **value**, y un tipo DID, cuando el modo de paso de parámetro es por **reference**. En este caso, el miembro **rank** de la excepción será 3;

– cuando el modo de paso de parámetro es por **referencia**, si el parámetro **parameter_value** es un DID que no identifica una variable global o dinámica existente cuyo tipo concuerda con el tipo de parámetro definido por la firma de la rutina. En este caso, el miembro **rank** de la excepción será 3.

### 15.4.4 Operación **getPrototype** (obtener prototipo)

#### Sinopsis

Interfaz: **RoutineInvocation**

Operación: **getPrototype**

Resultado: **Prototype**

#### Descripción

La operación **getPrototype** se utiliza para extraer la firma de la rutina.

Cuando se invoca la operación, el intérprete de guión retornará la firma de la rutina.

El resultado de la operación será una descripción de la firma de la rutina:

a) el campo **return_value_type** se fijará a RT[routine_id].TID;

b) el n-ésimo ítem del campo firma corresponderá a RT[routine_id].sig[n]:

   1) el campo **passing_mode** se fijará a **BY_VALUE** o **BY_REFERENCE** respectivamente cuando **RT[routine_id].sig[n].mod**, es "value", "reference", respectivamente;

   2) el campo **parameter_type_id** se fijará a **RT[routine_id].sig[n].TID**.

### 15.4.5 Operación **run** (ejecutar)

#### Sinopsis

Interfaz: **RoutineInvocation**

Operación: **run**

Resultado: **void**

Excepción: **OperationFailed**

#### Descripción

La operación **run** se utiliza para pedir la ejecución de la rutina, con los valores de parámetros proporcionados mediante la operación **setParameter**.

Cuando se invoca la operación, el proceso principal:

– creará una mensaje cuyo identificador es el índice de la rutina (o sea, el valor del atributo **routine_id**) y cuyos parámetros son los parámetros fijados por las anteriores operaciones **setParameter**.

– insertará este mensaje en la cola de mensajes del guión rt de destino;

– si el estado actual de guión rt es **ready**, lo conmutará a **running**.

Si el intérprete de guión opta por la política b) definida en 15.3.11, comprobará la validez, con respecto a la firma de la rutina, de todos los identificadores de tipo y de todos los valores de los parámetros previamente proporcionados mediante la operación **setParameter**.

Se señalará la excepción **OperationFailed** si cualquiera de los parámetros proporcionados no corresponde con la firma de la rutina.

### 15.4.6 Operación reset (reponer)

#### Sinopsis

Interfaz: **RoutineInvocation**

Operación: **reset**

Resultado: **void**

#### Descripción

La operación **reset** se utiliza para liberar el contexto de invocación de rutina con el fin de preparar una nueva invocación.

Cuando se invoca la operación, se liberaran los parámetros previamente almacenados en memoria tampón como resultado de la operación **setParameter**.

NOTA – La utilización de esta operación después de cada operación **run** evita el riesgo de colisión. Por otra parte, si no se utiliza esta operación se puede repetir la misma invocación sin tener que volver a dar los parámetros.

### 15.4.7 Operación getInvocationStatus (obtener estado de invocación)

#### Sinopsis

Interfaz: **RoutineInvocation**

Operación: **getInvocationStatus**

Resultado: **InvocationStatus**

#### Descripción

La operación **getInvocationStatus** se utiliza para extraer el estado actual de la invocación de rutina.

El resultado de la operación será uno de los valores siguientes:

– **NOT_STARTED**: no se ha invocado ninguna operación **run** desde la creación del objeto, o desde la última operación **reset**;

– **PROCESSING**: se ha invocado una operación **run**, pero la unidad de ejecución de guión rt no ha concluido la ejecución de la rutina (o bien la petición está en la cola de mensajes, o la rutina se está ejecutando en este momento);

– **TERMINATED**: la ejecución de rutina ocasionada por la última operación **run** invocada ha sido concluida por la unidad de ejecución de guión rt;

– **ABORTED**: la ejecución de rutina ocasionada por la última operación **run** invocada ha dado por resultado un error de ejecución de instrucción.

ANEXO A

## Especificación ASN.1 de guiones intercambiados

Este anexo especifica la notación ASN.1, según la Rec. UIT-T X.680 | ISO/CEI 8824-1 [1], para la sintaxis del componente "script data" (datos de guión) de la clase "guión" de MHEG.

Los guiones intercambiados tendrán la sintaxis definida por el módulo **ISOMHEG_sir** en ASN.1.

```
-- Module: MHEG-SIR (sir)--
--
-- Copyright statement:
-- --------------------
--      (c) ITU, 1996.
--          Permission to copy in any form is granted for use with conforming to
--          MHEG-3 engines and applications as defined by this Recommendation
--          provided this notice is included in all copies.

ISOMHEG-sir {joint-iso-itu-t (2) mheg (19) version (1) script-interchange-representation (11)}
DEFINITIONS IMPLICIT TAGS ::= BEGIN

EXPORTS    InterchangedScript;

InterchangedScript          ::=     SEQUENCE
{
        type-declarations                   SEQUENCE (SIZE (1.. max-nb-declared-types)) OF
                                    TypeDeclaration OPTIONAL,
        constant-declarations               [0] SEQUENCE (SIZE (1 .. max-nb-constants)) OF
                                    ConstantDeclaration OPTIONAL,
        global-variable-declarations        [1] SEQUENCE (SIZE (1 .. max-nb-global-variables)) OF
                                    VariableDeclaration OPTIONAL,
        external-package-declarations       [2] SEQUENCE (SIZE (1 .. max-nb-packages)) OF
                                    PackageDeclaration OPTIONAL,
        handler-declarations                [3] SEQUENCE (SIZE (1 .. max-nb-messages)) OF
                                    HandlerDeclaration OPTIONAL,
        routine-declarations                [4] SEQUENCE (SIZE (1 .. max-nb-routines)) OF
                                    RoutineDeclaration OPTIONAL
}

TypeDeclaration     ::=     SEQUENCE
{
        identifier                  [0] TypeIdentifier OPTIONAL,
        description                 TypeDescription
}

TypeDescription     ::=     CHOICE
{
        string-description              [1] INTEGER (0..max-size-string) OPTIONAL,
        sequence-description            [2] SequenceDescription,
        array-description               [3] ArrayDescription,
        structure-description           [4] StructureDescription,
        union-description               [5] UnionDescription
}

SequenceDescription         ::=     SEQUENCE
{
        bound                       INTEGER (0 .. max-size-sequence),
        element-type                TypeIdentifier
}
```

```
ArrayDescription    ::=    SEQUENCE
{
        size                        INTEGER (1 .. max-size-array),
        element-type                TypeIdentifier
}

UnionDescription    ::=    SEQUENCE (SIZE (1 .. max-size-union)) OF TypeIdentifier

StructureDescription        ::=    SEQUENCE (SIZE (1 .. max-size-structure)) OF TypeIdentifier

ConstantDeclaration         ::=    SEQUENCE
{
        identifier                  [0] DataIdentifier OPTIONAL,
        type                        TypeIdentifier ALL EXCEPT 0,
        value                       ConstantValue
}
ConstantValue               ::=    CHOICE
{
        octet                           [1] OctetValue,
        short                           [2] ShortValue,
        long                            [3] LongValue,
        unsigned-short                  [4] UnsignedShortValue,
        unsigned-long                   [5] UnsignedLongValue,
        float                           [6] FloatValue,
        double                          [7] DoubleValue,
        boolean                         [8] BooleanValue,
        character                       [9] CharacterValue,
        data-identifier                 [10] DataIdentifier (0..<max-nb-constants),
        string                          [11] StringValue,
        sequence                        [12] SequenceValue,
        array                           [13] ArrayValue,
        structure                       [14] StructureValue,
        union                           [15] UnionValue
}

SequenceValue               ::=    SEQUENCE (SIZE (0 .. max-size-sequence)) OF ConstantValue

ArrayValue          ::=    SEQUENCE (SIZE (1 .. max-size-array)) OF ConstantValue

UnionValue          ::=    SEQUENCE
{
        tag                         INTEGER (0 .. < max-size-union),
        value                       ConstantValue
}

StructureValue              ::=    SEQUENCE (SIZE (1 .. max-size-structure)) OF ConstantValue

VariableDeclaration         ::=    SEQUENCE
{
        identifier                      [0] DataIdentifier OPTIONAL,
        type                            TypeIdentifier,
        initial-value                   ConstantReference OPTIONAL
}

PackageDeclaration          ::=    SEQUENCE
{
        identifier                  [0] PackageIdentifier OPTIONAL,
        name                        VisibleString OPTIONAL,
        services                    SEQUENCE (SIZE (0 .. max-nb-services)) OF
                                            ServiceDescription,
        exceptions                  SEQUENCE (SIZE (0 .. max-nb-exceptions)) OF
```

```
}

ServiceDescription ::=     SEQUENCE
{
        identifier                      [0] FunctionIdentifier OPTIONAL,
        name                            VisibleString OPTIONAL,
        calling-mode                    ENUMERATED {synchronous (0), asynchronous (1)}
                                            DEFAULT synchronous,
        return-value-type               TypeIdentifier DEFAULT 0,
        parameters-description          SEQUENCE OF ServiceParameterDescription OPTIONAL
}

ServiceParameterDescription     ::=     SEQUENCE
{
        passing-mode            ENUMERATED {in (1), out (2), inout (3)} DEFAULT in,
        type                    TypeIdentifier ALL EXCEPT 0
}

ExceptionDescription        ::=     SEQUENCE
{
        identifier                          [0] MessageIdentifier OPTIONAL,
        name                                VisibleString OPTIONAL,
        parameters-description              SEQUENCE OF TypeIdentifier OPTIONAL
}

HandlerDeclaration          ::=     SEQUENCE
{
        message-identifier              MessageIdentifier,
        function-identifier             FunctionIdentifier
}

RoutineDeclaration ::=     SEQUENCE
{
        routine-description             RoutineDescription,
        program-code                    OCTET STRING
}

RoutineDescription ::=     SEQUENCE
{
        identifier                          [0] FunctionIdentifier OPTIONAL,
        return-value-type                   TypeIdentifier DEFAULT 0,
        parameters-description              [1] SEQUENCE OF RoutineParameterDescription OPTIONAL,
        local-variable-table                [2] SEQUENCE (SIZE (0 .. max-nb-local-variables)) OF
                                                VariableDeclaration OPTIONAL
}

RoutineParameterDescription     ::= SEQUENCE
{
        passing-mode                    ENUMERATED {value (1), reference (3)} DEFAULT value,
        type                            TypeIdentifier ALL EXCEPT 0
}

ConstantReference ::=     CHOICE
{
        identifier              [16] DataIdentifier,
        value                   ConstantValue
}

max-size-sequence                   INTEGER   ::=   65535
max-size-string                     INTEGER   ::=   65535
```

```
max-size-array                  INTEGER  ::=  65536
max-size-union                  INTEGER  ::=  256
max-size-structure              INTEGER  ::=  256
max-nb-global-variables         INTEGER  ::=  28672
max-nb-constants                INTEGER  ::=  4096
max-nb-local-variables          INTEGER  ::=  256
max-nb-dynamic-variables        INTEGER  ::=  32512
max-nb-data                     INTEGER  ::=  65536
    -- max-nb-constants+max-nb-global-variables+max-nb-local-variables+max-nb-dynamic-
    -- variables
max-nb-packages                 INTEGER  ::=  192
max-nb-services                 INTEGER  ::=  256
max-nb-routines                 INTEGER  ::=  4096
max-nb-predef-functions         INTEGER  ::=  12288
max-nb-functions                INTEGER  ::=  65536
    -- max-nb-packagesxmax-nb-services+max-nb-predef-functions+max-nb-routines
max-nb-exceptions               INTEGER  ::=  256
max-nb-predef-messages          INTEGER  ::=  16384
max-nb-messages                 INTEGER  ::=  65536
    -- max-nb-packagesxmax-nb-exceptions+max-nb-predef-messages
max-nb-declared-types           INTEGER  ::=  16384
max-nb-predef-types             INTEGER  ::=  16384
max-nb-types                    INTEGER  ::=  32768
    -- max-nb-predef-types + max-nb-declared-types

OctetValue                      ::=  OCTET STRING (SIZE (1))
ShortValue                      ::=  INTEGER (–32768 .. 32767)
LongValue                       ::=  INTEGER (–2147483648 .. 2147483647)
UnsignedShortValue              ::=  INTEGER (0 .. 65535)
UnsignedLongValue               ::=  INTEGER (0 .. 4294967295)
FloatValue                      ::=  REAL
DoubleValue                     ::=  REAL
BooleanValue                    ::=  BOOLEAN
CharacterValue                  ::=  BMPString (SIZE (1))
StringValue                     ::=  BMPString (SIZE (0.. max-size-string))

TypeIdentifier                  ::=  INTEGER (0 .. < max-nb-types)
DataIdentifier                  ::=  INTEGER (0 .. < max-nb-data)
FunctionIdentifier              ::=  INTEGER (0 .. < max-nb-functions)
MessageIdentifier               ::=  INTEGER (0 .. < max-nb-messages)
PackageIdentifier               ::=  INTEGER (0 .. < max-nb-packages)

END
```

ANEXO B

## Representación codificada de guiones intercambiados

### B.1 Codificación para guiones intercambiados

Los guiones intercambiados se codificarán de acuerdo con las reglas de codificación distinguida (DER, *distinguished encoding rules*) especificadas en la Rec. UIT-T X.690 | ISO/CEI 8825-1 [2].

NOTA – Esto tiene por finalidad hacer que la tarea de decodificación del motor MHEG-3 sea lo más eficiente posible al suprimirse todas las opciones de codificación ASN.1 que pudieran retardarla o complicarla.

## B.2 Codificación para el código de programa

El valor del componente **program-code** del tipo **RoutineDeclaration** definido por **ISOMHEG-sir** (véase el anexo A) se codificará de acuerdo con las reglas definidas en esta subcláusula.

La secuencia de instrucciones que constituyen el código de programa de una rutina se codificará como una secuencia de octetos. El orden de codificación será el mismo que el orden en que se desea que las instrucciones sean ejecutadas.

Cada instrucción se codificará utilizando un octeto para el código de operación (brevemente código op), seguido de cero a tres octetos para los operandos, lo que dependerá del código op.

### B.2.1 Códigos op de instrucción

Los códigos op se codificarán utilizando la cadena de bits definida en el cuadro B.1.

### B.2.2 Operandos de instrucciones

De acuerdo con el código op de la instrucción, los operandos tendrán la longitud y codificación definidas en el cuadro B.1. Todos los operandos constituido por más de un octeto se codificarán por el orden de su peso, es decir, el octeto más significativo se codificará primero.

#### B.2.2.1 Operandos identificador de datos

Los operandos identificador de datos (DID) se codificarán en dos octetos de la manera siguiente:

– si el bit 16 es "1" y los bits 15 a 9 son "0", el DID hará referencia a una variable local, donde los bits 8 a 1 representan el índice de variable local (de 0 a 255);

– en los demás casos, si el bit 16 es "1", el DID hará referencia a una variable dinámica, donde los bits 15 a 1 representan el índice de la variable dinámica (de 0 a 32511) incrementado en 256;

– si los bits 16 a 13 son "0000", el DID hará referencia a una constante, donde los bits 12 a 1 representan el índice de la constante (de 0 a 4095);

– en todos los demás casos, el DID hará referencia a una variable global, donde los bits 15 a 1 representan el índice de la variable global (de 0 a 28671) incrementado en 4096.

#### B.2.2.2 Operandos identificador de función

Los operandos identificador de función (FID) se codificarán en dos octetos de la manera siguiente:

– si los bits 16 a 13 son "0000", el FID hará referencia a una rutina, donde los bits 12 a 1 representan el índice de la rutina (de 0 a 4095);

– en los demás casos, si los bits 16 y 15 son "00", el FID hará referencia a una función predefinida, donde los bits 14 a 1 representan el índice de la función predefinida (de 0 a 12287) incrementado en 4096;

– en todos los demás casos, el FID hará referencia a un servicio, donde los bits 16 a 9 representan el identificador de lote (de 0 a 191) incrementado en 64, y donde los bits 8 a 1 representan el índice del servicio (de 0 a 255) en este lote.

#### B.2.2.3 Operandos numéricos diversos

Los operandos que tienen un "desplazamiento" ("offset") de un octeto se codificarán en la notación de complemento de uno, en un octeto; el bit 8 representa el sentido de la progresión, los bits 7 a 1 representan el número de unidades del desplazamiento en el sentido indicado.

Los operandos que tienen un "desplazamiento" de dos octetos se codificarán en la notación de complemento de uno, en dos octetos; el bit 16 representa el sentido de la progresión, los bits 15 a 1 representan el número de unidades del desplazamiento en el sentido indicado.

Los operandos "valor" ("*value*") se codificarán en la notación de complemento de dos, en dos octetos, y se interpretarán como valores enteros con signo.

Los operandos "índice" ("*index*") se codificarán en un octeto, y se interpretarán como valores enteros sin signo.

**Cuadro B.1/T.173 – Codificación de instrucciones MHEG-SIR**

| Nemónicos de instrucción | Código op (binario) | Código op (hexad.) | Longi-tud op1 | Codificación op1 | Longi-tud op2 | Codificación op2 |
|---|---|---|---|---|---|---|
| NOP | 0000 0000 | 00 | 0 | | | |
| YIELD | 0000 0010 | 02 | 0 | | | |
| RET | 0000 0011 | 03 | 0 | | | |
| FREE | 0000 1000 | 08 | 0 | | | |
| NOT_B | 0001 0000 | 10 | 0 | | | |
| NOT_O | 0001 0001 | 11 | 0 | | | |
| NOT_W | 0001 0010 | 12 | 0 | | | |
| NOT_U | 0001 0011 | 13 | 0 | | | |
| OR_B | 0001 0100 | 14 | 0 | | | |
| OR_O | 0001 0101 | 15 | 0 | | | |
| OR_W | 0001 0110 | 16 | 0 | | | |
| OR_U | 0001 0111 | 17 | 0 | | | |
| XOR_B | 0001 1000 | 18 | 0 | | | |
| XOR_O | 0001 1001 | 19 | 0 | | | |
| XOR_W | 0001 1010 | 1A | 0 | | | |
| XOR_U | 0001 1011 | 1B | 0 | | | |
| AND_B | 0001 1100 | 1C | 0 | | | |
| AND_O | 0001 1101 | 1D | 0 | | | |
| AND_W | 0001 1110 | 1E | 0 | | | |
| AND_U | 0001 1111 | 1F | 0 | | | |
| EQR | 0010 0000 | 20 | 0 | | | |
| EQ_O | 0010 0001 | 21 | 0 | | | |
| EQ_S | 0010 0010 | 22 | 0 | | | |
| EQ_L | 0010 0011 | 23 | 0 | | | |
| EQ_W | 0010 0100 | 24 | 0 | | | |
| EQ_U | 0010 0101 | 25 | 0 | | | |
| EQ_F | 0010 0110 | 26 | 0 | | | |
| EQ_D | 0010 0111 | 27 | 0 | | | |
| EQ_B | 0010 1000 | 28 | 0 | | | |
| EQ_C | 0010 1001 | 29 | 0 | | | |
| EQ_I | 0010 1010 | 2A | 0 | | | |

**Cuadro B.1/T.173 – Codificación de instrucciones MHEG-SIR** *(continuación)*

| Nemónicos de instrucción | Código op (binario) | Código op (hexad.) | Longi-tud op1 | Codificación op1 | Longi-tud op2 | Codificación op2 |
|---|---|---|---|---|---|---|
| EQ_R | 0010 1011 | 2B | 0 | | | |
| LT_C | 0011 0000 | 30 | 0 | | | |
| LT_O | 0011 0001 | 31 | 0 | | | |
| LT_S | 0011 0010 | 32 | 0 | | | |
| LT_L | 0011 0011 | 33 | 0 | | | |
| LT_W | 0011 0100 | 34 | 0 | | | |
| LT_U | 0011 0101 | 35 | 0 | | | |
| LT_F | 0011 0110 | 36 | 0 | | | |
| LT_D | 0011 0111 | 37 | 0 | | | |
| GT_C | 0011 1000 | 38 | 0 | | | |
| GT_O | 0011 1001 | 39 | 0 | | | |
| GT_S | 0011 1010 | 3A | 0 | | | |
| GT_L | 0011 1011 | 3B | 0 | | | |
| GT_W | 0011 1100 | 3C | 0 | | | |
| GT_U | 0011 1101 | 3D | 0 | | | |
| GT_F | 0011 1110 | 3E | 0 | | | |
| GT_D | 0011 1111 | 3F | 0 | | | |
| ADD_O | 0100 0001 | 41 | 0 | | | |
| ADD_S | 0100 0010 | 42 | 0 | | | |
| ADD_L | 0100 0011 | 43 | 0 | | | |
| ADD_W | 0100 0100 | 44 | 0 | | | |
| ADD_U | 0100 0101 | 45 | 0 | | | |
| ADD_F | 0100 0110 | 46 | 0 | | | |
| ADD_D | 0100 0111 | 47 | 0 | | | |
| SUB_O | 0100 1001 | 49 | 0 | | | |
| SUB_S | 0100 1010 | 4A | 0 | | | |
| SUB_L | 0100 1011 | 4B | 0 | | | |
| SUB_W | 0100 1100 | 4C | 0 | | | |
| SUB_U | 0100 1101 | 4D | 0 | | | |
| SUB_F | 0100 1110 | 4E | 0 | | | |
| SUB_D | 0100 1111 | 4F | 0 | | | |
| MUL_O | 0101 0001 | 51 | 0 | | | |
| MUL_S | 0101 0010 | 52 | 0 | | | |
| MUL_L | 0101 0011 | 53 | 0 | | | |
| MUL_W | 0101 0100 | 54 | 0 | | | |
| MUL_U | 0101 0101 | 55 | 0 | | | |
| MUL_F | 0101 0110 | 56 | 0 | | | |
| MUL_D | 0101 0111 | 57 | 0 | | | |
| DIV_O | 0101 1001 | 59 | 0 | | | |
| DIV_S | 0101 1010 | 5A | 0 | | | |
| DIV_L | 0101 1011 | 5B | 0 | | | |

**Cuadro B.1/T.173 – Codificación de instrucciones MHEG-SIR** *(continuación)*

| Nemónicos de instrucción | Código op (binario) | Código op (hexad.) | Longi-tud op1 | Codificación op1 | Longi-tud op2 | Codificación op2 |
|---|---|---|---|---|---|---|
| DIV_W | 0101 1100 | 5C | 0 | | | |
| DIV_U | 0101 1101 | 5D | 0 | | | |
| DIV_F | 0101 1110 | 5E | 0 | | | |
| DIV_D | 0101 1111 | 5F | 0 | | | |
| NEG_S | 0110 0010 | 62 | 0 | | | |
| NEG_L | 0110 0011 | 63 | 0 | | | |
| NEG_F | 0110 0110 | 66 | 0 | | | |
| NEG_D | 0110 0111 | 67 | 0 | | | |
| REM_O | 0111 1001 | 79 | 0 | | | |
| REM_S | 0111 1010 | 7A | 0 | | | |
| REM_L | 0111 1011 | 7B | 0 | | | |
| REM_W | 0111 1100 | 7C | 0 | | | |
| REM_U | 0111 1101 | 7D | 0 | | | |
| DUP_O | 1000 0001 | 81 | 0 | | | |
| DUP_S | 1000 0010 | 82 | 0 | | | |
| DUP_L | 1000 0011 | 83 | 0 | | | |
| DUP_W | 1000 0100 | 84 | 0 | | | |
| DUP_U | 1000 0101 | 85 | 0 | | | |
| DUP_F | 1000 0110 | 86 | 0 | | | |
| DUP_D | 1000 0111 | 87 | 0 | | | |
| DUP_B | 1000 1000 | 88 | 0 | | | |
| DUP_C | 1000 1001 | 89 | 0 | | | |
| DUP_I | 1000 1010 | 8A | 0 | | | |
| DUP_R | 1000 1011 | 8B | 0 | | | |
| CVT_SW | 1001 0100 | 94 | 0 | | | |
| CVT_WS | 1001 0101 | 95 | 0 | | | |
| CVT_LU | 1001 0110 | 96 | 0 | | | |
| CVT_UL | 1001 0111 | 97 | 0 | | | |
| CVT_CW | 1001 1010 | 9A | 0 | | | |
| CVT_WC | 1001 1011 | 9B | 0 | | | |
| CVT_BS | 1010 0000 | A0 | 0 | | | |
| CVT_OS | 1010 0001 | A1 | 0 | | | |
| CVT_SL | 1010 0010 | A2 | 0 | | | |
| CVT_LF | 1010 0011 | A3 | 0 | | | |
| CVT_WL | 1010 0100 | A4 | 0 | | | |
| CVT_UF | 1010 0101 | A5 | 0 | | | |
| CVT_FD | 1010 0110 | A6 | 0 | | | |
| CVT_BO | 1010 1000 | A8 | 0 | | | |
| CVT_OW | 1010 1001 | A9 | 0 | | | |
| CVT_SU | 1010 1010 | AA | 0 | | | |
| CVT_WU | 1010 1100 | AC | 0 | | | |

**Cuadro B.1/T.173 – Codificación de instrucciones MHEG-SIR** *(fin)*

| Nemónicos de instrucción | Código op (binario) | Código op (hexad.) | Longi-tud op1 | Codificación op1 | Longi-tud op2 | Codificación op2 |
|---|---|---|---|---|---|---|
| CVT_OB | 1011 0001 | B1 | 0 | | | |
| CVT_SB | 1011 0010 | B2 | 0 | | | |
| CVT_LB | 1011 0011 | B3 | 0 | | | |
| CVT_WB | 1011 0100 | B4 | 0 | | | |
| CVT_UB | 1011 0101 | B5 | 0 | | | |
| CVT_WO | 1011 1001 | B9 | 0 | | | |
| CVT_LS | 1011 1010 | BA | 0 | | | |
| CVT_FL | 1011 1011 | BB | 0 | | | |
| CVT_UW | 1011 1100 | BC | 0 | | | |
| CVT_FU | 1011 1101 | BD | 0 | | | |
| CVT_DF | 1011 1110 | BE | 0 | | | |
| JT | 1100 0000 | C0 | 1 | despl. (con signo) | | |
| JF | 1100 0001 | C1 | 1 | despl. (con signo) | | |
| JMP | 1100 0010 | C2 | 1 | despl. (con signo) | | |
| SHIFT_O | 1100 0101 | C5 | 1 | despl. (con signo) | | |
| SHIFT_W | 1100 0110 | C6 | 1 | despl. (con signo) | | |
| SHIFT_U | 1100 0111 | C7 | 1 | despl. (con signo) | | |
| GETOR | 1100 1001 | C9 | 1 | identif. paquete | | |
| LJT | 1101 0000 | D0 | 2 | despl. (con signo) | | |
| LJF | 1101 0001 | D1 | 2 | despl. (con signo) | | |
| LJMP | 1101 0010 | D2 | 2 | despl. (con signo) | | |
| CALL | 1101 0100 | D4 | 2 | identif. función | | |
| XCALL | 1101 0110 | D6 | 2 | identif. función | | |
| PUSH | 1110 0000 | E0 | 2 | identif. datos | | |
| PUSHR | 1110 0001 | E1 | 2 | identif. datos | | |
| PUSHI | 1110 0011 | E3 | 2 | valor (con signo) | | |
| POP | 1110 0100 | E4 | 2 | identif. datos | | |
| POPR | 1110 0101 | E5 | 2 | identif. datos | | |
| POPC | 1110 0110 | E6 | 2 | identif. datos | | |
| ALLOC | 1110 1000 | E8 | 2 | identif. tipo | | |
| INC | 1110 1100 | EA | 2 | identif. datos | | |
| DEC | 1110 1101 | EB | 2 | identif. datos | | |
| GET | 1111 0000 | F0 | 2 | identif. datos | 1 | índice (sin signo) |
| GETC | 1111 0010 | F2 | 2 | identif. datos | 1 | índice (sin signo) |
| SET | 1111 0100 | F4 | 2 | identif. datos | 1 | índice (sin signo) |
| SETC | 1111 0110 | F6 | 2 | identif. datos | 1 | índice (sin signo) |

ANEXO C

# Elementos predefinidos de la MHEG-SIR

En este anexo se indican los tipos, funciones y mensajes predefinidos de la MHEG-SIR, con sus índices correspondientes.

Los tipos, funciones y mensajes predefinidos pueden ser referenciados por su identificador y utilizados en guiones intercambiados de la misma manera que se utilizarían tipos, funciones y mensajes declarados en la parte de declaraciones globales de guiones intercambiados.

## C.1 Tipos predefinidos

Los tipos predefinidos MHEG-SIR comprenden:

– tipos primitivos;

– tipos MHEG API.

### C.1.1 Tipos primitivos

Los tipos primitivos definidos por esta Recomendación se codificarán utilizando los identificadores de tipos predefinidos indicados en el cuadro C.1.

**Cuadro C.1/T.173 – Identificadores de tipo predefinido para tipos primitivos**

| Nombre de tipo | Identificador de tipo |
|---|---|
| **void** | 0 |
| **octet** | 1 |
| **short** | 2 |
| **long** | 3 |
| **unsigned short** | 4 |
| **unsigned long** | 5 |
| **float** | 6 |
| **double** | 7 |
| **boolean** | 8 |
| **character** | 9 |
| **data identifier** | 10 |
| **object reference** | 11 |

Todos los tipos que pueden expresarse en MHEG-SIR (incluidos los tipos MHEG predefinidos) pueden construirse utilizando los tipos primitivos MHEG-SIR y los siguientes constructores:

– **string** (cadena);

– **sequence** (secuencia);

– **array** (matriz);

– **structure** (estructura);

– **union** (unión).

Por convenio, el tipo **string** no acotada (el único tipo construido sin un elemento o parámetro) será un predefinido y su identificador de tipo será 12.

### C.1.2 Tipos MHEG API

Los tipos MHEG API definidos por la MHEG API se codificarán utilizando identificadores de tipos predefinidos.

NOTA – Los tipos MHEG API están previstos para ser utilizados por guiones intercambiados para expresar información que se intercambia entre el intérprete de guión y entidades MHEG.

La definición IDL de estos tipos, proporcionada por una MHEG API, se hará corresponder con descripciones de tipos MHEG-SIR utilizando las reglas generales de correspondencia IDL definidas en la cláusula 14 y las reglas específicas de codificación MHEG API definidas en E.2.

## C.2 Funciones predefinidas

Las funciones predefinidas MHEG-SIR comprenden:

– operaciones MHEG API;

– operaciones MHEG-3 API.

### C.2.1 Operaciones MHEG API

Las operaciones MHEG API definidas por la MHEG API se codificarán utilizando identificadores de funciones predefinidas.

Los identificadores de mensajes predefinidos para las operaciones MHEG API comenzarán en 1100h.

La definición IDL de estas operaciones, proporcionada por la MHEG-3 API, se hará corresponder con descripciones de funciones MHEG-SIR utilizando las reglas generales de correspondencia IDL definidas en la cláusula 14 y las reglas específicas de codificación MHEG API definidas en E.2.

### C.2.2 Operaciones MHEG-3 API

Las operaciones MHEG-3 API definidas por la MHEG-3 API, definida en la cláusula 15, se codificarán utilizando los identificadores de funciones predefinidas indicados en el cuadro C.2.

**Cuadro C.2/T.173 – Identificadores de funciones predefinidas para operaciones MHEG-3 API**

| Nombre de la operación | Índice de función predefinida | Identificador de función |
|---|---|---|
| **prepare** | 0 | 1000h |
| **destroy** | 1 | 1001h |
| **new** | 2 | 1002h |
| **delete** | 3 | 1003h |
| **setPriority** | 4 | 1004h |
| **getPriority** | 5 | 1005h |
| **setData** | 6 | 1006h |
| **getData** | 7 | 1007h |
| **allocate** | 8 | 1008h |
| **free** | 9 | 1009h |
| **stop** | 10 | 100Ah |
| **reInit** | 11 | 100Bh |
| **getRtScriptStatus** | 12 | 100Ch |
| **open** | 13 | 100Dh |

**Cuadro C.2/T.173 – Identificadores de funciones predefinidas para operaciones MHEG-3 API** *(fin)*

| Nombre de la operación | Índice de función predefinida | Identificador de función |
|---|---|---|
| **close** | 14 | 100Eh |
| **getRoutineId** | 15 | 100Fh |
| **setParameter** | 16 | 1010h |
| **getPrototype** | 17 | 1011h |
| **run** | 18 | 1012h |
| **reset** | 19 | 1013h |
| **getInvocationStatus** | 20 | 1014h |

Las definiciones de estas operaciones, que figuran en el anexo F, se harán corresponder con las descripciones de funciones MHEG-SIR utilizando las reglas de correspondencia IDL definidas en la cláusula 14.

## C.3 Mensajes predefinidos

Los mensajes predefinidos MHEG-SIR destinados a un guión rt se obtienen como resultado de:

– la invocación de la operación **run** de la MHEG API;

– la excepción **InstructionExecutionError**;

– excepciones MHEG-3 API;

– excepciones MHEG API.

### C.3.1 Operaciones MHEG-3 API

El identificador del mensaje resultante de la invocación de una operación **run**, definida en 15.4.5, será igual al identificador de función de la rutina de destino.

Los mensajes resultantes de operaciones MHEG-3 API tendrán por tanto un valor de identificador de mensaje entre 0 y 0FFFh.

### C.3.2 La excepción InstructionExecutionError

La excepción **InstructionExecutionError**, definida en 9.5.2, tendrá 1000h como su identificador de mensaje.

La excepción **InstructionExecutionError** tendrá un miembro de tipo **unsigned long**, cuyo valor se fijará al valor del registro ER.

El código de error principal determinará el octeto menos significativo del miembro (y el registro ER) como se define en el cuadro C.3.

**Cuadro C.3/T.173 – Códigos de error de ejecución de instrucción**

| Nombre de error | Código de error |
|---|---|
| **InvalidOperand** | 1 |
| **InvalidParameter** | 2 |
| **InvalidType** | 3 |
| **InvalidIdentifier** | 4 |
| **InvalidLevel** | 5 |

**Cuadro C.3/T.173 – Códigos de error de ejecución de instrucción** *(fin)*

| Nombre de error | Código de error |
|---|---|
| **InvalidIndex** | 6 |
| **StackUnderflow** | 7 |
| **ArithmeticOverflow** | 8 |
| **DivisionByZero** | 9 |
| **HandlerNotFound** | 10 |
| **InvalidReturnValue** | 11 |
| **BadPackageStatus** | 12 |
| **InvalidObjectReference** | 13 |
| **TypeMismatch** | 14 |
| **JumpOutOfRange** | 15 |
| **AllocationFailed** | 16 |

### C.3.3 Excepciones MHEG-3 API

Las excepciones MHEG-3 API, definidas en la cláusula 15, deben tener los identificadores de mensajes definidos en el cuadro C.4.

**Cuadro C.4/T.173 – Identificadores de mensajes predefinidos para las excepciones MHEG-3 API**

| Nombre de excepción | Índice de mensaje predefinido | Identificador de mensaje |
|---|---|---|
| **InvalidScript** | 1 | 1001h |
| **InvalidParameter** | 2 | 1002h |
| **OperationFailed** | 3 | 1003h |

La definición IDL de estas excepciones, definidas en el anexo F, se hará corresponder con descripciones de mensajes MHEG-SIR utilizando las reglas de correspondencia IDL definidas en la cláusula 14.

### C.3.4 Excepciones MHEG API

Las excepciones MHEG API definidas por la MHEG API se codificarán utilizando identificadores de mensajes predefinidos.

Los identificadores de mensajes predefinidos para las excepciones MHEG API comienzan en 1100h.

La definición IDL de estas excepciones, proporcionadas por la MHEG API, se harán corresponder con descripciones de mensajes MHEG-SIR utilizando las reglas generales de correspondencia IDL definidas en la cláusula 14 y las reglas específicas de codificación MHEG API definidas en E.2.

## ANEXO D

## Plantilla en IDL para la especificación de correspondencia de plataforma

Los motores MHEG-3 permitirán el acceso a los servicios proporcionados por el entorno de ejecución (*run-time environment*) de una plataforma dada a condición de que este entorno de fase de ejecución cumpla con la "especificación de correspondencia de plataforma" para esta plataforma.

Las "especificaciones de correspondencia de plataforma" registradas se proporcionarán de acuerdo con la plantilla especificada en este anexo, llenándose todos los campos.

La especificación de correspondencia de plataforma MHEG-SIR define los mecanismos que deberán utilizar los motores MHEG-3 para ganar acceso a los servicios proporcionados por el entorno de ejecución en la plataforma.

**Descripción de la plataforma**

La plataforma a que se aplica esta especificación es <descripción_de_plataforma>.

**Procedimiento de disponibilidad de lote**

Para saber si una especificación IDL está disponible en el entorno de ejecución y localizarla, un motor MHEG-3 procederá como sigue. <procedimiento_de_disponibilidad_de_lote>

**Procedimiento de carga de lote**

Para hacer accesibles las operaciones de una especificación IDL disponible, un motor MHEG-3 procederá como sigue. <procedimiento_de_carga_de_lote>

**Procedimiento de descarga de lote**

Para hacer que las operaciones de una especificación IDL disponible dejen de estar accesibles, un motor MHEG-3 procederá como sigue. <procedimiento_de_descarga_de_lote>

**Procedimiento de invocación de operación**

Para invocar una operación de una especificación IDL accesible, un motor MHEG-3 procederá como sigue. <procedimiento_de_invocación_de_operación>

**Procedimiento de paso de parámetros**

Cuando se invoca una operación IDL, un motor MHEG-3 pasará parámetros **in** como sigue. <procedimiento_de_paso_de_parámetros_in>

Cuando se invoca una operación IDL, un motor MHEG-3 pasará parámetros **out** como sigue. <procedimiento_de_paso_de_parámetros_out>

Cuando se invoca una operación IDL, un motor MHEG-3 pasará parámetros **inout** como sigue. <procedimiento_de_paso_de_parámetros_inout>

**Procedimiento de extracción de parámetros de salida**

Para extraer los valores de parámetros **out** o **inout** después de invocar una operación IDL, un motor MHEG-3 procederá como sigue. <procedimiento_de_extracción_de_parámetros_de_salida>

**Procedimiento de extracción de valor de retorno**

Para extraer el valor de retorno de una operación IDL antes invocada, un motor MHEG-3 procederá como sigue. <procedimiento_de_extracción_de_valor_de_retorno>

**Reglas de codificación de datos**

Los valores de datos que se intercambian entre el motor MHEG-3 y el entorno de ejecución se codificarán como sigue <reglas_de_codificación_de_datos>

**Procedimiento de extracción de excepciones**

Para extraer excepciones que hayan sido señaladas por el entorno de ejecución, un motor MHEG-3 procederá como sigue. <procedimiento_de_extracción_de_excepción>

**Excepciones de sistema**

Las excepciones de sistema que pueden ser señaladas por el entorno de ejecución y extraídas por un motor MHEG-3 se definen como sigue. <definiciones_de_excepciones_de_sistema>

**Limitaciones de recursos**

Cuando se está utilizando el entorno de ejecución en la plataforma, son aplicables las siguientes limitaciones de recursos. <enunciado_de_limitaciones_de_recursos>

ANEXO E

## Proceso de definición de la MHEG API

Como se indicó en 8.3.3, esta Recomendación genérica no define una MHEG API específica. En cambio, define un conjunto genérico de reglas y procedimientos aplicables a la definición de la MHEG API que habrá de ser proporcionada por toda Recomendación que describa objetos de presentación. Comprende:

– las reglas que se utilizarán para producir la definición de MHEG API (véase E.1);
– el procedimiento que se utilizarán para definir la correspondencia de la MHEG API con la MHEG-SIR (véase E.2).

### E.1 Marco de la definición de MHEG API genérica

La producción de una especificación MHEG API a partir de otra Recomendación que describe objetos de presentación (que en lo sucesivo se denominará una especificación MHEG) es un proceso que consiste en producir elementos IDL a partir de elementos MHEG.

Los elementos MHEG a que se aplica este proceso se describen en E.1.1. Los elementos IDL producidos a partir de estos elementos MHEG se describen en E.1.2. Las reglas utilizadas para producir los elementos IDL a partir de los elementos MHEG se describen en E.1.3 y en las subcláusulas subsiguientes.

#### E.1.1 Elementos MHEG que están presentes a la entrada del proceso de definición de MHEG API

Las Recomendaciones UIT-T de la serie T.170 (y las diferentes partes de ISO/CEI 13522) comparten cierto número de características fundamentales. Los siguientes elementos MHEG tienen que estar presentes en la especificación MHEG fuente:

– tipos de datos MHEG, descritos mediante ASN.1 o la forma Backus-Naus ampliada (EBNF, *extended Backus-Naur form*);
– entidades MHEG (es decir, objetos sobre los cuales influirán acciones elementales MHEG), entre las cuales existen relaciones de herencia;
– atributos estáticos o dinámicos de entidades MHEG;
– acciones elementales MHEG que se aplican a entidades MHEG;
– excepciones MHEG señaladas como el efecto MHEG de acciones elementales.

### E.1.2 Elementos IDL que están presentes a la salida del proceso de definición de MHEG API

El proceso de definición debe consistir en establecer una correspondencia de estos elementos con un conjunto de elementos IDL:

– los tipos no objeto IDL deben corresponder a tipos de datos MHEG;

– las interfaces de objeto IDL entre las cuales hay una relación de herencia deben corresponder a entidades MHEG;

– los atributos IDL proporcionados por interfaces de objeto IDL deben corresponder a atributos estáticos y dinámicos de entidades MHEG;

– las operaciones IDL proporcionadas por interfaces de objeto IDL deben corresponder a acciones elementales MHEG;

– las excepciones IDL deben corresponder a excepciones MHEG señaladas como el efecto de acciones elementales.

### E.1.3 Requisitos del proceso de definición de MHEG API

De acuerdo con las directrices del JTC 1 de la ISO/CEI para la normalización de la API, la MHEG API se definirá como una especificación API abstracta, es decir, una descripción, independiente del lenguaje de programación, de la semántica de un conjunto de funcionalidades en una sintaxis abstracta mediante el empleo de tipos de datos abstractos.

En cumplimiento de las recomendaciones del Informe técnico ETR 225 (API y representación en guión para MHEG – Requisitos y marco) ("API and script representation for MHEG – Requirements and framework"), una definición de MHEG API debe cumplir los siguientes requisitos:

– portabilidad (véase E.1.3.1);

– genericidad (véase E.1.3.2);

– posibilidad de pruebas de conformidad (véase E.1.3.3);

– implementabilidad (véase E.1.3.4).

#### E.1.3.1 Portabilidad

El requisito de **portabilidad** indica que las aplicaciones MHEG deben utilizar los servicios de manipulación e intercambio de objetos proporcionados por motores MHEG (esto es, una MHEG API) de una manera independiente

– del lenguaje de programación utilizado para la aplicación MHEG;

– del sistema operativo subyacente.

Para satisfacer el requisito de la portabilidad, una MHEG API deberá definirse como una especificación de API abstracta.

#### E.1.3.2 Genericidad

El requisito de **genericidad** indica que todos los requisitos comunes de las aplicaciones MHEG deben ser satisfechos por una MHEG API.

Para satisfacer el requisito de la genericidad, una MHEG API deberá definirse en el nivel más primitivo, es decir, en términos de elementos primitivos que concuerdan con acciones elementales MHEG, y tipos de datos que concuerdan con tipos de datos MHEG. Esto garantiza que la gama de manipulaciones de objetos MHEG puestas a disposición de las aplicaciones será máxima.

### E.1.3.3 Posibilidad de pruebas de conformidad

El requisito de la **posibilidad de pruebas de conformidad** indica que debe ser lo más fácil posible efectuar pruebas para determinar

– que un motor MHEG es conforme con una especificación de MHEG API, es decir, la provisión correcta de esta API por un motor MHEG sometido a prueba;

– que una aplicación MHEG es conforme con una especificación de MHEG API, es decir, el uso correcto de esta API por una aplicación MHEG sometida a prueba.

Para satisfacer el requisito de posibilidad de pruebas de conformidad, una MHEG API deberá expresar formalmente los requisitos de implementaciones conformes y de aplicaciones conformes, y utilizará una técnica de descripción formal para la definición de la MHEG API.

### E.1.3.4 Implementabilidad

El requisito de la **implementabilidad** indica que la implementación de motores MHEG que sean conformes con la especificación de la MHEG API debe ser lo más fácil posible. Con este fin, en la definición de la MHEG API debe tenerse en cuenta la simplicidad y la claridad, tanto en la definición propiamente dicha, como en la formulación.

Para satisfacer el requisito de la implementabilidad, una MHEG API debe proporcionar, o hacer referencia a, directrices para producir especificaciones de correspondencias de lenguajes y reglas de codificación de mensajes a partir de la especificación de la API abstracta.

### E.1.3.5 Cumplimiento de requisitos técnicos

La utilización de IDL contribuye a al cumplimiento de los requisitos técnicos de portabilidad e implementabilidad.

– El lenguaje IDL es independiente del lenguaje de programación. Además, hay especificaciones públicas de las correspondencias de IDL con lenguajes de programación usuales como C y C++;

– IDL proporciona un lenguaje de descripción formal completo, que permite una especificación muy concisa, legible y eficiente de una MHEG API. Además, IDL es también apropiado para compilación automática, lo que permite generar automáticamente implementaciones de MHEG API para un determinado lenguaje y sistema operativo, utilizando compiladores IDL apropiados.

## E.1.4 Estructura general de la MHEG API

La MHEG API se definirá utilizando el lenguaje de definición de interfaz IDL, definido en ISO/CEI 14750-1 [8]. La utilización de IDL no implica un entorno arquitectura de intermediario de petición de objeto común (CORBA, *common object request broker architecture*). Por tanto, además de utilizarse IDL, la MHEG API deberá definirse de una manera independiente de la arquitectura CORBA.

La definición de la interfaz IDL consistirá en dos módulos que proporcionan una funcionalidad API diferentes. El primer módulo proporcionará servicios de interpretación MHEG. Se denominará módulo **MHEG_<part>**, donde **<part>** es un número que designa la Recomendación a la que pretende ajustarse la API. El segundo módulo proporcionará servicios de accesor y modificador para entidades MHEG codificadas. Se denominará módulo **MHEG_<part>_Access**, donde **<part>** es un número que designa la Recomendación a la que pretende ajustarse la API.

La API proporcionará una interfaz que permite la creación de una instancia de objeto de interfaz. Esta interfaz se denominará **MHEGEntityManager**. Las interfaces que correspondan a entidades MHEG proporcionarán operaciones que permitan la supresión de una instancia IDL.

Cada interfaz que corresponde a una entidad MHEG que proporciona un esquema de referencia común para sus subclases deberá también proporcionar las siguientes operaciones:

– **attach** (ligar);

– **detach** (desligar);

– **getIdentifier** (obtener identificador).

La operación **attach** (ligar) vincula una instancia de objeto de interfaz con una entidad MHEG y retorna el identificador del objeto vinculado. La operación **detach** (desligar) anula la vinculación. La operación **getIdentifier** (obtener identificador) extrae el identificador de la entidad MHEG vinculada con la instancia de objeto de interfaz.

Los conceptos de IDL utilizados para diseñar la interfaz corresponderán con conceptos MHEG de acuerdo con las reglas descritas en las subcláusulas siguientes. Los identificadores IDL corresponderán con identificadores ASN.1 de MHEG de acuerdo con las reglas definidas en las subcláusulas siguientes.

Las siguientes operaciones serán síncronas: operaciones de crear, vincular, desvincular, accesores, modificadores, operaciones de obtener (GET). Otras acciones elementales se harán corresponder con operaciones asíncronas.

En la definición de API no podrá haber niveles de alcance contenidos unos dentro de otros ("anidados").

### E.1.5 Definición de tipos de datos no objeto

Los tipos no objeto IDL corresponderán con tipos de datos MHEG expresados utilizando la ASN.1.

#### E.1.5.1 Correspondencia de nombres

##### E.1.5.1.1 Tipos de datos

Los nombres de los tipos de datos ASN.1 se harán corresponder con nombres de tipos IDL como sigue:

– si el nombre está formado por más de una palabra, no se utilizarán separadores en el nombre IDL;

– cada palabra comenzará por una letra mayúscula, y todas las demás letras serán minúsculas.

Ejemplo: El **type-name** ASN.1 se hará corresponder con el **TypeName** IDL.

##### E.1.5.1.2 Componentes

Los nombres de componentes ASN.1 se harán corresponder con nombres de campos IDL como sigue:

– si el nombre ASN.1 está formado por más de una palabra, se utilizara el carácter de subrayado ("_") como separador;

– todas las letras serán minúsculas;

– puesto que IDL es insensible a la escritura en mayúsculas o minúsculas, los nombre de tipos y los nombres de campos podrían entrar en colisión cuando se utilizan nombres formados por una sola palabra. En este caso, el nombre de campo debe ir precedido por ("the_"). Se aplicará la misma regla cuando un componente formado por una sola palabra pudiera entrar en colisión con una palabra clave IDL.

Ejemplo:

```
// regular field name
TypeName    field_name

// collision between field name and type name
Alias       the_alias

// collision between field name and IDL keyword ("string")
Type        the_string
```

#### E.1.5.1.3 Valores

Los nombres de valores ASN.1 se harán corresponder con valores IDL como sigue:

a) si el nombre está formado por más de una palabra, se utilizará el carácter de subrayado ("_") como separador;

b) todas las letras serán mayúsculas;

c) puesto que una enumeración IDL no crea un nuevo alcance (scope), los valores utilizados en enumeraciones podría entrar en colisión unos con otros. En tal caso, para una de las enumeraciones que entrarían en colisión, todos los nombres de valores (no sólo el que entra en colisión) deberán ir precedidos por el nombre del tipo enumerado. Este nombre de tipo se construirá aplicando las reglas a) y b) (y no las reglas usuales para nombres de tipo).

Ejemplo:

```
// regular value name
FIRST_VALUE

// value name in case of a collision
FIRST_ENUMERATION_FIRST_VALUE
```

### E.1.5.2 Correspondencia de tipos

#### E.1.5.2.1 INTEGER (entero)

Un tipo **INTEGER** ASN.1 se hará corresponder con un tipo **long** IDL.

Ejemplo:

```
-- ASN.1
Type ::= INTEGER

// IDL
typedef long Type;
```

#### E.1.5.2.2 BOOLEAN (booleano)

Un tipo **BOOLEAN** ASN.1 se hará corresponder con un tipo **boolean** IDL.

Ejemplo:

```
-- ASN.1
Type ::= BOOLEAN

// IDL
typedef boolean Type;
```

#### E.1.5.2.3 OCTET STRING (cadena de octetos)

Un tipo **OCTET STRING** ASN.1 se hará corresponder con un tipo **sequence <octet>** IDL.

Ejemplo:

```
-- ASN.1
Type ::= OCTET STRING

// IDL
typedef sequence <octet> Type;
```

#### E.1.5.2.4 ENUMERATED (enumerado)

Un tipo **ENUMERATED** ASN.1 se hará corresponder con un tipo **enum** IDL.

Ejemplo:

```
-- ASN.1
Type ::= ENUMERATED {value_1 (1), value_2 (2)}

// IDL
enum Type (VALUE_1, VALUE_2);
```

#### E.1.5.2.5 SEQUENCE OF (secuencia de)

Un tipo **SEQUENCE OF** ASN.1 se hará corresponder con un tipo **sequence** IDL.

Ejemplo:

```
-- ASN.1
Type ::= SEQUENCE OF OtherType

// IDL
typedef sequence <OtherType> Type;
```

#### E.1.5.2.6 CHOICE (elección)

Un tipo **CHOICE** ASN.1 se hará corresponder con un tipo union (unión) discriminada mediante una combinación de **enum** y **union**. El nombre de tipo **enum** se derivará del nombre del tipo **CHOICE** al que se añade el sufijo "Tag" para que no entre en colisión con el nombre de tipo **union**. Los nombres de valores **enum** se derivarán de los nombres de los campos de tipo **CHOICE** a los que se añade el sufijo "_TAG" para que no entren en colisión con los nombres de campos dentro del **switch**. Dentro de un **CHOICE**, un componente **NULL** se hará corresponder con un **case** vacío.

Ejemplo:

```
-- ASN.1
Type ::= CHOICE
{
        a       A,
        b       B,
        c       NULL
}

// IDL
enum TypeTag { A_TAG, B_TAG, C_TAG };
union Type
switch TypeTag {
        case A_TAG:
                A       the_a;
        case B_TAG:
```

```
        B       the_b;
    // no'case' for C_TAG
};
```

#### E.1.5.2.7 SEQUENCE (secuencia)

Un tipo **SEQUENCE** ASN.1 se hará corresponder con un tipo **struct** IDL. Los componentes ASN.1 **OPTIONAL** con o sin valores **DEFAULT** se harán corresponder con una **sequence** IDL de, como máximo, un elemento de ese tipo.

Ejemplo:

```
-- ASN.1
Type ::= SEQUENCE
{
    a       A OPTIONAL,
    b       B,
    c       INTEGER DEFAULT 0
}

// IDL
struct Type {
    sequence <A,1>          the_a;
    B                   the_b;
    sequence <long,1>       c;
};
```

### E.1.5.3 Orden de las declaraciones

El orden de las declaraciones de tipos puede ser diferente en ASN.1 e IDL porque IDL no permite utilizar un tipo antes de que haya sido declarado [(y no proporciona una funcionalidad de declaración hacia adelante (*forward declaration*)]. Las declaraciones de tipo IDL deberán reordenarse de modo que se satisfaga esta restricción.

Para permitir la reordenación, las referencias cruzadas de la sintaxis ASN.1 deberán suprimirse.

Los siguientes ejemplos muestran referencias cruzadas usualmente utilizadas en ASN.1 y la forma de suprimirlas.

Ejemplo 1:

```
-- ASN.1 definition using a cross-reference
-- Production rules used :
-- T = A | B
-- B = T C
-- Allowed values :
-- a c*

-- T refers to B
T ::= CHOICE
{
    a       A,
    b       B
}
-- B refers to T
B ::= SEQUENCE
{
    t       T,
    c       C
}
```

*-- Equivalent ASN.1 definition without cross-reference*
*-- The information described remains the same, the structure has changed*
*-- Production rules used :*
*-- T = A C**

```
T ::= SEQUENCE
{
        a       A,
        s_o_c   SEQUENCE OF C OPTIONAL
}
```

Ejemplo 2:

*-- ASN.1 definition using a cross-reference*
*-- Production rules used :*
*-- T = A | B*
*-- B = C T*
*-- Allowed values :*
*-- c*a*

```
-- T refers to B
T ::= CHOICE
{
        a       A,
        b       B
}
-- B refers to T
B ::= SEQUENCE
{
        c       C,
        t       T
}
```

*-- Equivalent ASN.1 definition without cross-reference*
*-- The information described remains the same, the structure has changed*
*-- Production rules used :*
*-- T = C* A*

```
T ::= SEQUENCE
{
        s_o_c   SEQUENCE OF C OPTIONAL,
        a       A
}
```

Sin embargo, en la mayoría de los casos, las referencias cruzadas deben suprimirse mediante la creación de tipos anidados, como se muestra en el ejemplo 3.

Ejemplo 3:

*-- ASN.1 definition using a cross-reference*

```
-- T refers to A
T ::= CHOICE
{
        a       A,
        b       B
}
```

**A ::=** *-- A definition that refers to T (either directly or indirectly)*

*-- Equivalent ASN.1 definition without cross-reference*

```
T ::= CHOICE
{
        a       -- A definition
        b       B
}
```

El inconveniente del método mostrado en el ejemplo 3 es que los tipos de datos creados podrían ser muy complejos. Además, parte de los tipos de datos así creados podrían duplicar tipos ya definidos en la sintaxis.

Otra posibilidad es traducir el tipo de datos "A" ASN.1 en una interfaz "A" IDL que tuviese insertado el correspondiente tipo de datos "A" IDL. Esto permite la utilización de una declaración hacia adelante de la interfaz "A" de IDL, como se muestra en el ejemplo 4.

Ejemplo 4:

```
interface A; // forward declaration

enum TTag { A_TAG, B_TAG };
union T
switch TTag {
        case A_TAG:
                A       the_a;
        case B_TAG:
                B       the_b;
};

interface A {

// this interface embeds the definition of type A which refers to type T

};
```

### E.1.6 Definición de interfaz IDL

Las entidades MHEG deberán hacerse corresponder con interfaces IDL de acuerdo con las reglas siguientes:

– todo elemento que pueda designarse como un objetivo de una acción elemental MHEG se considerará como una entidad. No todas las entidades están explícitamente definidas por la norma MHEG. Además, no todas ellas existen en el modelo de objeto MHEG. El modelo de objeto MHEG deberá refinarse de modo que integre todas las entidades utilizadas;

– si una acción elemental puede estar dirigida a entidades de tipos diferentes, se creará una nueva entidad que tenga todas estas entidades agregadas;

– el nombre de la nueva entidad creada debe ser la concatenación de los nombres de las entidades agregadas en ella, utilizándose "Or" como separador;

– el modelo de objeto obtenido es el modelo de objeto de API. Habrá una correspondencia biunívoca entre entidades e interfaces;

– la jerarquía de herencia IDL corresponderá a la jerarquía del modelo de objeto de API;

– los nombres de interfaz se derivarán de los nombres de entidad aplicando las mismas reglas que para los tipos de datos.

### E.1.7 Definición de atributo IDL

Se identifican las siguientes categorías de atributo MHEG:

– atributos intercambiados;

– atributos internos.

#### E.1.7.1 Atributos intercambiados MHEG

Los atributos intercambiados MHEG se harán corresponder con atributos IDL en el módulo **MHEG-<part>-access** de acuerdo con las siguientes reglas:

a) todo tipo ASN.1 cuyo nombre termine por "Class" se hará corresponder a una interfaz;

b) dentro del tipo [véase a)], los ASN.1 **COMPONENTS OF** ASN.1 se harán corresponder con la funcionalidad de herencia IDL;

c) dentro del tipo [véase a)], cada componente se hará corresponder con un atributo;

d) los atributos de denominación y tipificación seguirán las reglas definidas en E.1.5.

Ejemplo:

```
-- ASN.1

B-Class ::= SEQUENCE
{
        COMPONENTS OF       A-Class,
        i                   INTEGER,
        j                   C-Type
}

// IDL

interface BClass : AClass {
        attribute long      i;
        attribute CType     j;
};
```

#### E.1.7.2 Atributos internos MHEG

Los atributos internos MHEG no se harán corresponder con atributos IDL. Se ganará acceso, y se modificarán, los atributos internos MHEG mediante acciones elementales MHEG que deberán hacerse corresponder con operaciones IDL.

### E.1.8 Definición de operación IDL

Las interfaces MHEG API proporcionarán las siguientes categorías de operaciones:

– operaciones que establecen la correspondencia de acciones elementales MHEG dirigidas a la entidad MHEG a que corresponde la interfaz;

– operaciones utilizadas para suprimir una instancia de interfaz (véase la nota);

– operaciones utilizadas para ligar y desligar una instancia de interfaz a/de una entidad MHEG.

NOTA – La creación de una instancia de interfaz la proporciona la interfaz de fábrica.

#### E.1.8.1 Operaciones que establecen la correspondencia de acciones elementales MHEG

Las acciones elementales MHEG se harán corresponder con operaciones IDL de acuerdo con las siguientes reglas:

a) Habrá una correspondencia biunívoca entre acciones elementales y operaciones.

b) La interfaz que proporciona una operación corresponderá con la entidad que es el objetivo (destino) de una acción elemental.

c) El nombre de la operación se derivará del nombre de la acción elemental aplicando las siguientes reglas:

   1) si un nombre de operación está formado por más de una palabra, no se utilizarán separadores;

   2) el nombre de la operación empezará con una letra minúscula;

   3) cada palabra (salvo la primera) empezará con una letra mayúscula.

d) Una acción elemental de "Set" (fijar) se hará corresponder con una operación de acuerdo con las siguientes reglas:

   1) el valor de retorno de la operación será de tipo **void**;

   2) sólo habrá parámetros de entrada;

   3) los tipos de datos y los nombre de los parámetros se derivarán de la definición de acción elemental ASN.1 aplicando las reglas definidas en E.1.5;

   4) no se establecerá la correspondencia del primer parámetro de la definición de acción elemental ASN.1 (dicho parámetro representa el objetivo de la acción elemental);

   5) si se utiliza la funcionalidad de macro para la definición de un parámetro, deberá utilizarse el tipo de datos incrustado (*embedded datatype*);

   6) si el tipo de datos incrustado a que se hace referencia en d 5) no está explícitamente definido, deberá crearse como se indica en el Ejemplo 1 en E.1.5.3.

e) Una acción elemental de "Get" (obtener) se hará corresponder con una operación de acuerdo con las siguientes reglas:

   1) el valor de retorno de la operación será el valor evaluado por la acción elemental;

   2) el tipo de datos del valor de retorno no existe como tal en la sintaxis ASN.1 de MHEG porque las acciones elementales están evaluando un valor genérico. En consecuencia, será uno de los siguientes:

      i) un tipo de datos simple;

      ii) un tipo de datos complejo que se utiliza en la correspondiente acción elemental de "Set";

      iii) si se retorna un tipo de datos complejo y no existe una acción elemental de "Set" correspondiente, se creará el tipo de datos.

   3) no habrá parámetros de salida, todos los resultados se pasarán utilizando el valor de retorno;

   4) para los parámetros de entrada se aplicarán las reglas definidas en d 3), d 4), d 5) y d 6).

Ejemplo:

```
Elementary-Action-N ::=  SEQUENCE
{
        target          Target, -- to be skipped
        param1-param    Param1-Parameter, -- replace
        param2-param    Param2-Parameter -- replace
}
```

```
Param1-Parameter ::= CHOICE
{
        param1              Param1,
        param1-macro        Param1-Macro
}

Param1-Macro ::=  SEQUENCE
{
        -- ...
        -- ...
}
Param1 ::=   -- ...

--      In "Elementary-Action-N" the "param1-parameter" parameter shall be replaced with the
--      "param1" parameter

Param2-Parameter ::= CHOICE
{
        a-param                 A-Parameter,
        b-param                 B-Parameter
}

A-Parameter ::= CHOICE
{
        a                   A,
        a-macro                 A-Macro
}

A-Macro ::= SEQUENCE
{
        -- ...
        -- ...
}

A ::=           -- ...

B-Parameter ::=   SEQUENCE
{
        -- ...
        -- ...
}

B-Macro ::= SEQUENCE
{
        -- ...
        -- ...
}

B ::=           -- ...

--      In "Elementary-Action-N" the "param2-parameter" parameter cannot be replaced with the
--      "param2" parameter because it does not exist. Consequently an IDL datatype that
--      corresponds to the following ASN.1 datatype shall be created:

Param2 ::= CHOICE
{
        a               A,
        b               B
}
```

### E.1.8.2 Operaciones que permiten la supresión de una instancia de interfaz

Cada interfaz proporcionará una operación que permita la supresión de una instancia de interfaz. Esta operación tendrá el siguiente prototipo:

**void kill()**;

### E.1.8.3 Operaciones para ligar y desligar una instancia de interfaz a/de una entidad MHEG

Cada interfaz proporcionará operaciones que permitan ligar desligar a, respectivamente desligar de, una entidad MHEG. Estas operaciones se denominarán **attach** y **detach**, respectivamente. La operación **attach** aceptará una referencia de entidad MHEG como parámetro, resolverá esta referencia y retornará un identificador de entidad MHEG. La operación **detach** no aceptará ningún parámetro.

Cada interfaz proporcionará una operación para extraer el identificador de la entidad MHEG ligada una instancia de interfaz. Esta operación se denominará **getIdentifier** (obtener identificador). No aceptará parámetros y retornará un identificador de entidad MHEG.

## E.1.9 Definición de excepción IDL

Las excepciones IDL corresponderán con condiciones de error MHEG asociadas con acciones elementales. Las condiciones de error deben clasificarse de acuerdo con su naturaleza de modo que se obtenga un conjunto limitado de excepciones IDL.

Se recomienda el siguiente conjunto de excepciones:

– **InvalidTarget** (objetivo no válido);

– **InvalidParameter** (parámetro no válido);

– **NotBound** (no vinculado);

– **AlreadyBound** (ya vinculado).

La excepción **InvalidTarget** se señala cuando la entidad MHEG de destino no está disponible. El miembro **period** (periodo) retorna el estado actual (periodo de ciclo de vida) del objetivo.

La excepción **InvalidParameter** se señala cuando el valor de uno de los parámetros prohíbe la ejecución normal de la acción. El miembro **completion_status** (estado de compleción) indica si la acción ha sido concluida (con el valor por defecto asignado al parámetro inadecuado), o no. El miembro **parameter_number** (número de parámetro) identifica el rango del parámetro no válido.

La excepción **NotBound** se señala cuando la instancia de objeto de interfaz no está vinculada con una entidad MHEG.

La excepción **AlreadyBound** se señala cuando la instancia de objeto de interfaz ya está vinculada con una entidad MHEG. El miembro **entity_identifier** (identificador de entidad) identifica la entidad vinculada.

Las definiciones de estas excepciones son las siguientes:

```
exception InvalidTarget {
        unsigned short period;
};

enum CompletionStatus { YES, NO };

exception InvalidParameter {
        CompletionStatus   completion_status;
        unsigned short     parameter_number;
};

typedef long EntityIdentifier;
```

```
exception AlreadyBound {
        EntityIdentifier entity_identifier;
};

exception NotBound {};
```

Estas excepciones se definirán con el alcance global.

Si las mencionadas excepciones no pueden tratar una condición de error, se creará una excepción específica al nivel apropiado de alcance de la interfaz.

Los nombres de excepción IDL en el caso de las excepciones específicas se derivarán del nombre de la condición de error MHEG aplicando las mismas reglas que para los tipos de datos:

– no se utilizarán separadores;

– el nombre de la excepción empezará con una letra mayúscula;

– si un nombre de excepción está formado por más de una palabra, no se utilizarán separadores y cada palabra empezará con una letra mayúscula.

## E.2 Correspondencia de MHEG API a MHEG-SIR

El establecimiento de la correspondencia de MHEG API a MHEG-SIR es un proceso que consiste en asignar identificadores de tipos predefinidos, identificadores de funciones predefinidas e identificadores de mensajes predefinidos para hacer corresponder tipos, operaciones y excepciones MHEG API.

Este proceso de establecimiento de correspondencias consiste en tres pasos consecutivos:

a) se extraen todos los tipos intermedios hasta que las definiciones de tipo sólo tengan un nivel (véase el Ejemplo 1 en E.1.5.3);

b) se cambian los tipos **enum** IDL por **unsigned long** (MHEG-SIR no proporciona el tipo de datos enumerados). Se calcula para cada definición de tipo su número de "nivel", es decir, su mayor distancia posible desde la hoja. Se unifican los tipos equivalentes, empezando por los de número de "nivel" más bajo;

c) se explora la especificación IDL resultante para atribuir identificadores predefinidos. Se asignan identificadores de tipo (TID) e identificadores de mensaje (MID) predefinidos tan pronto como se encuentran nuevos tipos o excepciones. Por cada objeto, se asignan identificadores de funciones (FID) predefinidas para cada operación definida explícitamente, para cada operación heredada que no es contraordenada por una definida explícitamente, para cada accesor de atributo y para cada modificador de atributo.

Ejemplo:

```
struct A {
        int a;
        struct {
                int b;
                struct {
                        int c;
                        int d;
                } b
        } c
};

// after extraction of intermediate types (a))
```

```
struct A {
        int a;
        B b;
};
struct B {
        int b;
        C c;
};
struct C {
        int c;
        int d;
};
```

ANEXO F

## Especificación IDL de la MHEG-3 API

Para expresar la manipulación directa de otros guiones, los guiones intercambiados utilizarán la correspondencia MHEG-3 (definida en el anexo B) de la MHEG-3 API.

Los motores MHEG-3 proporcionarán la MHEG-3 API como se especifica por el módulo **MHEG_3** de IDL.

```
module MHEG_3  {

        enum RtScriptStatus {RUNNING, READY, ERRONEOUS};
        enum InvocationStatus {NOT_STARTED, PROCESSING, TERMINATED, ABORTED};
        enum PassingMode {BY_VALUE, BY_REFERENCE};
        enum Entity {TYPE, DATA, FUNCTION, MESSAGE, PACKAGE, HANDLER};

        typedef unsigned short FID;
        typedef unsigned short DID;
        typedef unsigned short TID;

        struct ContentReference {
                string public_id;
                string system_id;
        };
        struct ParameterDescription {
                PassingMode         passing_mode;
                TID                 parameter_type_id;
        };
        struct Prototype {
                TID                 return_value_type_id;
                sequence<ParameterDescription> signature;
        };

        exception InvalidParameter {
                unsigned short rank;
        };
        exception InvalidScript {
                Entity          the_entity;
                unsigned short identifier;
        };
        exception OperationFailed {};

        interface MhScript;
        interface RtScript;
        interface RoutineInvocation;
```

```
        interface ScriptInterpreter {
                MhScript prepare (in ContentReference content_reference)
                        raises (InvalidScript, InvalidParameter, OperationFailed);
                void kill ();
        };

        interface MhScript {
                RtScript new ()
                        raises (OperationFailed);
                void destroy ();
        };
        interface RtScript {
                RoutineInvocation open (in FID routine_id)
                        raises (InvalidParameter);

                RtScriptStatus getRtScriptStatus();

                void setPriority (in unsigned short priority);
                unsigned short getPriority ();

                DID allocate (in TID variable_type_id)
                        raises (InvalidParameter, OperationFailed);
                void free (in DID variable_id)
                        raises (InvalidParameter);

                void setData (in DID variable_id, in any variable_value)
                        raises (InvalidParameter, OperationFailed);
                any getData (in DID data_id)
                        raises (InvalidParameter, OperationFailed);

                void stop ()
                        raises (OperationFailed);
                void reInit ()
                        raises (OperationFailed);
                void delete ();
        };

        interface RoutineInvocation    {

                readonly attribute FID routine_id;

                void setParameter (in unsigned short rank, in TID parameter_type_id,
                                        in any parameter_value)
                        raises(InvalidParameter);

                Prototype getPrototype ();
                InvocationStatus getInvocationStatus ();

                void run ()
                        raises (OperationFailed);

                void reset ();
                void close ();
        };
};
```

ANEXO G

# Relaciones con otras partes de las Recomendaciones UIT-T de la serie T.170 (y partes de ISO/CEI 13522)

## G.1 Relaciones con la Rec. UIT-T T.171 (e ISO/CEI 13522-1)

Esta subcláusula especifica los requisitos aplicables cuando esta Recomendación se utiliza para ampliar las disposiciones de la Rec. UIT-T T.171 (e ISO/CEI 13522-1) [5].

La expresión "otras Recomendaciones UIT-T de la serie T.170" (y "una de las partes de ISO/CEI 13522") utilizada en esta Recomendación se interpretará como "la Rec. UIT-T T.171" ("ISO/CEI 13522-1").

La MHEG-SIR definida en esta Recomendación se aplicará a la codificación del componente **script-data** de objetos conformes con la Rec. UIT-T T.171 (ISO/CEI 13522-1) de la clase **script**, cuyo componente **script-classification** está registrado en el catálogo **registered-script-classification** para indicar "guión" y cuyo componente **catalogued-script-encoding** está registrado en la **registered-script-classification** para indicar "MHEG-SIR", de acuerdo con los valores mantenidos por la autoridad de registro MHEG.

NOTA – Estos valores están registrados de acuerdo con las disposiciones de ISO/CEI 13522-4 [6].

En este marco, el componente **script-data** se codificará de acuerdo con las siguientes restricciones al módulo **ISOMHEG-sc** definido en la Rec. UIT-T T.171 (ISO/CEI 13522-1) [5]:

– si la elección **script-inclusion** se selecciona para el componente **script-data**, se seleccionará la elección **interchangedscript** para el componente **script-inclusion**;

– si la elección **data-reference** se selecciona para el componente **script-data**, los datos referenciados se codificarán como un **InterchangedScript** como se especifica por el módulo **ISO-MHEG-sir**.

Por consiguiente, los objetos de guión MHEG-3 serán conformes con la sintaxis definida por el subtipo **SIR-Script-Class** que se define más adelante.

```
-- ISOMHEG-sir {joint-iso-itu-t(2) mheg (19) version (1) script-interchanged-representation (11)}

IMPORTS
        MHEG-Identifier, Catalogued-Script-Classification
                FROM ISOMHEG-ud { joint-iso-itu(2) mheg(19) version(1) useful-definitions (9) }
        Registered-Script-Encoding, Registered-Script-Classification
                FROM ISOMHEG-cat { joint-iso-itu(2) mheg(19) version(1) catalogues (12) }
        Script-Class
                FROM ISOMHEG-sc { joint-iso-itu(2) mheg(19) version(1) script-class(3) }
;

SIR-Script-Class ::= Script-Class (WITH COMPONENTS
{       ...,
        script-classification (WITH COMPONENTS
        {       registered-script-classification     ('script') }) PRESENT,
                                        -- registered value as provided by ISOMHEG-cat

        script-hook (WITH COMPONENTS
        {       ...,
                catalogued-script-encoding (WITH COMPONENTS
                {       registered-script-encoding ('MHEG-SIR') }) PRESENT,
                                        -- registered value as provided by ISOMHEG-cat
        }) PRESENT,
```

```
        script-data (WITH COMPONENTS
        {       ...,
                script-inclusion (WITH COMPONENTS       { interchangedscript }),
                data-reference (WITH COMPONENTS         { null-data ABSENT })
        }) PRESENT
})
```

## G.2 Relaciones con la Rec. UIT-T T.172 (e ISO/CEI 13522-5)

Esta subcláusula especifica los requisitos aplicables cuando esta Recomendación se utiliza para ampliar las disposiciones de la Rec. UIT-T T.172 (e ISO/CEI 13522-5) [7].

La expresión "otras Recomendaciones UIT-T de la serie T.170" ("una de las partes de ISO/CEI 13522") utilizada en esta Recomendación se interpretará como "la Rec. UIT-T T.172" ("ISO/CEI 13522-5").

La MHEG-SIR definida en esta Recomendación se aplicará a la codificación del atributo **original-content** de objetos de **InterchangedProgram** conformes con la Rec. UIT-T T.172 (e ISO/CEI 13522-5) cuyo componente **content-hook** indica "MHEG-SIR", de acuerdo con la definición del dominio de aplicación.

Dentro de un valor del tipo **InterchangedProgramClass**, las siguientes restricciones se aplicarán al módulo **ISOMHEG-MHEG-5** definido en la Rec. UIT-T T.172 (e ISO/CEI 13522-5) [7]:

– si el **included-content** está seleccionado para el componente **original-content**, el valor **OCTET STRING** del componente **included-content** se sustituirá por un valor del tipo **InterchangedScript** como se especifica por el módulo **ISO-MHEG-sir**;

– si el **referenced-content** se selecciona para el componente **original-content**, los datos referenciados serán un valor de del tipo **InterchangedScript** como se especifica por el módulo **ISO-MHEG-sir**.

Por consiguiente, los objetos de guión MHEG-3 serán conformes con la sintaxis definida por el subtipo **SIR-Script-Class** más adelante.

```
ISOMHEG-sir { joint-iso-itu-t(2) mheg (19) version (1) script-interchanged-representation (11) }

IMPORTS
        InterchangedProgramClass
                FROM ISOMHEG-MHEG-5 { joint-iso-itu(2) mheg(19) version(1) MHEG-5 (17) }
;

SIR-Script-Class ::= InterchangedProgramClass (WITH COMPONENTS
{       ...,
        content-hook ('MHEG-SIR')  PRESENT,

        original-content PRESENT,
                -- data encoded as InterchangedScript
})
```

La Rec. UIT-T T.172 (e ISO/CEI 13522-5) [7] no distingue entre objetos mh y objetos rt; tanto los unos como los otros se denominan "objetos MHEG-5". Por consiguiente, habrá un solo guión rt para cada guión mh. En consecuencia:

– la inicialización de guión mh y la inicialización de guión rt deberá efectuarse en una sola operación (que corresponde a la invocación de las operaciones **prepare** y **new**);

– la invocación de varias operaciones nuevas (new) sobre el mismo guión mh señalará una excepción;

– la destrucción de guión rt y la destrucción de guión mh deberá efectuarse en una sola operación (que corresponde a la invocación de las operaciones **delete** y **destroy**);

## APÉNDICE I

## Sintaxis MHEG-SIR (notación EBNF)

Este apéndice describe la sintaxis de guiones intercambiados MHEG-SIR. Esta sintaxis es equivalente a la especificación ASN.1 en el anexo A.

```
// Structure
InterchangedScript      ::=  TypeDeclaration*
                             ConstantDeclaration*
                             VariableDeclaration*
                             PackageDeclaration*
                             HandlerDeclaration*
                             RoutineDeclaration*

// Type declarations
TypeDeclaration         ::=  TypeIdentifier?
                             TypeDescription

TypeDescription         ::=  SequenceDescription
                        |    StringDescription
                        |    ArrayDescription
                        |    StructureDescription
                        |    UnionDescription

SequenceDescription          ::=  INTEGER?        // Sequence bound
                             TypeIdentifier

StringDescription       ::=  INTEGER?        // String bound

ArrayDescription        ::=  INTEGER         // Array size
                             TypeIdentifier

UnionDescription        ::=  TypeIdentifier+

StructureDescription         ::=  TypeIdentifier+

// Data declarations
ConstantDeclaration     ::=  DataIdentifier?
                             TypeIdentifier
                             ConstantValue

ConstantValue           ::=  BOOLEAN
                        |    OCTET
                        |    INTEGER          // all numeric types
                        |    REAL             // float or double
                        |    STRING           // character or string
                        |    DataIdentifier
                        |    ConstantValue*       // sequence, array or structure
                        |    UnionValue

UnionValue              ::=  INTEGER          // Tag index
                             ConstantValue
```

```
VariableDeclaration             ::=  DataIdentifier?
                                     TypeIdentifier
                                     ConstantReference?      // Initial value

ConstantReference               ::=  DataIdentifier
                                |    ConstantValue

// Package declarations
PackageDeclaration              ::=  PackageIdentifier?
                                     VisibleString           // Package name
                                     ServiceDescription*
                                     ExceptionDescription*

ServiceDescription              ::=  FunctionIdentifier?
                                     VisibleString?          // IDL global name
                                     CallingMode?
                                     TypeIdentifier?         // return value
                                     ParameterDescription*

ServiceParameterDescription     ::=  ServicePassingMode?
                                     TypeIdentifier

CallingMode                     ::=  'SYNCHRONOUS' | 'ASYNCHRONOUS'
ServicePassingMode              ::=  'IN' | 'OUT' | 'INOUT'

ExceptionDescription            ::=  MessageIdentifier?
                                     VisibleString?          //IDL exception global name
                                     TypeIdentifier*         //Parameter types

// Handler declarations
HandlerDeclaration              ::=  MessageIdentifier
                                     FunctionIdentifier

// Routine declarations
RoutineDeclaration              ::=  FunctionIdentifier?
                                     TypeIdentifier?         // for return value
                                     RoutineParameterDescription*
                                     VariableDeclaration*
                                     OCTET STRING            // program code

RoutineParameterDescription     ::=  RoutinePassingMode?
                                     TypeIdentifier

RoutinePassingMode              ::=  'VALUE' | 'REFERENCE'

// Useful definitions
TypeIdentifier                  ::=  INTEGER
DataIdentifier                  ::=  INTEGER
FunctionIdentifier              ::=  INTEGER
MessageIdentifier               ::=  INTEGER
PackageIdentifier               ::=  INTEGER
```

APÉNDICE II

## Notación textual para guiones MHEG-SIR

Este apéndice describe una notación textual, legible por las personas, para la expresión de guiones MHEG-SIR. Puede ser útil para expresar ejemplos de MHEG-SIR. Se presenta en la notación EBNF.

```
InterchangedScript      ::=  "SCRIPT "
                        TypeDeclaration*
                        ConstantDeclaration*
                        VariableDeclaration*
                        PackageDeclaration*
                        HandlerDeclaration*
                        RoutineDeclaration*
                        "ENDSCRIPT "

// Type declarations
TypeDeclaration         ::=  "TYPE "
                        Identification?
                        TypeDescription
                        "ENDTYPE "

TypeDescription         ::=  "SEQUENCE "  SequenceDescription  "ENDSEQUENCE "
                        |    "STRING "  StringDescription  "ENDSTRING "
                        |    "ARRAY "  ArrayDescription  "ENDARRAY "
                        |    "STRUCT "  StructureDescription  "ENDSTRUCT "
                        |    "UNION "  UnionDescription  "ENDUNION "

SequenceDescription     ::=  INTEGER  // Sequence bound
                        Reference  // Type identifier

StringDescription       ::=  INTEGER?  // String bound

ArrayDescription        ::=  INTEGER  // Array size
                        Reference  // Type identifier

UnionDescription        ::=  Reference+  // Type identifiers

StructureDescription    ::=  Reference+  // Type identifiers

// Data declarations
ConstantDeclaration     ::=  "CONSTANT "
                        Identification?
                        Reference  // Type identifier
                        ConstantValue
                        "ENDCONSTANT "

ConstantValue           ::=  "BOOLEAN "  BOOLEAN
                        |    "OCTET "  OCTET
                        |    "SHORT "  INTEGER
                        |    "LONG "  INTEGER
                        |    "WORD "  INTEGER
                        |    "UNSIGNED "  INTEGER
                        |    "FLOAT "  REAL
                        |    "DOUBLE "  REAL                    |  "CHAR "  STRING
                        |    "STRING "  STRING
                        |    "IDENTIFIER "  Reference  // Data identifier
                        |    "SEQUENCE "  ConstantValue*
                        |    "STRUCT "  ConstantValue*
                        |    "ARRAY "  ConstantValue*
```

```
                    |       "UNION "    UnionValue

UnionValue                  ::=     INTEGER             // Tag index
                            ConstantValue

VariableDeclaration             ::=     "VARIABLE "
                            Identification?
                            Reference           // Type identifier
                            ConstantReference?      // Initial value (constant)
                            "ENDVARIABLE "

ConstantReference               ::=     Reference           // Data identifier
                    |       ConstantValue

// Package declarations
PackageDeclaration              ::=     "PACKAGE "
                            IntegerIdentification?
                            STRING?                 // Package name
                            ServiceDescription*
                            ExceptionDescription*
                            "ENDPACKAGE "

ServiceDescription              ::=     "SERVICE "
                            IntegerIdentification?
                            STRING?                 // IDL operation global name
                            CallingMode?
                            Reference?              // Return type identifier
                            ServiceParameterDescription*
                            "ENDSERVICE "

ServiceParameterDescription             ::=     "PARAM "
                            ServicePassingMode
                            Reference               // Type identifier

CallingMode                 ::=     "SYNC " | "ASYNC "
ServicePassingMode              ::=     "IN " | "OUT " | "INOUT "

ExceptionDescription                    ::=     "EXCEPTION "
                            IntegerIdentification?
                            STRING?                 //IDL exception global name
                            Reference*              // Parameter type identifiers
                            "ENDEXCEPTION "

// Handler declarations
HandlerDeclaration              ::=     "HANDLER "
                            Reference               // Message identifier
                            Reference               // Function identifier
                            "ENDHANDLER "

// Routine declarations
RoutineDeclaration              ::=     "ROUTINE "
                            Identification?
                            Reference?              // Return type identifier
        RoutineParameterDescription*
                            VariableDeclaration*
                            Instruction+
                            "ENDROUTINE "
```

```
RoutineParameterDescription     ::=  "PARAM "
                                     RoutinePassingMode?
                                     Reference                  // Type identifier

RoutinePassingMode              ::=  "VAL " | "VAR "

// Useful definitions
Identification                  ::=  "ID " Reference
IntegerIdentification           ::=  "ID " INTEGER

INTEGER                 ::=  DecimalInteger | HexaInteger
DecimalInteger          ::=  Sign? Digit+ " "
Sign                    ::=  "+" | "-"
Digit                   ::=  "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
HexaInteger             ::=  "h" Hex+
Hex             ::=     Digit | "A" | "B" | "C" | "D" | "E" | "F"
BOOLEAN                 ::=  "TRUE " | "FALSE "
OCTET                   ::=  Hex Hex " "
STRING                  ::=  """ Character* """
Character               ::=  ... // visible character
REAL                    ::=  Sign? Digit+ "." Digit* "e" Sign? Digit+  " "

// Program code
Instruction             ::=  SimpleInstruction " "
                        |    LogicalOperator LogicalTypeCode " "
                        |    "EQ_" TypeCode " "
                        |    ComparisonOperator ComparisonTypeCode " "
                        |    ArithmeticOperator NumericTypeCode  " "
                        |    "NEG_" SignedTypeCode " "
                        |    "REM_" IntegerTypeCode "´"
                        |    "DUP_" TypeCode " "
                        |    "CVT_" ConvertibleTypes " "
                        |    "SHIFT_" UnsignedTypeCode "´" INTEGER
                        |    JumpInstruction " " Destination
                        |    UnaryInstruction " " Reference
                        |    "PUSHI " INTEGER
                        |    BinaryInstruction " " Reference INTEGER
                        |    "LABEL " STRING        // Label in routine code

IntegerTypeCode         ::=  SignedIntegerTypeCode | UnsignedTypeCode
SignedIntegerTypeCode           ::=  "S" | "L"
SignedTypeCode          ::=  SignedIntegerTypeCode | "F" | "D"
UnsignedTypeCode        ::=  "O" | "W" | "U"
LogicalTypeCode         ::=  "B" | UnsignedTypeCode
NumericTypeCode         ::=  SignedTypeCode | UnsignedTypeCode
ComparisonTypeCode      ::=  NumericTypeCode | "C"
ConvertibleTypes        ::=  "SW" | "WS" | "LU" | "UL" | "CW" | "WC" | "BS" | "OS"
                        |    "SL" | "LF" | "WL" | "UF" | "FD" | "BO" | "OW" | "SU"
                        |    "WU" | "OB" | "SB" | "LB" | "WB" | "UB" | "WO" | "LS"
                        |    "FL" | "UW" | "FU" | "DF"
TypeCode                ::=  ComparisonTypeCode | "B" | "I" | "R"

Reference               ::=  INTEGER            // Identifier
                        |    STRING             // Logical name or IDL name
Destination             ::=  INTEGER            // Offset
                        |    STRING             // Label

SimpleInstruction               ::=  "EQR" | "YIELD" | "RET" | "NOP" | "FREE"

ComparisonOperator      ::=  "GT_" |    "LT_"
```

**LogicalOperator** ::= **"AND_" | "OR_" | "XOR_" | "NOT_"**

**ArithmeticOperator** ::= **"ADD_" | "SUB_" | "MUL_" | "DIV_"**

**JumpInstruction** ::= **"JMP" | "LJMP" | "JT" | "JF" | "LJT" | "LJF"**

**UnaryInstruction** ::= **"INC" | "DEC" | "PUSHR" | "PUSH" | "POPR" | "POP" |"POPC"**
**| "GETOR" | "ALLOC" | "CALL" | "XCALL"**

**BinaryInstruction** ::= **"SET" | "SETC" | "GET" | "GETC"**

APÉNDICE III

## Entidades MHEG

Las entidades MHEG incluyen objetos MHEG, objetos mh, objetos rt, objetos MHEG intercambiados, zócalos (*sockets*), canales.

### III.1 Objetos MHEG (*MHEG objects*)

Según la Rec. UIT-T T.172 (e ISO/CEI 13522-1) [5], un objeto MHEG se define como una representación codificada. Por tanto, los objetos MHEG son cadenas de bits. La identidad de un objeto MHEG es su cadena de bits. Los objetos MHEG son objetos de "forma 1", descritos en 6.2.4 de ISO/CEI 13522-1 [5]. El objeto MHEG A y el objeto MHEG B son idénticos si, y únicamente si, **son** la misma secuencia de bits.

Un objeto MHEG no es un objeto físico, sino una abstracción (una secuencia especificada de bits) que puede tener muchas representaciones (esto es, diferentes objetos) de diferentes tipos: objetos MHEG intercambiados, objetos MHEG almacenados, objetos mh, etc. Estas representaciones son tratadas por diferentes servicios informatizados.

Un objeto MHEG puede identificarse por un identificador MHEG. Los identificadores MHEG son la **única** manera de identificar objetos MHEG. La estructura y la representación codificada de los identificadores MHEG se define en la Rec. UIT-T T.172 (e ISO/CEI 13522-1) [5]. El identificador MHEG de un objeto MHEG tiene que estar codificado dentro del objeto MHEG. Como el atributo es facultativo, algunos objetos MHEG no tienen un identificador MHEG. Estos objetos no pueden ser identificados. La Rec. UIT-T T.172 (e ISO/CEI 13522-1) [5] impone una constricción al diseño de aplicaciones MHEG: el objeto MHEG A y el objeto MHEG B no tendrán el mismo identificador MHEG a menos que sean idénticos.

La referencia genérica MHEG describe todas las formas posibles de hacer referencia (referenciar) un objeto MHEG.

### III.2 Objetos mh (*mh-objects*)

Un objeto mh es una representación interna de un objeto MHEG en un proceso o sistema. Un objeto mh no es un objeto MHEG. Dentro de un motor MHEG, objetos mh representan objetos MHEG "disponibles" ("*available*"). Los objetos mh son objetos de "forma 2", descritos en 6.2.4 de ISO/CEI 13522-1 [5]. Un objeto mh representa un objeto MHEG, es decir, siempre hay una cadena de bits que corresponde a un objeto mh. Un motor MHEG no debe tratar más de un objeto mh para representar un objeto MHEG.

En consecuencia, los objetos mh tratados por motores MHEG pueden identificarse utilizando identificadores MHEG. Además, la aplicación puede definir otros mecanismos para identificar objetos mh (por ejemplo, una identificación simbólica), a condición de que su representación interna

lo permita. Esto es especialmente útil cuando algunos de los objetos MHEG representados por objetos mh de un motor MHEG no son identificables, es decir, no tienen un identificador MHEG. Esto permite garantizar que todos los objetos mh son identificables.

Los objetos mh son referenciados de la misma manera que los objetos MHEG. Las referencias a objetos MHEG para las cuales el motor MHEG trata un objeto mh se resolverá generalmente direccionando este objeto mh.

### III.3 Objetos rt (*rt-objects*)

Un objeto rt es una "instancia" (o copia), en la fase de ejecución (*run-time*) de un objeto mh "modelo", que es creado y tratado por un motor MHEG con fines de presentación. Un objeto rt no es un objeto MHEG. Dentro de un motor MHEG, los objetos rt representan objetos MHEG "disponibles en la fase de ejecución" ("*rt-available*"). Los objetos rt son objetos de "forma 3", descritos en 6.2.4 de ISO/CEI 13522-1 [5]. Puede haber ninguno o varios objetos rt que sean copias "presentables" de un objeto mh. Un objeto rt siempre tiene exactamente un objeto mh como su modelo.

Los objetos rt pueden identificarse utilizando identificadores de objeto rt cuyo componente "identificación de objeto modelo" es un identificador MHEG. La estructura y la representación codificada de los identificadores de objeto rt se define en la Rec. UIT-T T.172 (e ISO/CEI 13522-1) [5]. Además, la aplicación puede definir otros mecanismos para identificar objetos rt (por ejemplo, una identificación simbólica), a condición de que su representación interna lo permita. Esto es especialmente útil cuando algunos de los objetos MHEG representados por objetos mh de un motor MHEG utilizados como modelos para objetos rt no son identificables, es decir, no tienen un identificador MHEG. Esto permite garantizar que todos los objetos rt son identificables.

Los objetos rt pueden ser referenciados mediante referencias genéricas MHEG.

### III.4 Objetos MHEG intercambiados (*interchanged MHEG objects*)

Los objetos MHEG intercambiados son representaciones de objetos MHEG que están siendo comunicados en un instante dado mediante una red o un dispositivo de almacenamiento. Un objeto dado (por ejemplo, una cadena de bits) puede ser intercambiado muchas veces entre muchos lugares, es decir, puede ser representado por muchos objetos MHEG intercambiados. Un identificador MHEG externo puede identificar un objeto MHEG intercambiado y, por tanto, hacer referencia a un objeto MHEG por medio de su ubicación y hora de intercambio. Debe señalarse, no obstante, que un identificador MHEG externo no tiene necesariamente que identificar un objeto MHEG.

Los objetos MHEG almacenados son representaciones de objetos MHEG situados usualmente en ficheros, o en registros de bases de datos. Por ejemplo, un objeto MHEG dado (por ejemplo, una cadena de bits) puede estar almacenada en muchos lugares, es decir, puede estar representada por muchos objetos MHEG almacenados. Esas ubicaciones usualmente se identifican mediante nombres de fichero o identificadores de base de datos. Un identificador MHEG externo puede identificar una ubicación de almacenamiento para un objeto MHEG, y por tanto hacer referencia a un objeto MHEG por medio de su ubicación de almacenamiento.

APÉNDICE IV

## Características principales de MHEG-SIR

Esta Recomendación se ha elaborado para responder a varios requisitos y constricciones relacionados con:

– las aplicaciones que la utilizan;

– la funcionalidad que proporciona;

– su contexto de utilización por las aplicaciones;

– su funcionamiento.

La MHEG-SIR está dotada de las características descritas en este apéndice.

### IV.1 Características de las aplicaciones que utilizan la MHEG-SIR

La MHEG-SIR se adapta a las exigencias de las aplicaciones que emplean:

1) manipulación de entidades MHEG;

2) cálculos, tratamiento de variables y estructuras de control;

3) control de dispositivos externos;

4) adquisición de datos;

5) acceso a datos externos;

6) acceso a cualesquiera servicios externos en la fase de ejecución.

#### IV.1.1 Manipulación de entidades MHEG

Las aplicaciones suelen manipular entidades MHEG para fines de la presentación multimedios.

Esta característica se consigue mediante el mecanismo descrito en IV.2.2.

#### IV.1.2 Cálculos, tratamiento de variables y estructuras de control

Las aplicaciones necesitan estas características de procesamiento de datos para obtener un comportamiento dinámico avanzado.

Esta característica se consigue mediante el mecanismo descrito en IV.2.1.

#### IV.1.3 Control de dispositivos externos

Los dispositivos a menudo proporcionan una interfaz específica de la plataforma. Los componentes de la plataforma de ejecución que aplican otra Recomendación y sus normas monomedio encapsuladas usualmente soportan cierto número de dispositivos. Además, algunas aplicaciones necesitan controlar directamente, sea estos dispositivos, sea otros dispositivos no soportados de otra manera.

Esta característica se consigue mediante el mecanismo descrito en IV.2.2.

NOTA – En este contexto, un gestor de dispositivo (*device driver*) es un lote de un servicio que debe ser proporcionado por el entorno de ejecución.

#### IV.1.4 Adquisición de datos

Los componentes de la plataforma de ejecución que aplican otra Recomendación y sus normas monomedio encapsuladas usualmente soportan cierto número de dispositivos. Además, algunas aplicaciones necesitan controlar directamente la adquisición de datos (por ejemplo para una sintonización más fina), sea mediante estos mecanismos, sea mediante otros mecanismos no soportados de otra manera.

Esta característica se consigue mediante el mecanismo descrito en IV.2.2.

NOTA – En este contexto, los datos adquiridos pueden ser extraídos por un guión rt por medio de una reacción asíncrona a mensajes de notificación enviados por el gestor de un dispositivo externo de adquisición.

### IV.1.5 Acceso a datos externos

Algunas aplicaciones necesitan ganar acceso a datos que no son de MHEG, que deben ser extraídos de un almacenamiento local, de un tren (*stream*), o de un depositario de datos distante, utilizando un componente de comunicación del entorno de ejecución.

Esta característica se consigue mediante el mecanismo descrito en IV.2.2.

NOTA – En este contexto, los datos externos pueden extraerse mediante un componente del sistema que es un paquete de un servicio proporcionado por el entorno de ejecución.

### IV.1.6 Acceso a cualesquiera servicios externos en la fase de ejecución

Por cualesquiera servicios en la fase de ejecución ha de entenderse todos los servicios cuya definición de interfaz no es conocida por la implementación antes de que la aplicación es intercambiada.

Esta característica se consigue mediante el mecanismo descrito en IV.2.2.

NOTA – En este contexto, la capacidad de cálculo externo puede proporcionarla una biblioteca o proceso que es un lote de un servicio proporcionado por el entorno de ejecución.

## IV.2 Características funcionales

La MHEG-SIR se utiliza para expresar:

1) operaciones de procesamiento de datos (véase IV.2.1);
2) acceso a datos y funciones externos (véase IV.2.2).

### IV.2.1 Operaciones de procesamiento de datos

Para el soporte de operaciones de procesamiento de datos, la MHEG-SIR proporciona mecanismos utilizados por guiones intercambiados para expresar:

– la estructura de tipos de datos numéricos construidos y avanzados;
– variables y valores de estos tipos;
– instrucciones que proporcionan el acceso a datos o la asignación de variables;
– instrucciones que afectan al flujo de control de la ejecución del guión;
– instrucciones que proporcionan operadores aritméticos, lógicos y de comparación.

Estos mecanismos se describen en las cláusulas 8, 12 y 13.

### IV.2.2 Acceso a datos y funciones externos

El acceso a datos y funciones externos requiere la cooperación entre varios componentes del motor MHEG-3 y el entorno de ejecución, para invocar funciones, enviar y recibir mensajes e intercambiar datos.

Para el soporte de la manipulación de entidades MHEG (es decir, del acceso a datos y funciones MHEG), la MHEG-SIR proporciona mecanismos utilizados por guiones intercambiados para:

– invocar acciones elementales MHEG;
– responder a acciones MHEG dirigidas al guión rt;
– expresar variables y valores de tipos de datos MHEG.

Estos mecanismos se describen en la cláusula 11. Se utilizan también para expresar intercambio de datos y sincronización entre el guión rt y otros objetos rt, o entre guiones rt.

Para el soporte de la cooperación con el entorno de ejecución (es decir, del acceso a datos y funciones que no son MHEG), la MHEG-SIR proporciona mecanismos utilizados por guiones intercambiados para:

– declarar la estructura de lotes proporcionados por el entorno de ejecución;
– invocar servicios proporcionados por el entorno de ejecución;
– reaccionar a mensajes enviados por el entorno de ejecución;
– declarar tipos de datos numéricos construidos y avanzados;
– expresar variables y valores de estos tipos.

Estos mecanismos se describen en la cláusula 10. Expresan también el intercambio de datos complejos y sincronización entre el intérprete de guión MHEG-SIR y procesos que forman parte del entorno de ejecución.

Además, esta Recomendación define la correspondencia de las declaraciones de lotes MHEG-SIR con componentes del entorno de ejecución real. Esto se hace en dos pasos:

– Expresión y utilización de una especificación de interfaz abstracta por la MHEG-SIR; éste es el mecanismo de correspondencia IDL descrito en la cláusula 14. Consiste en disposiciones para guiones intercambiados de MHEG-3.
– Correspondencia entre una especificación de interfaz abstracta y su implementación en el entorno de ejecución de un tipo de plataforma; ésta es la plantilla de especificación de correspondencia de plataforma descrita en el anexo D. Consiste en disposiciones para implementaciones de MHEG-3.

### IV.3 Características técnicas

La MHEG-SIR satisface los siguientes requisitos técnicos:

1) independencia con respecto al soporte físico;
2) representación en forma final;
3) compacidad;
4) facilidad de implementación;
5) eficiencia de interpretación;
6) calidad de abierta y extensibilidad;
7) no revisabilidad;
8) disposiciones para intercambio en tiempo real;
9) validación semántica para fines de calidad de servicio;
10) posibilidad de verificación de la sintaxis (para evitar riesgos de contaminación);
11) representación no privada (no propietaria);
12) procesamiento securizado de guiones.

#### IV.3.1 Independencia con respecto al soporte físico

La independencia de la MHEG-SIR con respecto al soporte físico, y por tanto la portabilidad de los guiones intercambiados, se obtiene mediante la definición de un código de máquina virtual para expresar guión intercambiados y la definición de una maquina virtual para interpretar este código.

Sólo se utilizan datos tipificados. No se establecen requisitos en cuanto a la manera en que los datos habrán de ser representados o tratados internamente por motores MHEG-3.

La representación codificada se basa en reglas de codificación ASN.1, que son independientes del soporte físico.

Las declaraciones de interfaz se basan en una correspondencia con especificaciones de interfaz abstracta que pueden expresarse utilizando IDL, que es independiente del soporte físico.

La capacidad de un componente del entorno de ejecución de una plataforma dada para interoperar con cualquier motor MHEG-3 conforme, en esta plataforma, se garantiza mediante la utilización de la especificación de correspondencia de plataforma.

La capacidad de una implementación de motor MHEG-3, en una plataforma dada, de interoperar con cualquier proveedor de servicio en el entorno de ejecución se garantiza mediante la utilización de la especificación de correspondencia de plataforma, a condición de que dichos proveedores sean conformes con esta especificación.

### IV.3.2 Representación en forma final

La representación en forma final de guiones intercambiados se obtiene mediante:

– la utilización de ASN.1 para la codificación de guiones intercambiados;

– una codificación de máquina virtual que está semánticamente cercana a una amplia clase de computadores de uso corriente;

– una arquitectura de máquina de pila que emplea una eficiente codificación de instrucciones en un modo de direccionamiento implícito;

– una ordenación de las declaraciones que reduce la tara que entraña el procesamiento de referencias hacia adelante;

– la secuenciación apropiada de instrucciones dentro de las rutinas.

### IV.3.3 Compacidad

La compacidad de la representación codificada de guiones intercambiados se consigue mediante varias optimizaciones:

– la definición de un código de máquina virtual basado en pila permite que las instrucciones tengan un número reducido de operandos, o ninguno; la instrucción MHEG-SIR más larga se codifica en cuatro octetos;

– se utiliza una codificación de trenes de octetos para el código de rutinas, con lo que se evita la tara introducida por la codificación TLV (tipo, longitud, valor);

– en la codificación de las rutinas, las instrucción están empacadas ("*packed*"), es decir, no tienen octetos de relleno;

– se utilizan constantes para la declaración de valores inmediatos;

– se emplean códigos predefinidos para tipos, operaciones y mensajes MHEG;

– se utiliza la declaración de una tabla de definiciones de manejadores para optimizar la expresión de la correspondencia entre mensajes dirigidos al guión y rutinas previstas para tratar estos mensajes.

### IV.3.4 Facilidad de implementación

La facilidad de implementación de intérpretes de guiones MHEG-SIR se consigue mediante:

– la definición de un conjunto reducido de instrucciones;

– la definición clara de una máquina virtual;

– el número limitado de conceptos e identificadores que los intérpretes de guiones deben tratar;
– la definición formal de semánticas de instrucción.

### IV.3.5 Eficiencia de la interpretación

La eficiencia de la interpretación, por intérpretes de guiones MHEG-SIR, de guiones intercambiados se consigue mediante:

– la utilización de un código de máquina virtual basada en pila;
– la utilización de instrucciones de bajo nivel;
– la utilización una representación en forma final.

### IV.3.6 Calidad de abierta y extensibilidad

La calidad de abierta y la extensibilidad de la MHEG-SIR se consigue mediante:

– una definición genérica de las interfaces a que se puede ganar acceso desde el código de guión MHEG-SIR;
– la capacidad para ganar acceso a objetos MHEG y para invocar rutinas desde otro guión rt;
– la posibilidad de añadir nuevas instrucciones a la representación sin modificar la estructura de guiones intercambiados.

### IV.3.7 No revisabilidad

La no revisabilidad se consigue mediante la utilización de una representación en forma final, de bajo nivel. Esta representación está prevista para que la produzcan herramientas informáticas especializadas y no permite un fácil regreso al código de fuente original, como en el caso de las representaciones diseñadas por personas utilizando lenguajes de guión y/o entornos autorizantes; de este modo se limita el riesgo de una alteración indebida de la semántica del programa.

### IV.3.8 Disposiciones para intercambio en tiempo real

La MHEG-SIR se adapta al marco de MHEG, cuya estructura general ha sido concebida para satisfacer los requisitos del intercambio en tiempo real.

La sintaxis de guiones intercambiados está definida con el fin de optimizar las posibilidades de tratar "al vuelo" ("*on the fly*") las declaraciones contenidas.

Además la utilización de la codificación ASN.1 permite detectar errores (en cierta medida) cuando el intercambio de guión se está efectuando en entornos de red caracterizados por altos niveles de ruido.

### IV.3.9 Validación semántica para fines de calidad de servicio

Debido a los requisitos impuestos por esta Recomendación a la semántica, se puede probar el comportamiento de un guión intercambiado, con el fin de validar su calidad de funcionamiento antes de usarlo efectivamente en el contexto de un servicio comercial. Pueden construirse intérpretes de guión MHEG-SIR para que sirvan de referencia en cuanto a la forma en que los motores MHEG-3 deben comportarse cuando interpreten los guiones intercambiados sometidos a prueba.

### IV.3.10 Posibilidad de comprobación de la sintaxis (para evitar riesgos de contaminación)

La definición formal de la sintaxis de la MHEG-SIR puede utilizarse para verificar su corrección y, de ese modo, evitar el intercambio de fragmentos de código que contengan, por ejemplo, virus, introducidos con la intención de causar daño, de alguna manera, al sistema receptor. Las implementaciones pueden decidir efectuar comprobaciones sintácticas y semánticas en la fase de ejecución y en la fase de carga.

NOTA – El diseño de MHEG-SIR como una representación interpretada, independiente de la máquina, reduce el riesgo de contaminación. Puede existir un riesgo residual proveniente de implementaciones no certificadas, o por otra razón incorrectas. La provisión de mecanismos de encripción, autenticación y de otros mecanismos de seguridad en el nivel de transporte está fuera del ámbito de esta Recomendación.

### IV.3.11 Representación (no privada) no propietaria

Esta Recomendación sigue la política de derechos de propiedad intelectual de los organismos internacionales de normalización. Para más detalles, sírvase referirse a la declaración sobre derechos de propiedad intelectual que figura al principio de esta Recomendación.

### IV.3.12 Procesamiento securizado de los guiones

Un diseñador de sistema puede desear asegurarse de que la ejecución, intencional o accidental, de un guión defectuoso, tendrá una repercusión mínima en el sistema de entrega, y que todas las operaciones de acceso a servicios externos puedan ser cuidadosamente vigiladas. La máquina virtual MHEG-SIR incluye varias características que permiten alcanzar este objetivo:

– interfaces explícitas, intensamente tipificadas, con todos los servicios externos;
– un conjunto de instrucciones intensamente tipificadas, de modo que todas las operaciones y operandos puedan ser verificados, o bien por un procesamiento previo, o bien durante la ejecución;
– no hay direccionamiento directo de la memoria (es decir, ni se utiliza una aritmética de punteros), con lo que se elimina la posibilidad de efectos marginales no deseados;
– aislamiento del contexto para cada objeto de guión rt;
– aislamiento de los contextos definidos por cada trama de llamada;
– no hay manipulación directa de asas, identificadores de tipos, ni identificadores de datos.

## 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 |