Unión Internacional de Telecomunicaciones

UIT-T Z.143

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

(03/2006)

SERIE Z: LENGUAJES Y ASPECTOS GENERALES DE SOPORTE LÓGICO PARA SISTEMAS DE TELECOMUNICACIÓN

Técnicas de descripción formal – Notación de prueba y de control de prueba

Notación de pruebas y de control de pruebas versión 3: Semántica operacional

Recomendación UIT-T Z.143

RECOMENDACIONES UIT-T DE LA SERIE Z

LENGUAJES Y ASPECTOS GENERALES DE SOPORTE LÓGICO PARA SISTEMAS DE TELECOMUNICACIÓN

TÉCNICAS DE DESCRIPCIÓN FORMAL	
Lenguaje de especificación y descripción	Z.100–Z.109
Aplicación de técnicas de descripción formal	Z.110–Z.119
Gráficos de secuencias de mensajes	Z.120–Z.129
Lenguaje ampliado de definición de objetos	Z.130–Z.139
Notación de prueba y de control de prueba	**Z.140–Z.149**
Notación de requisitos de usuarios	Z.150–Z.159
LENGUAJES DE PROGRAMACIÓN	
CHILL: el lenguaje de alto nivel del UIT-T	Z.200–Z.209
LENGUAJE HOMBRE-MÁQUINA	
Principios generales	Z.300–Z.309
Sintaxis básica y procedimientos de diálogo	Z.310–Z.319
LHM ampliado para terminales con pantalla de visualización	Z.320–Z.329
Especificación de la interfaz hombre-máquina	Z.330–Z.349
Interfaces hombre-máquina orientadas a datos	Z.350–Z.359
Interfaces hombre-máquina para la gestión de las redes de telecomunicaciones	Z.360–Z.379
CALIDAD	
Calidad de soportes lógicos de telecomunicaciones	Z.400–Z.409
Aspectos de la calidad de las Recomendaciones relativas a los protocolos	Z.450–Z.459
MÉTODOS	
Métodos para validación y pruebas	Z.500–Z.519
SOPORTE INTERMEDIO	
Entorno del procesamiento distribuido	Z.600–Z.609

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

Recomendación UIT-T Z.143

Notación de pruebas y de control de pruebas versión 3: Semántica operacional

Resumen

En esta nueva Recomendación se define la semántica operacional de la notación de pruebas y de control de pruebas 3 (TTCN-3). La semántica operacional es necesaria para interpretar inequívocamente las especificaciones hechas con TTCN-3. La presente Recomendación utiliza el lenguaje núcleo definido en la Rec. UIT-T Z.140.

Orígenes

La Recomendación UIT-T Z.143 fue aprobada el 16 de marzo de 2006 por la Comisión de Estudio 17 (2005-2008) del UIT-T por el procedimiento de la Recomendación UIT-T A.8.

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

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

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.

La observancia de esta Recomendación es voluntaria. Ahora bien, la Recomendación puede contener ciertas disposiciones obligatorias (para asegurar, por ejemplo, la aplicabilidad o la interoperabilidad), por lo que la observancia se consigue con el cumplimiento exacto y puntual de todas las disposiciones obligatorias. La obligatoriedad de un elemento preceptivo o requisito se expresa mediante las frases "tener que, haber de, hay que + infinitivo" o el verbo principal en tiempo futuro simple de mandato, en modo afirmativo o negativo. El hecho de que se utilice esta formulación no entraña que la observancia se imponga a ninguna de las partes.

PROPIEDAD INTELECTUAL

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

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

ÍNDICE

Página

Recomendación UIT-T Z.143

Notación de pruebas y de control de pruebas versión 3: Semántica operacional

1 Alcance

En esta Recomendación se define la semántica operacional de TTCN-3. La Recomendación utiliza el lenguaje núcleo definido en la Recomendación UIT-T Z.140 [1].

2 Referencias

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. En esta Recomendación, la referencia a un documento, en tanto que autónomo, no le otorga el rango de una Recomendación.

[1] Recomendación UIT-T Z.140 (2006), *Notación de pruebas y de control de pruebas versión 3: Lenguaje núcleo*.

3 Definiciones, abreviaturas, siglas o acrónimos

3.1 Definiciones

A los efectos de esta Recomendación, se aplican los términos y las definiciones de la Rec. UIT-T Z.140 [1].

3.2 Abreviaturas, siglas o acrónimos

En esta Recomendación se utilizan las siguientes abreviaturas, siglas o acrónimos.

ASN.1 Notación de sintaxis abstracta uno (*abstract syntax notation one*)

BNF Forma Backus-Nauer (*Backus-Nauer form*)

IDL Lenguaje de definición de interfaz (*interface definition language*)

MTC Componente de prueba principal (*main test component*)

SUT Sistema sometido a prueba (*system under test*)

TTCN Notación de pruebas y de control de pruebas (*testing and test control notation*)

4 Introducción

En esta cláusula se define intuitiva e inequívocamente el significado del comportamiento de TTCN-3. No se pretende que la semántica operacional sea formal, y por lo tanto la posibilidad de efectuar pruebas matemáticas utilizando esta semántica está limitada.

Esta semántica operacional proporciona una panorámica de los estados de ejecución de los módulos de TTCN. Se presentan diversos tipos de estado y se describe el significado de las diferentes construcciones de TTCN-3:

1) utilizando la información del estado para definir las condiciones necesarias previas a la ejecución de la construcción; y
2) definiendo la forma en que cambia el estado como resultado de la ejecución de una construcción.

La semántica operacional se limita al significado del comportamiento en TTCN-3, es decir, a funciones, alternativas de operaciones (altsteps), casos de prueba, control del módulo y construcciones del lenguaje para definir el comportamiento de prueba, como por ejemplo, las operaciones **`send`** y **`receive`** y las instrucciones **`if-else`**- o **`while`**-. Se explica el significado de algunas de las construcciones de TTCN-3 sustituyéndolas por construcciones de otros lenguajes. Por ejemplo, como las instrucciones **`interleave`** son formas simplificadas de una secuencia de

instrucciones **alt** anidadas, se explica el significado de casa instrucción **interleave** sustituyéndola por la secuencia correspondiente de instrucciones **alt** anidadas.

En la mayoría de los casos, la definición de la semántica del lenguaje utiliza árboles de la sintaxis abstracta del código que ha de describirse. La presente semántica no utiliza árboles de sintaxis abstracta, sino representaciones gráficas de descripciones de comportamiento de TTCN-3 en la forma de diagramas de flujo. Los diagramas de flujo describen el flujo del control en una función, un altstep, un caso de prueba, o el control del módulo. La correspondencia de las descripciones de comportamiento de la TTCN-3 a diagramas de flujo es directa.

NOTA – La correspondencia de instrucciones TTCN-3 con diagramas de flujo es un paso informal y no está definida en la Rec. UIT-T Z.140 [1] usando reglas de la BNF. Esto se debe a que las reglas de la BNF no son óptimas para aplicar correspondencia intuitiva, pues varias de las reglas semánticas estáticas se codifican en reglas BNF a fin de permitir verificaciones de semántica estática durante la verificación de la semántica.

5 Estructura de la presente Recomendación

Esta Recomendación está compuesta por cuatro partes:

1) En la primera parte (véase la cláusula 6) se describen las restricciones de la semántica operacional, es decir, los aspectos relacionados con la semántica, que no se tratan en la presente Recomendación.

2) En la segunda parte (véase la cláusula 7) se define el significado de notaciones macro y notaciones simplificadas de la TTCN-3, mediante su reemplazo por otras construcciones de lenguaje de la TTCN-3. Se puede considerar que estos reemplazos en un módulo TTCN-3 son un paso obligatorio previo a la interpretación del módulo conforme a la descripción de la semántica operacional del paso siguiente.

3) En la tercera parte (véase la cláusula 8) se describe la semántica operacional de TTCN-3 mediante la interpretación de los diagramas de flujo y la modificación del estado.

4) En la cuarta parte (véase la cláusula 9) se especifica la traducción de las diversas instrucciones TTCN-3 a segmentos de diagramas de flujo, que son los componentes fundamentales para los diagramas de flujo que representan funciones, etapas alternativas (altsteps), casos de prueba y control del módulo.

6 Restricciones

La semántica operacional cubre únicamente los aspectos comportamentales de TTCN-3, es decir, describe el significado de las instrucciones y de las operaciones. No proporciona:

a) Una semántica para los aspectos relacionados con los datos de la TTCN-3, incluidos aspectos como la codificación, la decodificación y el uso de datos importados de especificaciones diferentes a la TTCN-3.

b) Una semántica para el mecanismo de agrupamiento. El agrupamiento tiene que ver con la parte de definiciones en un módulo TTCN-3 y no conlleva aspectos comportamentales.

c) Una semántica para la instrucción **import**. La importación de definiciones debe llevarse a cabo en la parte de definiciones de los módulos TTCN-3. La semántica operacional trata las definiciones importadas como si estas se definiesen en el módulo de importación.

d) Una semántica para la parametrización de puertos.

7 Sustitución de las formas simplificadas

Antes de que pueda utilizarse esta semántica operacional para explicar el comportamiento de TTCN-3, se deben expandir las formas simplificadas presentando las correspondientes definiciones completas de forma literal.

Las formas simplificadas de TTCN-3 son:

- listas de declaraciones de parámetros de los módulos, de constantes y de variables de un mismo tipo y listas de declaraciones de temporizadores;
- operaciones de recepción autónomas;
- solicitudes de altstep autónomas;
- operaciones **trigger**;
- omisión de instrucciones **return** y **stop** al final de definiciones de funciones o de casos de prueba;
- omisión de instrucciones **stop**; e
- instrucciones **interleave**.

Además del procesamiento de las formas simplificadas, la semántica operacional exige que también se procesen los parámetros de los módulos y las constantes globales, es decir, las constantes definidas en la parte de definiciones del módulo. Han de reemplazarse por valores concretos todas las referencias a los parámetros de los módulos y a las constantes globales. Esto implica que se supone que se pueden determinar los valores de los parámetros de los módulos y de las constantes globales antes de que la semántica operacional tome pertinencia.

NOTA 1 – El procesamiento de los parámetros de los módulos y las constantes globales en la semántica operacional es diferente a su procesamiento en un compilador de TTCN-3. La semántica operacional describe el significado del comportamiento de la TTCN-3 y no constituye una orientación para la creación de un compilador de TTCN-3.

NOTA 2 – La semántica operacional trata las constantes locales de los componentes de prueba, casos de prueba, funciones y controles de módulo, así como sus parámetros, como si fueran variables. Se debe verificar estáticamente el uso erróneo de las constantes locales y de los parámetros **in**, **out** e **inout**.

7.1 Orden de los pasos de sustitución

El siguiente es el orden en que se debe efectuar la sustitución literal de formas simplificadas, constantes globales y parámetros de los módulos:

1) sustitución de listas de declaraciones de parámetros de los módulos, constantes, variables y temporizadores, por instrucciones individuales;
2) sustitución de constantes globales y parámetros de los módulos, por valores concretos;
3) incorporación de operaciones de recepción autónomas en instrucciones **alt**;
4) incorporación de solicitudes altstep autónomas en instrucciones **alt**;
5) expansión de las instrucciones **interleave**;
6) sustitución de todas las operaciones **trigger** por operaciones **receive** e instrucciones **repeat** equivalentes;
7) introducción de **return** al final de las funciones sin instrucción **return**, introducción de operaciones **self.stop** al final de las definiciones testcase sin instrucción stop.
8) introducción de **stop** al final de la parte de control de módulo sin instrucción stop.

NOTA – Si los pasos no se efectúan en este orden, el resultado de las sustituciones no representará el comportamiento definido.

7.2 Sustitución de constantes globales y parámetros de módulo

Las constantes declaradas en la parte de definiciones del módulo son globales para el control del módulo y para todos los componentes de prueba que se creen durante la ejecución del módulo TTCN-3. Se pretende que los parámetros de módulo sean constantes globales durante la ejecución.

Han de sustituirse todas las referencias hechas a constantes globales y a parámetros del módulo por sus valores reales, antes de que la semántica operacional inicie la interpretación del módulo. Si el valor de una constante o de un parámetro de módulo está dado en la forma de una expresión, debe calcularse la expresión. El resultado del cálculo deberá luego sustituir todas las referencias que se hayan hecho a la constante o al parámetro del módulo.

7.3 Incorporación de operaciones de recepción individuales en instrucciones alt

Las operaciones de recepción de TTCN-3 son: **receive**, **trigger**, **getcall**, **getreply**, **catch**, **check**, **timeout** y **done**.

NOTA – Las operaciones **receive**, **trigger**, **getcall**, **getreply**, **catch** y **check** actúan sobre puertos y permiten la ramificación ocasionada por la recepción de mensajes, solicitudes de procedimiento, respuestas y excepciones. Las operaciones **timeout** y **done** no son verdaderas operaciones de recepción, pero pueden utilizarse de la misma forma que las operaciones de recepción, es decir, como alternativas en instrucciones **alt**. Por lo tanto, la semántica operacional procesará **timeout** y **done** de la misma forma que las operaciones de recepción.

Las operaciones de recepción pueden usarse como instrucciones autónomas en funciones, altstep y casos de prueba. La operación **timeout** también puede usarse como instrucción autónoma en el control del módulo. En ese caso se considera que la operación de recepción es una simplificación de una instrucción **alt** con una sola alternativa, que es definida por la operación de recepción. Desde el punto de vista de la semántica operacional, una instrucción **alt** en la que se haya incorporado la instrucción de recepción, sustituirá todas las operaciones de recepción autónomas existentes.

EJEMPLO:

```
// Cuando aparezca
   :
  MyCL.trigger(MyType:?);
   :
```

```
// se sustituirá por
    :
   alt {
      []   MyCL.trigger (MyType:?) { }
   }
    :

// además
    :
   MyPTC.done;
    :

// se sustituirá por
    :
   alt {
      []   MyPTC.done { }
   }
    :
```

7.4 Incorporación de solicitudes autónomas altstep en instrucciones alt

TTCN-3 permite hacer solicitudes de altstep como funciones dentro de funciones, altsteps, casos de prueba y controles de módulo. El significado de una solicitud altstep viene dado por una instrucción **alt** con una sola rama, que solicita el altstep. La instrucción **alt** debe determinar la situación instantánea que se evalúa dentro del altstep e invocar el mecanismo por defecto si no se puede elegir ninguna de las alternativas del altstep.

NOTA – Los altstep utilizados en el control del módulo pueden incluir únicamente alternativas con operaciones de **timeout** y una rama **else**.

EJEMPLO:

```
// Cuando aparezca
    :
   myAltstep(MyPar1Val);
    :

// se sustituirá por
    :
   alt {
      []   myAltstep(MyPar1Val) { }
   }
    :
```

7.5 Sustitución de instrucciones interleave

Se define el significado de una instrucción **interleave** sustituyéndola por un conjunto de instrucciones **alt** anidadas con el mismo significado. En la presente cláusula se describe el algoritmo para efectuar la sustitución de una instrucción **interleave**. La sustitución se hará a nivel sintáctico.

En las instrucciones **interleave** no se permite:

1) Utilizar las instrucciones de transferencia de control: **for**, **while**, **do-while**, **goto**, **activate**, **deactivate**, **stop**, **repeat** y **return**;
2) solicitar altsteps;
3) solicitar funciones definidas por el usuario que incluyan operaciones de comunicaciones;
4) utilizar expresiones booleanas para controlar el acceso a ramas de la instrucción **interleave**; y
5) especificar ramas **else**.

Como consecuencia de estas restricciones, las instrucciones autónomas que no se mencionan (por ejemplo, assignment, **log**, **send** y **reply**), las operaciones **call** *bloqueantes* y las instrucciones compuestas **interleave**, **if-else** y **alt**, pueden utilizarse en instrucciones **interleave**.

NOTA 1 – Si no incluyen instrucciones **alt**, las operaciones **call** bloqueantes y las instrucciones **if-else** pueden procesarse como si fuesen instrucciones autónomas. En caso de incluirse instrucciones **alt**, las alternativas aportan a la instrucción **interleave** y requieren un procesamiento especial. En aras de simplicidad, el algoritmo que se presenta a continuación no diferencia entre los dos casos.

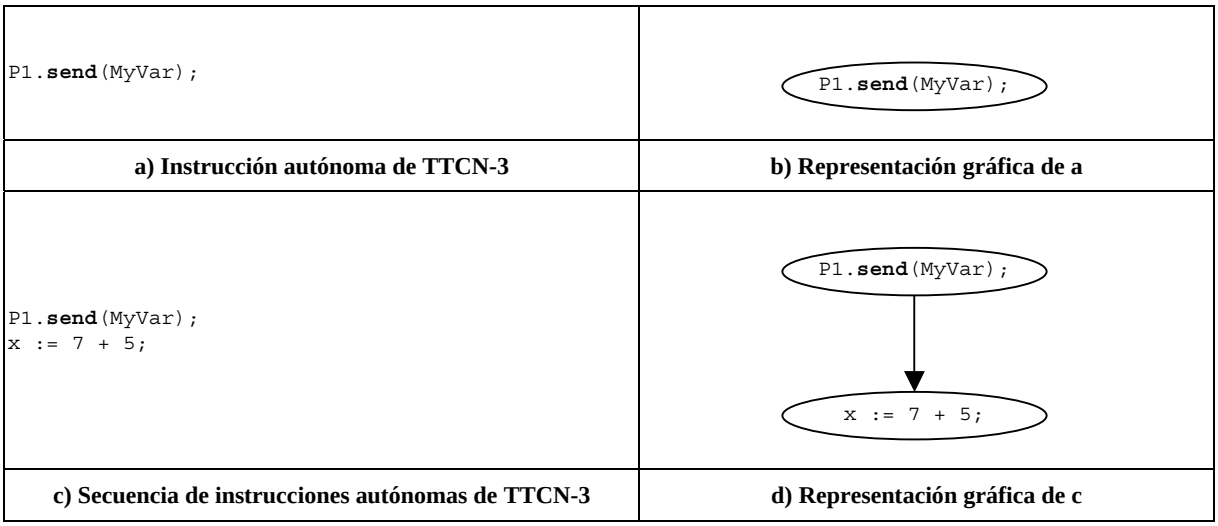
NOTA 2 – También se permiten operaciones **call** no bloqueantes en instrucciones interleave. Se les considera instrucciones autónomas.

El algoritmo que se describe en esta cláusula sirve únicamente para instrucciones **interleave** sin instrucciones **interleave** anidadas. En caso de haber instrucciones **interleave** con instrucciones **interleave** anidadas, deben sustituirse las instrucciones **interleave** anidadas antes de que pueda aplicarse el algoritmo.

NOTA 3 – Gracias a las restricciones 1 a 5, siempre será posible encontrar sustituciones de instrucciones **interleave** anidadas.

El algoritmo de sustitución funciona para la interpretación gráfica de una instrucción interleave, transformándola en una estructura en árbol, semánticamente equivalente, que describe un conjunto de instrucciones **alt** anidadas. Con este objetivo, es necesario contar con una representación gráfica de las instrucciones autónomas, de las instrucciones compuestas **if-else**, de **call** *bloqueante*, de **alt** y de **interleave**.

Las instrucciones autónomas se describen mediante un nodo marcado con el nombre de la instrucción. Las secuencias de instrucciones autónomas se describen mediante un conjunto de nodos conectados con líneas de flujo, como se muestra en la figura 1.



a) Instrucción autónoma de TTCN-3	b) Representación gráfica de a
c) Secuencia de instrucciones autónomas de TTCN-3	d) Representación gráfica de c

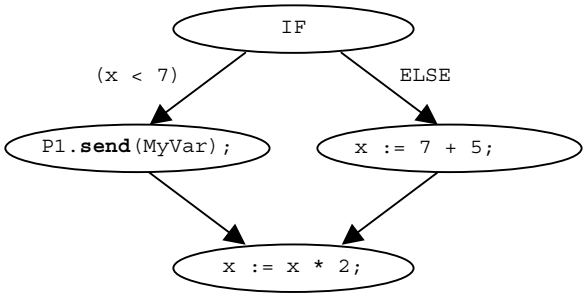
Figura 1/Z.143 – Representación gráfica de instrucciones autónomas de TTCN-3

En la figura 2 se muestra la representación gráfica de una instrucción **if-else**. Las instrucciones **if-else** se representan mediante un nodo IF con dos líneas de flujo que se conectan a la primera instrucción de cada una de las dos alternativas. Las instrucciones **if-else** sin rama ELSE se representan de la misma forma si hay instrucciones después de la instrucción **if-else**. En este caso, la línea de flujo que representa la rama *else* se conecta a la primera instrucción después de la instrucción **if-else**. Las instrucciones **if-else** sin rama ELSE que no estén seguidas por otras instrucciones, se representan mediante un nodo IF con una sola línea de flujo.

NOTA 4 – En la figura 1 se incluyen los nombres de las líneas de flujo únicamente con el fin de facilitar la lectura. El algoritmo utiliza únicamente la relación expresada por la línea de flujo y no la inscripción.

<table>
<tr>
<td>

```
if (x < 7) {
    P1.send(MyVar);
}
else {
    x := 7 + 5;
}
x := x * 2;
```

</td>
<td> 
</td>
</tr>
<tr>
<td>a) Instrucción if-else de TTCN-3</td>
<td>b) Representación gráfica de a</td>
</tr>
<tr>
<td>

```
if (x < 7) {
    P1.send(MyVar);
}
x := x * 2;
```

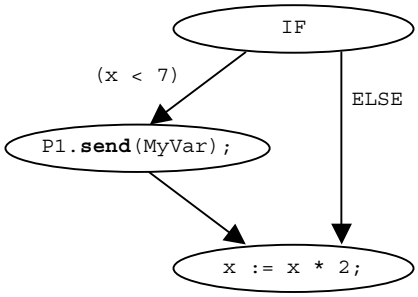
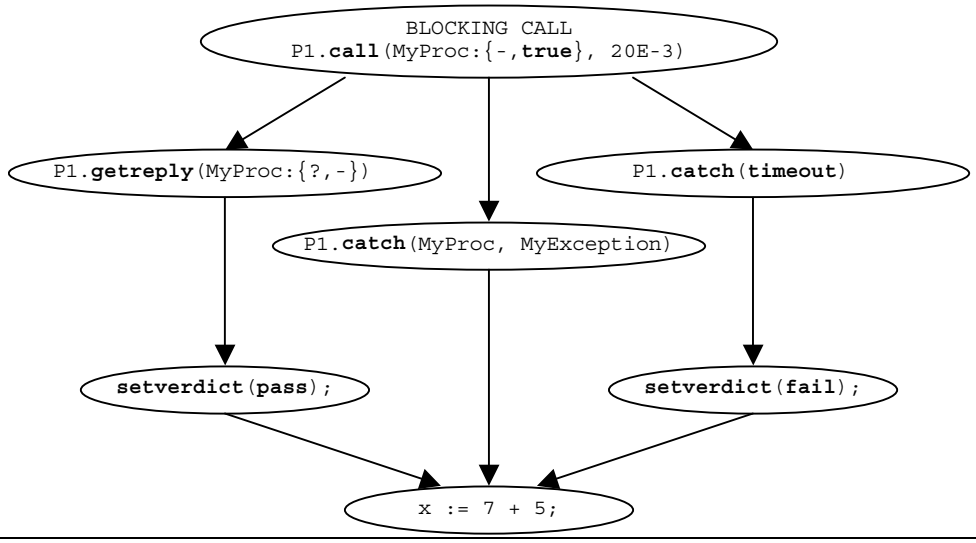
</td>
<td> 
</td>
</tr>
<tr>
<td>c) Instrucción if-else de TTCN-3 sin rama else</td>
<td>d) Representación gráfica de c</td>
</tr>
</table>

Figura 2/Z.143 – Representación gráfica de instrucciones if-else de TTCN-3

En la figura 3 se muestra la representación gráfica de una instrucción **`call`** bloqueante. Las instrucciones **`call`** bloqueantes se representan mediante un nodo BLOCKING-CALL con líneas de flujo conectadas a las instrucciones **`getreply`** y **`catch`** de las diversas alternativas.

```
P1.call (MyProc:{-, true}, 20E-3) {
   [] P1.getreply(MyProc:{?,-} {
         setverdict(pass);
      }
   [] P1.catch(MyProc, MyException) {}
   [] P1.catch(timeout) {
         setverdict(fail);
      }
}
x := 7 + 5;
```

a) Instrucción call bloqueante de TTCN-3



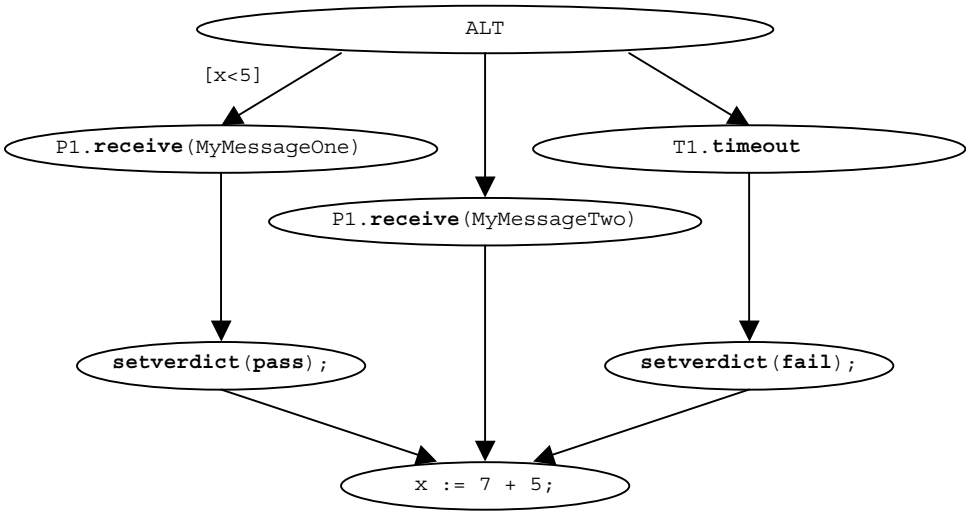
b) Representación gráfica de a

Figura 3/Z.143 – Representación gráfica de una instrucción call bloqueante de TTCN-3

En la figura 4 se muestra la representación gráfica de una instrucción **`alt`**. Las instrucciones **`alt`** se representan mediante un nodo *alt* con líneas de flujo conectadas a las diversas alternativas.

```
alt {
   [x<5] P1.receive(MyMessageOne} {
         setverdict(pass);
      }
   [] P1.receive(MyMessageTwo) {}
   [] T1.timeout {
         setverdict(fail);
      }
}
x := 7 + 5;
```

a) Instrucción alt de TTCN-3



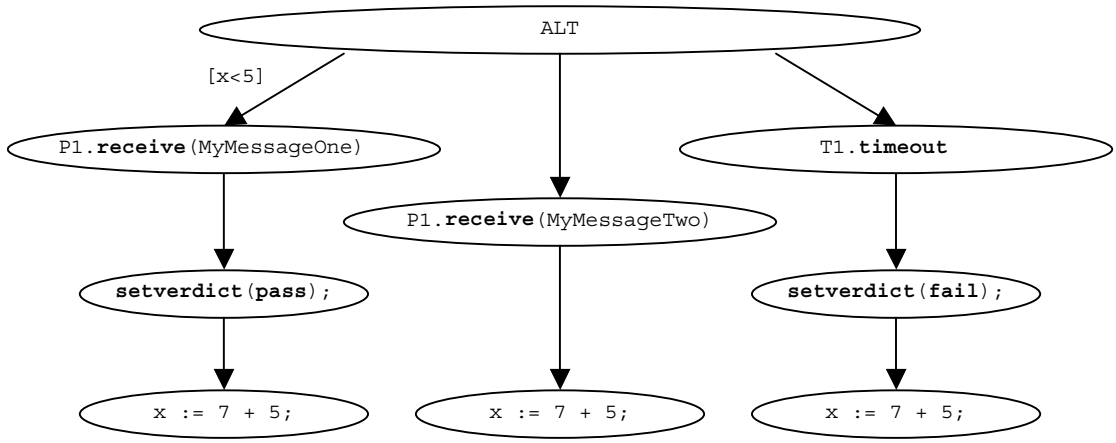
b) Representación gráfica de a

Figura 4/Z.143 – Representación gráfica de una instrucción alt de TTCN-3

Por lo general, las representaciones gráficas de las instrucciones **`if-else`**, **`call`** *bloqueante* y **`alt`** son gráficos dirigidos sin bucles, en los que las líneas de flujo de las diversas alternativas se unen al salir de la instrucción. Es posible transformar estos gráficos dirigidos en representaciones en árbol equivalentes, utilizando la duplicación. Esto se ilustra en la figura 5 para el caso de la instrucción alt de la figura 4. Estas representaciones en árbol se construyen mediante el algoritmo que se describe más adelante.

```
Alt {
   [x<5] P1.receive(MyMessageOne} {
         setverdict(pass);
         x:= 7 + 5;
      }
   [] P1.receive(MyMessageTwo) {
         x:= 7 + 5;
      }
   [] T1.timeout {
         setverdict(fail);
         x:= 7 + 5;
      }
}
```

a) Instrucción alt de TTCN-3 semánticamente equivalente a la figura 4-a



b) Representación gráfica de a semánticamente equivalente a la figura 4-b

Figura 5/Z.143 – Representación gráfica de una instrucción alt de TTCN-3

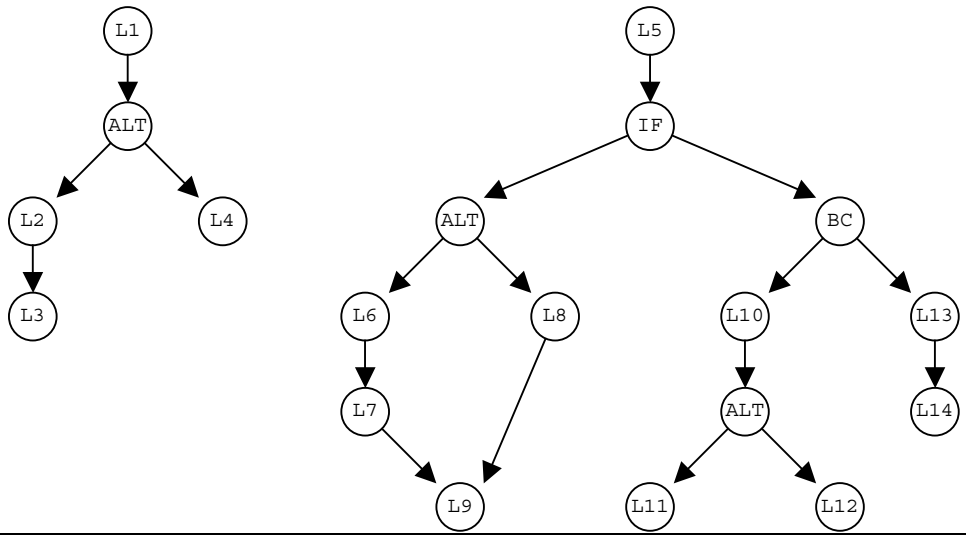
Pueden describirse las instrucciones **`interleave`** mediante un gráfico compuesto por un conjunto de subgráficos dirigidos, cada uno de los cuales se construye con instrucciones autónomas y las instrucciones compuestas **`if-else`**, **`call`** *bloqueante* y **`alt`**. Los subgráficos dirigidos describen los flujos de control entrelazados. En la figura 6 se muestra un ejemplo. Los nombres de los nodos de la figura 6-b se refieren a las etiquetas de las instrucciones de TTCN-3 en la figura 6-a.

```
interleave {
   [] P1.receive(M1} {                                  // L1
         alt {                                             // ALT
            [] P1.receive(M3) {                          // L2
                  setverdict(pass);                      // L3
               }
            [] T1.timeout { }                               // L4
         }

      }
   [] P2.receive(M2) {                                  // L5
         if (x < 5 ) {                                     // IF
            alt {                                        // ALT
               [] P2.receive(M4) {                       // L6
                  setverdict(pass);                      // L7
                  }
               [] Comp1.done { }                         // L8
            }
            x := 7 + 5;                                     // L9
         }
         else {
            P3.call(MyProcTempl, 20E-3) {    // BC (= SOLICITUD BLOQUEANTE (BLOCKING CALL))

               [] P3.getreply(ReplyTempl) {              // L10
                     alt {                                   // ALT
                        [] P2.receive(M5)  { }           // L11
                        [] P2.receive(M6)  { }           // L12
                     }
                  }
               [] P3.catch(timeout) {                    // L13
                     setverdict(fail);                   // L14
                  }
            }
         }
      }
}
```

a) Instrucción interleave de TTCN-3



b) Representación gráfica de a

Figura 6/Z.143 – Representación gráfica de una instrucción interleave de TTCN-3

Se pueden escribir formalmente las instrucciones **`interleave`** mediante un gráfico ***GI*** = (***St***, ***F***) donde:

St es el conjunto de instrucciones permitidas de TTCN-3; y

$\boldsymbol{F} \subseteq (\boldsymbol{St} \times \boldsymbol{St})$ describe la relación de flujo.

El término *instrucciones permitidas de TTCN-3* hace referencia a las restricciones estáticas 1 a 5 que se presentaron anteriormente.

Es necesario definir las siguientes funciones para el algoritmo de construcción:

- La función *REACHABLE* devuelve todas las instrucciones asequibles desde una instrucción ***s*** en un gráfico ***GI*** = (***St***, ***F***):

  REACHABLE (***s***, ***GI***) = { ***s*** } ∪
  { ***stmt*** | ***stmt*** ∈ ***St*** ∧ ∃(***s*** = $x_1, x_2, \ldots, x_n$ = ***stmt***) donde $x_i \in$ ***St***,
  i ∈ {1…n} ∧ (x_i, x_{i+1}) ∈ ***F***}

- La función *SUCCESSORS* devuelve todos los sucesores a la instrucción ***s*** en un gráfico ***GI*** = (***St***, ***F***):

  SUCCESSORS (**s**, **GI**) = { **stmt** | **stmt** ∈ **St** ∧ (**s**, **stmt**) ∈ **F**}

- La función *ENABLED* devuelve todas las instrucciones de un gráfico ***GI*** = (***St***, ***F***) que no tengan predecesor:

  ENABLED (***GI***) = { ***stmt*** | ***stmt*** ∈ ***St*** ∧ (***F*** ∩ (***S*** × {***s***}) = ∅)}

- La función *KIND* devuelve el tipo de una instrucción TTCN-3 en un gráfico que representa una instrucción **`interleave`**.

- La función *DISCARD* borra una instrucción ***s*** o un conjunto de instrucciones ***S*** en un gráfico ***GI*** = (***St***, ***F***) y devuelve el gráfico resultante ***GI'***= (***St'***, ***F'***):

  Para nodos independientes:

  DISCARD (***s***, ***GI***) = ***GI'*** donde ***GI'*** = (***St'***, ***F'***), con ***St'*** = ***St***\{***s***} y
  F' = ***F*** ∩ (***St***\{***s***} × ***St***\{***s***}).

  Para conjuntos de nodos:

  DISCARD (***S***, ***GI***) = ***GI'*** donde: ***GI'*** = (***St'***, ***F'***), con ***St'*** = ***St******S*** y ***F'*** = ***F*** ∩ (***St******S*** × ***St******S***).

- La función *RECEIVING* toma un conjunto de instrucciones de un gráfico ***GI*** y devuelve todas las instrucciones de recepción:

  RECEIVING (*S*) = { *stmt* | *stmt* ∈ *St* ∧ *KIND*(*stmt*) ∈ {receive, trigger, getcall, getreply, catch, check, done, timeout}}

- La función *RANDOM* elige aleatoriamente un elemento ***s*** de un conjunto dado ***S*** y devuelve ***s***.

  RANDOM (***S***) = ***s*** donde ***s*** ∈ ***S***

El algoritmo de construcción (véase la figura 7) del árbol consiste en un procedimiento iterativo en el que en cada iteración se construyen los nodos sucesores de un nodo dado. Se presenta el procedimiento utilizando una notación de seudocódigo similar al lenguaje C, que utiliza las funciones definidas anteriormente, junto con alguna notación matemática adicional.

```
CONSTRUCT-SUCCESSORS (statementType *predecessor, graphType GI) {
    // - statementType se refiere al tipo de nodo del árbol en construcción
    // - *predecessor se refiere al último nodo creado
    // - graphType indica el tipo de gráfico de las instrucciones TTCN-3
    // - GI se solicita por valor y se refiere al subgráfico que contiene las instrucciones TTCN-3
    // restantes que se deben considerar

    var graphType myGraph;
    var statementType i, myStmt;
    var statementType *newStmt, *firstInBranch; // apuntadores a los nuevos nodos de instrucciones
                                                // del árbol que se está construyendo de forma
                                                // recursiva

    // Recuperación de conjuntos de instrucciones TTCN-3 sin predecesores en 'GI'
    var statementSet enabStmts:= ENABLED(GI);           // todas las instrucciones sin predecesor
    var statementSet enabRecStmts:= RECEIVING(enabStmts);   // instrucciones de recepción
                                                            // en 'enabStmts'
    var statementSet enabNonRecStmts:= enabStmts\enabRecStmts;
                                        // instrucciones en 'enabStmts' que no son de recepción

    if (GI.St == ∅) {   // Se supone que GI.St se refiere al conjunto de instrucciones en GI
        return;         // No hay más instrucciones, criterio para la finalización de la iteración
    }
    elseif (enabNonRecStmts != ∅) { // Procesamiento de instrucciones de 'enabStmts' que
                                    // no son de recepción

        myStmt:= RANDOM(enabNonRecStmts);
                // Sólo puede haber una instrucción en 'enabNonRec', ya que el algoritmo
                // continúa la construcción hasta hallar una rama que aporte a la
                // instrucción interlave.
        newStmt:= create(myStmt, predecessor);
                // Creación de un nuevo nodo del árbol, que representa a 'myStmt' en el árbol
                // y actualización de los apuntadores en 'newStmt' y en 'predecessor'.

        if (KIND(myStmt) == IF || KIND(myStmt) == BLOCKING_CALL) {
            for each i in SUCCESSORS(myStmt, GI) {

                firstInBranch:= create(i, newStmt);
                        // Creación de un segundo nodo para la primera instrucción de una rama,
                        // debido a una instrucción if-else.
                        // Nota, se utilizará esta instrucción create para crear nodos del árbol
                        // que representan las instrucciones de recepción en operaciones de
                        // solicitudes bloqueantes.

                myGraph:= DISCARD({i, myStmt} ∪ REACHABLE(myStmt, GI)\REACHABLE(i, GI))
                        // Supresión de i, myStmt y todas las instrucciones asequibles desde
                        // myStmt pero no asequibles desde i. Esto tiene en cuenta la ramificación
                        // de los controles de flujo en subgráficos de GI.

                CONSTRUCT-SUCCESSORS(firstInBranch, myGraph);   // SIGUIENTE PASO DE LA ITERACIÓN
            }
        }
        elseif (KIND(myStmt) == ALT) {
            for each (i in SUCCESSORS(myStmt, GI) {

            CONSTRUCT-SUCCESSORS(mystmt, DISCARD(REACHABLE(myStmt, GI)\REACHABLE(i, GI)));
                // SIGUIENTE PASO DE LA ITERACIÓN, el argumento DISCARD(REACHABLE(myStmt,
                // GI)\REACHABLE(i, GI))
                        // tiene en cuenta la ramificación de los controles de flujo ocasionada
                        // por diversos eventos de recepción.
            }
        }
        else {      // mystmt es una instrucción autónoma
            CONSTRUCT-SUCCESSORS(newSonNode, DISCARD(myStmt, GI));
                    // SIGUIENTE PASO DE LA ITERACIÓN
        }
    }
    else {  // Tratamiento de eventos de recepción que se intercalan
        if (KIND(predecessor) != ALT) { // falta un nodo alt y debe crearse si la acción de
                                // intercalar no se ve afectada por ninguna instrucción alt anidada
            predecessor:= create(ALT, predecessor);
        }

        for each i in enabRecStmts) {
            newStmt:= create(i, predecessor);               // Nuevo nodo del árbol
            CONSTRUCT-SUCCESSORS(newStmt, DISCARD(i, GI));  // SIGUIENTE(S) PASO(S) DE LA ITERACIÓN
        }
    }
}
```

Figura 7/Z.143 – Algoritmo de sustitución de instrucciones interleave de TTCN-3

Inicialmente la función CONSTRUCT-SUCCESSORS (véase la figura 7) se solicita con el *nodo raíz* de un árbol vacío y el gráfico de instrucciones de TTCN-3 que describen la instrucción **`interleave`** que se ha de sustituir. Tras la finalización, se puede utilizar el *nodo raíz* para acceder al árbol que se construyó.

El uso de la función CONSTRUCT-SUCCESSORS con la instrucción **`interleave`** de la figura 6, genera el árbol que se muestra en la figura 8. Las etiquetas se refieren a las instrucciones de la figura 6-a. Las etiquetas múltiples son consecuencia de la duplicación de código. En la figura 9 se muestra el código de TTCN-3 correspondiente al árbol de la figura 8.

NOTA 5 – El ejemplo de utilización del algoritmo de la figura 7 es muy completo (véanse las figuras 6, 8 y 9). Se presenta este ejemplo para ilustrar la mayoría de las situaciones especiales, es decir, ramificación y unión de líneas de flujo, una instrucción **`alt`** incorporada, una instrucción **`call`** *bloqueante* y una instrucción **`if-else`**.

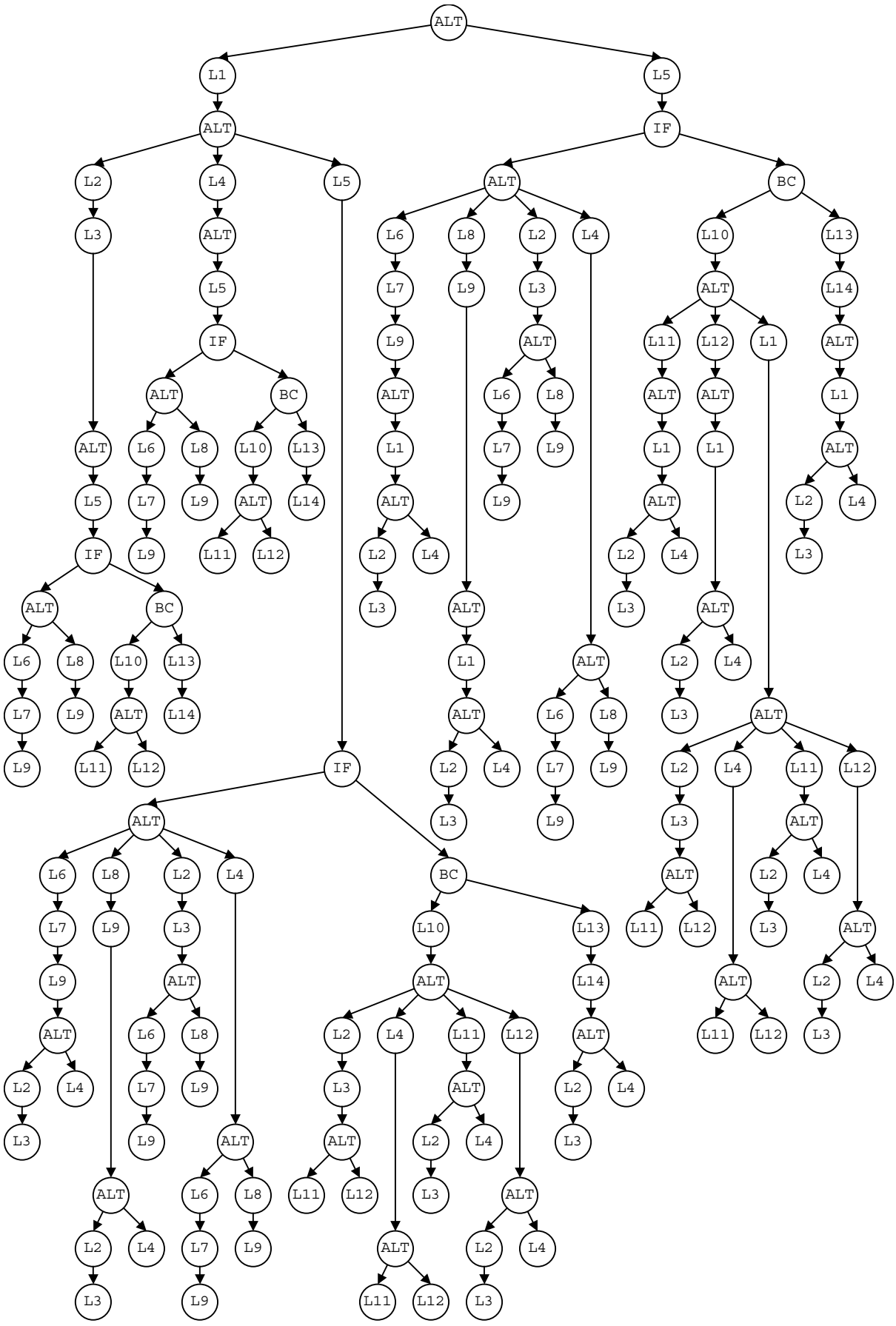


Figura 8/Z.143 – Resultado de aplicar el algoritmo de la figura 7 a la instrucción interleave de la figura 6

```
alt {                                                                                   // ALT
    []  P1.receive(M1} {                                                                // L1
            alt {                                                                       // ALT
                []  P1.receive(M3) {                                                    // L2
                        setverdict(pass);                                               // L3
                        alt {                                                           // ALT
                            []  P2.receive(M2) {                                        // L5
                                    if (x < 5 ) {                                       // IF
                                        alt {                                           // ALT
                                                []  P2.receive(M4) {                    // L6
                                                    setverdict(pass);                   // L7
                                                    x:= 7 + 5;                          // L9
                                                }
                                                []  Comp1.done {                        // L8
                                                    x:= 7 + 5;                          // L9
                                    }   }           }
                                    else {
                                        P3.call(MyProcTempl, 20E-3) {//BC(= BLOCKING CALL)
                                            []  P3.getreply(ReplyTempl) {               // L10
                                                alt {                                   // ALT
                                                    []  P2.receive(M5) { }              // L11
                                                    []  P2.receive(M6) { }              // L12
                                                }   }
                                            []  P3.catch(timeout) {                     // L13
                                                setverdict(fail);                       // L14
                    }   }       }   }   }       }
                []  T1.timeout {                                                        // L4
                        alt {                                                           // ALT
                            []  P2.receive(M2) {                                        // L5
                                    if (x < 5 ) {                                       // IF
                                        alt {                                           // ALT
                                                []  P2.receive(M4) {                    // L6
                                                    setverdict(pass);                   // L7
                                                    x:= 7 + 5;                          // L9
                                                }
                                                []  Comp1.done {                        // L8
                                                    x:= 7 + 5;                          // L9
                                    }   }           }
                                    else {
                                        P3.call(MyProcTempl, 20E-3) {// BC = BLOCKING CALL)
                                            []  P3.getreply(ReplyTempl) {               // L10
                                                alt {                                   // ALT
                                                    []  P2.receive(M5) { }              // L11
                                                    []  P2.receive(M6) { }              // L12
                                                }   }
                                            []  P3.catch(timeout) {                     // L13
                                                setverdict(fail);                       // L14
                    }   }       }   }   }       }
                []  P2.receive(M2) {                                                    // L5
                        if (x < 5 ) {                                                   // IF
                            alt {                                                       // ALT
                                []  P2.receive(M4) {                                    // L6
                                    setverdict(pass);                                   // L7
                                    x:= 7 + 5;                                          // L9
                                    alt {                                               // ALT
                                        []  P1.receive(M3) {                            // L2
                                            setverdict(pass);                           // L3
                                        }
                                        []  T1.timeout { }                              // L4
                                }   }
                                []  Comp1.done {                                        // L8
                                    x:= 7 + 5;                                          // L9
                                    alt {                                               // ALT
                                        []  P1.receive(M3) {                            // L2
                                            setverdict(pass);                           // L3
                                        }
                                        []  T1.timeout { }                              // L4
                                }   }
                                []  P1.receive(M3) {                                    // L2
                                    setverdict(pass);                                   // L3
                                    alt {                                               // ALT
                                        []  P2.receive(M4) {                            // L6
                                            setverdict(pass);                           // L7
                                            x:= 7 + 5;                                  // L9
                                        }
                                        []  Comp1.done {                                // L8
                                            x:= 7 + 5;                                  // L9
                                }   }       }
```

```
                                []  T1.timeout {                                        // L4
                                        alt {                                           // ALT
                                            []  P2.receive(M4) {                        // L6
                                                    setverdict(pass);                   // L7
                                                    x:= 7 + 5;                          // L9
                                                }
                                            []  Comp1.done {                            // L8
                                                    x:= 7 + 5;                          // L9
                        }   }       }   }       }
                        else {
                            P3.call(MyProcTempl, 20E-3) {           // BC (= BLOCKING CALL)
                                []  P3.getreply(ReplyTempl) {                           // L10
                                        alt {                                           // ALT
                                            []  P1.receive(M3) {                        // L2
                                                    setverdict(pass);                   // L3
                                                    alt {                               // ALT
                                                        []  P2.receive(M5) { }          // L11
                                                        []  P2.receive(M6) { }          // L12
                                                    }   }
                                            []  T1.timeout {                            // L4
                                                    alt {                               // ALT
                                                        []  P2.receive(M5) { }          // L11
                                                        []  P2.receive(M6) { }          // L12
                                                    }   }
                                            []  P2.receive(M5) {                        // L11
                                                    alt {                               // ALT
                                                        []  P1.receive(M3) {            // L2
                                                                setverdict(pass);       // L3
                                                            }
                                                        []  T1.timeout { }              // L4
                                                    }   }
                                            []  P2.receive(M6) {                        // L12
                                                    alt {                               // ALT
                                                        []  P1.receive(M3) {            // L2
                                                                setverdict(pass);       // L3
                                                            }
                                                        []  T1.timeout { }              // L4
                                        }   }       }   }
                                []  P3.catch(timeout) {                                 // L13
                                        setverdict(fail);                               // L14
                                        alt {                                           // ALT
                                            []  P1.receive(M3) {                        // L2
                                                    setverdict(pass);                   // L3
                                                }
                                            []  T1.timeout { }                          // L4
    }   }       }   }   }       }   }
[]  P2.receive(M2) {                                                                    // L5
        if (x < 5 ) {                                                                   // IF
            alt {                                                                       // ALT
                []  P2.receive(M4) {                                                    // L6
                        setverdict(pass);                                               // L7
                        x:= 7 + 5;                                                      // L9
                        alt {                                                           // ALT
                            []  P1.receive(M1} {                                        // L1
                                    alt {                                               // ALT
                                        []  P1.receive(M3) {                            // L2
                                                setverdict(pass);                       // L3
                                            }
                                        []  T1.timeout { }                              // L4
                        }   }       }   }
                []  Comp1.done {                                                        // L8
                        x:= 7 + 5;                                                      // L9
                        alt {                                                           // ALT
                            []  P1.receive(M1} {                                        // L1
                                    alt {                                               // ALT
                                        []  P1.receive(M3) {                            // L2
                                                setverdict(pass);                       // L3
                                            }
                                        []  T1.timeout { }                              // L4
                        }   }       }   }
                []  P1.receive(M3) {                                                    // L2
                        setverdict(pass);                                               // L3
                        alt {                                                           // ALT
                            []  P2.receive(M4) {                                        // L6
                                    setverdict(pass);                                   // L7
                                    x:= 7 + 5;                                          // L9
                                }
                            []  Comp1.done {                                            // L8
                                    x:= 7 + 5;                                          // L9
                        }   }       }
```

```
                    []  T1.timeout {                                                    // L4
                        alt {                                                           // ALT
                            []  P2.receive(M4) {                                        // L6
                                setverdict(pass);                                       // L7
                                x:= 7 + 5;                                              // L9
                            }
                            []  Comp1.done {                                            // L8
                                x:= 7 + 5;                                              // L9
          }   }       }   }       }
          else {
              P3.call(MyProcTempl, 20E-3) {                        // BC (= BLOCKING CALL)
                  []  P3.getreply(ReplyTempl) {                                         // L10
                      alt {                                                             // ALT
                          []  P2.receive(M5) {                                          // L11
                              alt {                                                     // ALT
                                  []  P1.receive(M1} {                                  // L1
                                      alt {                                             // ALT
                                          []  P1.receive(M3) {                          // L2
                                              setverdict(pass);                         // L3
                                          }
                                          []  T1.timeout { }                            // L4
                              }   }       }   }
                          []  P2.receive(M6) {                                          // L12
                              alt {                                                     // ALT
                                  []  P1.receive(M1} {                                  // L1
                                      alt {                                             // ALT
                                          []  P1.receive(M3) {                          // L2
                                              setverdict(pass);                         // L3
                                          }
                                          []  T1.timeout { }                            // L4
                              }   }       }   }
                          []  P1.receive(M1} {                                          // L1
                              alt {                                                     // ALT
                                  []  P1.receive(M3) {                                  // L2
                                      setverdict(pass);                                 // L3
                                      alt {                                             // ALT
                                          []  P2.receive(M5) { }                        // L11
                                          []  P2.receive(M6) { }                        // L12
                                      }   }
                                  []  T1.timeout {                                      // L4
                                      alt {                                             // ALT
                                          []  P2.receive(M5) { }                        // L11
                                          []  P2.receive(M6) { }                        // L12
                                      }   }
                                  []  P2.receive(M5) {                                  // L11
                                      alt {                                             // ALT
                                          []  P1.receive(M3) {                          // L2
                                              setverdict(pass);                         // L3
                                          }
                                          []  T1.timeout { }                            // L4
                                      }   }
                                  []  P2.receive(M6) {                                  // L12
                                      alt {                                             // ALT
                                          []  P1.receive(M3) {                          // L2
                                              setverdict(pass);                         // L3
                                          }
                                          []  T1.timeout { }                            // L4
                      }   }       }   }       }   }
                  []  P3.catch(timeout) {                                               // L13
                      setverdict(fail);                                                 // L14
                      alt {                                                             // ALT
                          []  P1.receive(M1} {                                          // L1
                              alt {                                                     // ALT
                                  []  P1.receive(M3) {                                  // L2
                                      setverdict(pass);                                 // L3
                                  }
                                  []  T1.timeout { }                                    // L4
}       }   }   }       }   }       }   }
```

Figura 9/Z.143 – Código de TTCN-3 semánticamente equivalente a la instrucción interleave de la figura 6

7.6 Sustitución de operaciones de validación

La operación **trigger** utiliza ciertos criterios de concordancia para filtrar mensajes de un tren de mensajes en un puerto dado. Se puede describir la semántica de la operación **trigger** sustituyéndola con dos operaciones **receive** y una instrucción **goto**. La semántica operacional supone que esta sustitución se hace a nivel sintáctico.

EJEMPLO 1:

```
// La siguiente operación de validación ...

   alt {
   []  MyCL.trigger (MyType:?) { }
   }

// se sustituirá por ...

   alt {
       []   MyCL.receive (MyType:?) { }
       []   MyCL.receive {
            repeat
       }
   }
```

Si la instrucción **trigger** se utiliza en una instrucción **alt** más compleja, la sustitución se lleva a cabo de la misma manera.

EJEMPLO 2:

```
// La siguiente instrucción alt incluye una instrucción de validación ...

   alt {
       []   PCO2.receive {
            stop;
       }
       []   MyCL.trigger (MyType:?) { }
       []   PCO3.catch {
            setverdict(fail);
            stop;
      }
  }

// que puede sustituirse por

   alt {
       []   PCO2.receive {
            stop;
       }
       []   MyCL.receive (MyType:?) { }
       []   MyCL.receive {
            repeat;
       }
       []   PCO3.catch {
            setverdict(fail);
            stop;
       }
  }
```

8 Semántica de los diagramas de flujo de TTCN-3

La semántica operacional de TTCN-3 se fundamenta en la interpretación de diagramas de flujo. En esta cláusula se presentan los diagramas de flujo (véase 8.1); en 8.2 se explica la elaboración de diagramas de flujo para representar el control de los módulos TTCN-3, casos de prueba , altstep, funciones y definiciones de tipo de componente; en 8.3 se definen los estados de los módulos y de los componentes usados para describir los estados de ejecución de los módulos TTCN-3; en 8.4 se describe el tratamiento de mensajes, solicitudes de procedimiento a distancia, repuestas a solicitudes de procedimiento a distancia y excepciones, mientras que en 8.6 se explica el procedimiento de evaluación del control del módulo y de los casos de prueba.

8.1 Diagramas de flujo

Los diagramas de flujo son gráficos dirigidos compuestos por nodos con etiqueta y bordes con etiqueta. El recorrido a través del diagrama de flujo describe uno de los posibles flujos de control al ejecutarse la descripción de un comportamiento descrito.

8.1.1 Marco del diagrama de flujo

El diagrama de flujo se encuadra dentro de un marco que define el límite del diagrama de flujo. El nombre del diagrama de flujo va precedido por las palabras clave **diagrama de flujo** (estas no son palabras clave del lenguaje núcleo de TTCN-3) y se ubica en el ángulo superior izquierdo del diagrama de flujo. Por convenio, se supone que el nombre del diagrama de flujo se refiere a la descripción del comportamiento representada por el diagrama de flujo. En la figura 10 se presenta un diagrama de flujo sencillo.

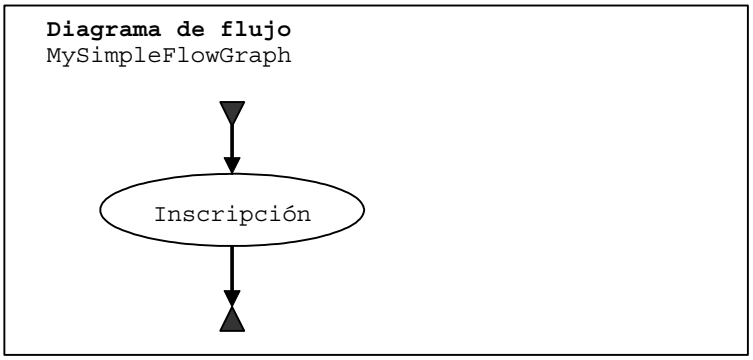


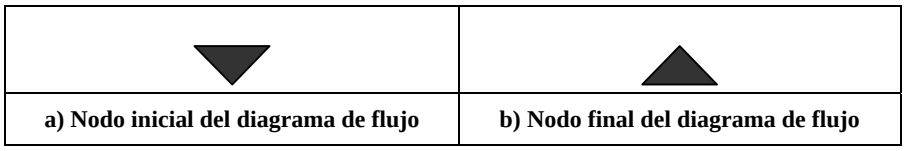
Figura 10/Z.143 – Un diagrama de flujo sencillo

8.1.2 Nodos de los diagramas de flujo

Los diagramas de flujo están compuestos por *nodos iniciales*, *nodos finales*, *nodos básicos* y *nodos de referencia*.

8.1.2.1 Nodos iniciales

El nodo inicial indica el punto de partida del diagrama de flujo. Cada diagrama de flujo contendrá únicamente un nodo inicial. En la figura 11-a se presenta un nodo inicial.



a) Nodo inicial del diagrama de flujo	b) Nodo final del diagrama de flujo

Figura 11/Z.143 – Nodos inicial y final

8.1.2.2 Nodos finales

Los nodos finales indican los puntos de finalización del diagrama de flujo. Cada diagrama de flujo puede contener varios nodos finales, y en el caso de bucles, podría no contener ningún nodo final. Los nodos básicos (véase 8.1.2.3) y los nodos de referencia (véase 8.1.2.4) sin nodos sucesores, se conectarán a un nodo final a fin de indicar que describen la última acción en un trayecto del diagrama de flujo. En la figura 11-b se muestra un nodo final.

8.1.2.3 Nodos básicos

Un nodo básico describe una unidad de ejecución, es decir que se ejecuta en un solo paso. Cada nodo básico tiene un tipo y, dependiendo del tipo, podría tener una lista asociada de atributos. En la figura 12 se muestran dos nodos básicos.

En la inscripción del nodo básico, los atributos del nodo se escriben entre paréntesis después del tipo de nodo. Se emplean el tipo y los atributos para determinar la acción a realizarse durante la ejecución de la construcción de lenguaje que es representada. Los atributos describen la información que ha de recuperase de la correspondiente construcción de TTCN-3.

Los atributos tienen valores y la semántica operacional recuperará estos valores mediante referencia al nombre del atributo. Se permite que, de ser necesario, se asignen valores explícitos en los nodos básicos, utilizando el símbolo de asignación, ':='. En la figura 12-b se muestra un ejemplo.

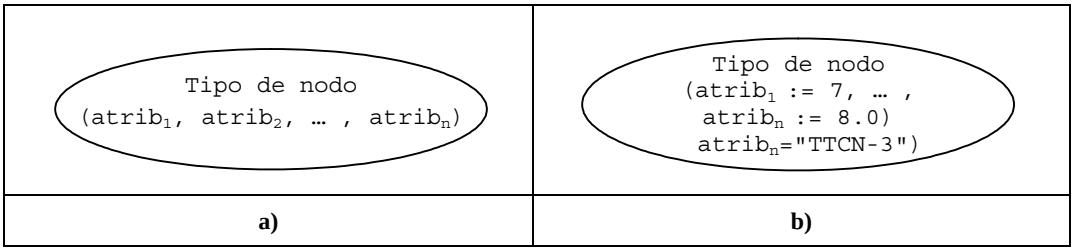


Figura 12/Z.143 – Nodos básicos con atributos

8.1.2.4 Nodos de referencia

Los nodos de referencia se refieren a segmentos de diagrama de flujo (véase 8.1.4), que son subdiagramas de flujo. Se define el significado del nodo de referencia mediante su sustitución por el segmento del diagrama de flujo referenciado en el diagrama de flujo. La inscripción del nodo en el nodo de referencia indica la referencia al segmento de diagrama de flujo. En la figura 13-a se muestra un nodo de referencia.

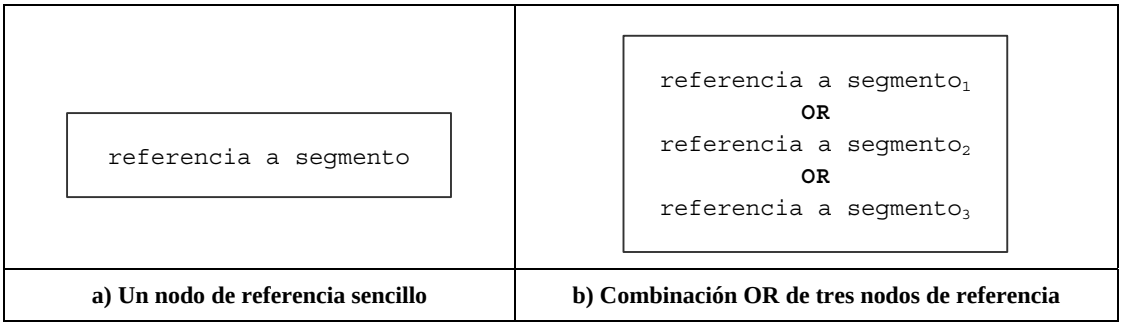


Figura 13/Z.143 – Nodo de referencia

8.1.2.4.1 Combinación OR de nodos de referencia

En algunos casos, un nodo de referencia podría ser sustituido por varios segmentos de diagrama de flujo. En estos casos puede utilizarse el operador **OR** para referirse a varios segmentos de diagrama de flujo (véase la figura 13-b). En el diagrama de flujo efectivo que representa un control de módulo, un caso de prueba o una función, la construcción representada determina una de las alternativas.

8.1.2.4.2 Apariciones múltiples de nodos de referencia

En algunos casos, un mismo tipo de nodo de referencia puede aparecer ninguna, una o varias veces en un diagrama de flujo. En expresiones corrientes, se describe la posible repetición de partes de una expresión corriente con los símbolos de operador '+' (para indicar una o varias repeticiones) y '*' (para indicar ninguna o varias repeticiones). Como se indica en la figura 14, estos operadores se incorporan a los diagramas de flujo con la introducción de nodos de referencia de marco doble junto con el símbolo del operador asociado. Una línea de flujo sencilla (véase 8.1.3) sustituirá al nodo de referencia si este aparece cero veces (se usa un nodo de referencia de marco doble junto con el operador '*').

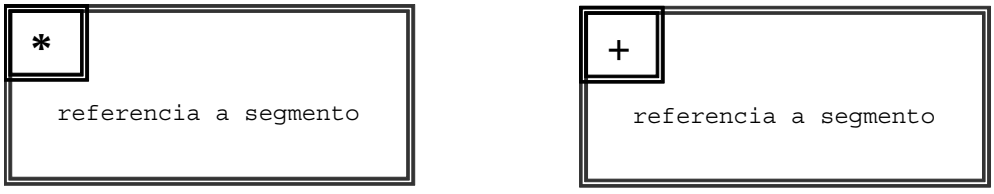


Figura 14/Z.143 – Repetición de nodos de referencia

Se puede indicar un número máximo de repeticiones del nodo de referencia colocando entre paréntesis un número después del símbolo '*' o '+' en el nodo de referencia de marco doble. La referencia a segmento que se muestra en la figura 15 puede ocurrir desde cero hasta 5 veces.

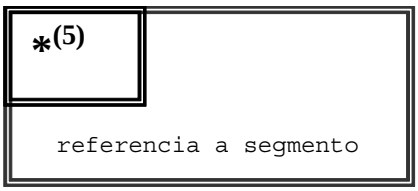
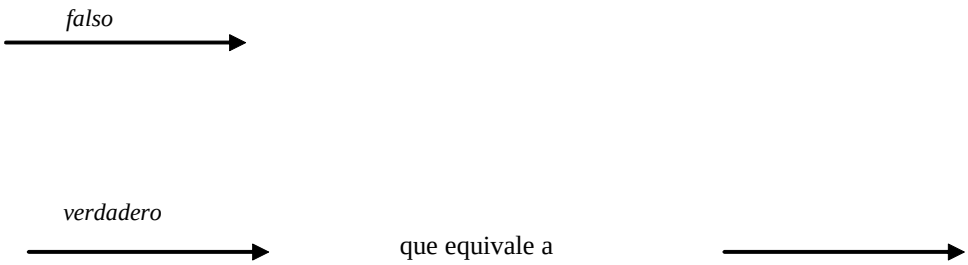


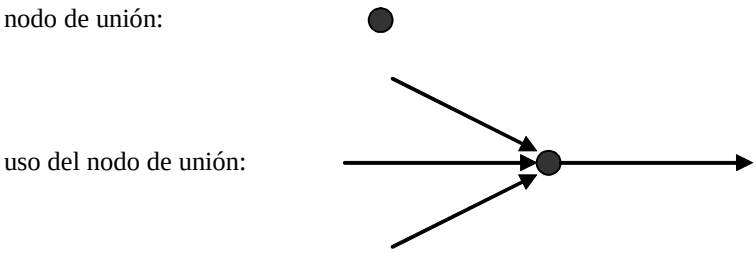
Figura 15/Z.143 – Repetición limitada de un nodo de referencia

8.1.3 Líneas de flujo

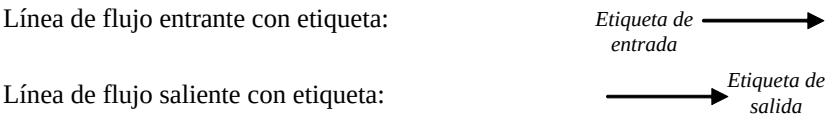
Las líneas de flujo se representan mediante flechas. Las líneas de flujo tienen la inscripción *true* o *false (verdadero* o *falso)*, que indica una condición para que se escoja la línea de flujo durante la interpretación del diagrama de flujo. Se admite una notación simplificada en la que se omite la inscripción true. A continuación se muestran ejemplos de las líneas de flujo:



Para representar la unión a nivel gráfico de varias líneas de flujo en una sola línea de flujo, se introduce un nodo especial de unión. A continuación se muestra un nodo de unión junto con un ejemplo de su uso:



Es incómodo dibujar líneas de flujo largas en diagramas grandes, como por ejemplo, las que se necesitan para modelar las construcciones **`goto`** y **`label`** de TTCN-3. En estos casos se pueden utilizar etiquetas en las líneas de flujo salientes y entrantes. A continuación se muestran algunos ejemplos:



Una línea de flujo saliente se conecta con una línea de flujo entrante mediante sus etiquetas, si éstas son idénticas. Las etiquetas de las líneas de flujo entrantes deben ser únicas. Se considera que varias líneas de flujo salientes con una misma etiqueta son una unión de líneas que desembocan en la línea de flujo entrante con esa misma etiqueta.

8.1.4 Segmentos de diagramas de flujo

Los segmentos de diagramas de flujo son subdiagramas de flujo. Se hace referencia a ellos en los nodos de referencia y definen el significado de ese nodo de referencia. Los segmentos de diagramas de flujo pueden incluir otros nodos de referencia.

Según se aprecia en la figura 16, los segmentos de diagrama de flujo poseen interfaces precisas compuestas por líneas de flujo entrantes y salientes. Hay tan sólo una línea de flujo sin etiqueta entrante y máximo una línea de flujo saliente sin etiqueta. Además, podrá haber varias líneas de flujo entrantes o salientes con etiqueta. Por ejemplo, la líneas de flujo entrantes y salientes son necesarias para describir el significado de las instrucciones **goto** y **alt** de TTCN-3.

Los segmentos del diagrama de flujo se colocan dentro de un marco y se escribe el nombre del segmento de diagrama de flujo después de la palabra clave **segment**, en el ángulo superior izquierdo del marco. Las líneas de flujo que describen la interfaz del segmento pasarán sobre el marco del segmento de diagrama de flujo.

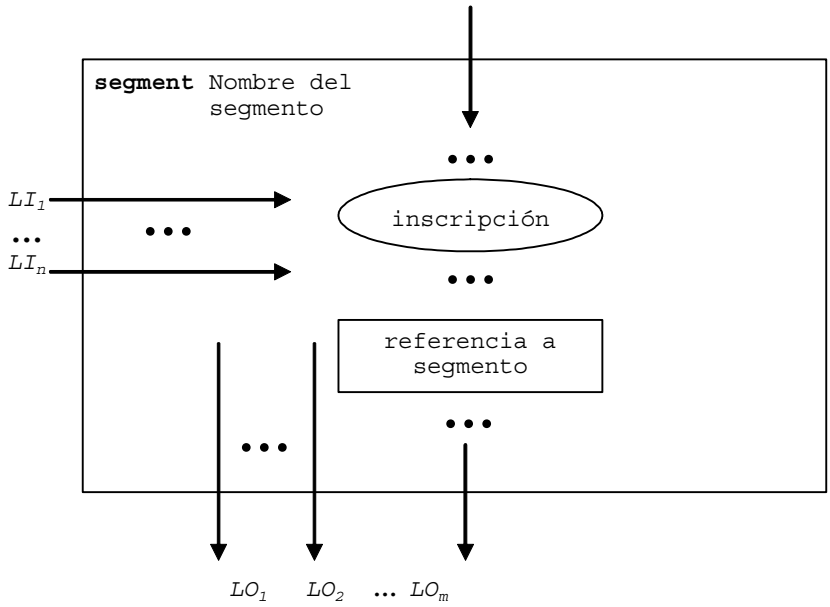


Figura 16/Z.143 – Estructura de la descripción de los segmentos de diagrama de flujo

8.1.5 Comentarios

A fin de facilitar la lectura y la coherencia, se puede utilizar un símbolo especial de comentario para asociar comentarios con los nodos y líneas de flujo del diagrama de flujo. En la figura 17 se muestra el símbolo de comentario y se ilustra su uso.

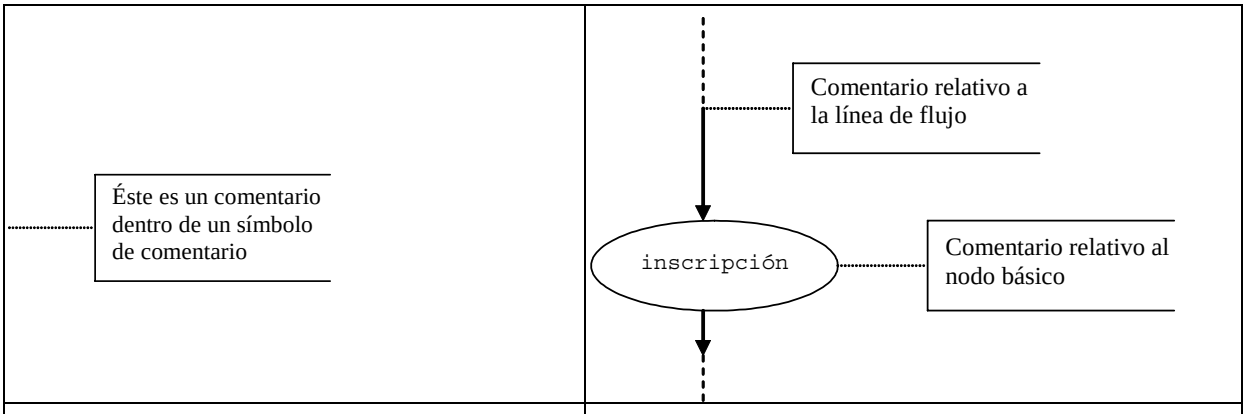


Figura 17/Z.143 – Representación de comentarios en los diagramas de flujo

8.1.6 Tratamiento de las descripciones de los diagramas de flujo

El procedimiento de evaluación de la semántica operacional recorre diagramas de flujo compuestos únicamente por nodos básicos, es decir que deben expandirse todos los nodos de referencia utilizando sus correspondientes definiciones de segmento de diagrama de flujo. Es preciso emplear la función *NEXT* para efectuar dicho recorrido. A continuación se define la función *NEXT*:

actualNodeRef.*NEXT*(*bool*) := `successorNodeRef` donde:

- *actualNodeRef* es la referencia a un nodo de diagrama de flujo básico;
- *successorNodeRef* es la referencia a un nodo sucesor del nodo al que hace referencia *actualNodeRef*;
- *bool* es una variable booleana que indica si se devuelve el sucesor true o el sucesor false (véase 8.1.3).

8.2 Representación del comportamiento de TTCN-3 mediante diagramas de flujo

La semántica operacional supone que las descripciones del comportamiento de TTCN-3 tienen la forma de un conjunto de diagramas de flujo. Es decir, se debe construir un diagrama de flujo independiente para cada descripción de comportamiento de TTCN-3.

La semántica operacional interpreta que los siguientes tipos de definiciones TTCN-3 son descripciones de comportamiento:

a) control del módulo;

b) definiciones de estudios de caso;

c) definiciones de funciones;

d) definiciones de altstep;

e) definiciones de tipo de componente.

El módulo de control especifica el recorrido de la prueba, es decir, el orden de ejecución (posiblemente repetitivo) de los casos de prueba reales. Las definiciones de prueba definen el comportamiento del MTC. Las funciones estructuran el comportamiento. Las ejecuta el módulo de control o los componentes de prueba. Los altstep se utilizan para definir el comportamiento por defecto o para estructurar el comportamiento de la misma manera que las funciones. Se supone que las definiciones de tipo de componente son descripciones de comportamiento ya que especifican la creación, declaración e inicialización de puertos, constantes, variables y temporizadores durante la creación de los ejemplares de tipos de componente.

8.2.1 Procedimiento de construcción de diagramas de flujo

Los diagramas de flujo de las figuras 18 a 22 y los segmentos de diagrama de flujo presentados en la cláusula 8 son sólo plantillas. Indican la *ubicación* de la información que debe suministrarse durante la creación de un diagrama de flujo o de un segmento de diagrama de flujo concretos. Estas ubicaciones se señalan mediante los paréntesis angulares '<' y '>'.

La representación de un módulo de TTCN-3 mediante un diagrama de flujo se realiza en tres pasos:

1) Para cada instrucción de TTCN-3 del control de módulo, los casos de prueba, los altstep, las funciones y las definiciones de tipo de componente, se construye un segmento concreto de diagrama de flujo.

2) Para el control de módulo y para cada caso de prueba, altstep, función y definición de tipo de componente se construye un diagrama de flujo (con nodos de referencia).

3) Todos los nodos de referencia de los diagramas de flujo se sustituyen, paso a paso, por sus correspondientes definiciones de segmento de diagrama de flujo, hasta que los diagramas de flujo contengan únicamente un nodo inicial, nodos finales y nodos básicos de diagramas de flujo.

NOTA 1 – Los nodos básicos de diagramas de flujo describen unidades de ejecución indivisibles. La semántica operacional de TTCN-3 se basa en la interpretación de nodos básicos de diagramas de flujo. En la cláusula 8.6 se presentan algunos métodos de ejecución aplicables únicamente a nodos básicos de diagramas de flujo.

La sustitución de un nodo de referencia por su correspondiente definición de segmento de diagrama de flujo podría resultar en partes desconectadas del diagrama de flujo, es decir, partes a las que no se puede tener acceso desde el nodo inicial al recorrer el diagrama de flujo por las líneas de flujo. La semántica operacional hará caso omiso de las partes desconectadas de los diagramas de flujo.

NOTA 2 – Las partes desconectadas del diagrama de flujo resultan del procedimiento de sustitución mecánico. Para la construcción de una representación óptima mediante diagramas de flujo, deben también tenerse en cuenta las diversas combinaciones de instrucciones de TTCN-3. Sin embargo, esta Recomendación tiene como fin presentar una semántica correcta y completa y no una representación óptima con diagramas de flujo.

8.2.2 Representación del control del módulo mediante diagramas de flujo

Esquemáticamente, la estructura sintáctica de los módulos de TTCN-3 es:

```
module <identifier> <module-definitions-part> control <statement-block>
```

La siguiente es la única información pertinente al representar el comportamiento del diagrama de flujo:

```
module <identifier> <statement-block>
```

Ésta es similar a la definición de una función, y por lo tanto la representación del control del módulo mediante diagramas de flujo se asemeja a la representación de las funciones mediante diagramas de flujo (véase 8.2.4). La semántica accederá al diagrama de flujo que representa al control del módulo, utilizando el nombre del módulo.

NOTA – En la semántica operacional no se trata la parte de definiciones del módulo. Los parámetros del módulo se definen como constantes globales durante la ejecución. Deben sustituirse la referencias a parámetros de los módulos por sus valores concretos desde el punto de vista sintáctico (véase 8.3).

En la figura 18 se esquematiza la representación del control de módulo mediante diagramas de flujo. Con el nombre del diagrama de flujo, `control`, se identifica el diagrama de flujo que representa el control del módulo. Los nodos del diagrama de flujo se asocian con comentarios que describen el significado de los diversos nodos. El nodo de referencia `<stop-entity-op>` cubre el caso en que no haya una operación **`stop`** explícita, es decir, la semántica operacional supone que se añade implícitamente una operación **`stop`**.

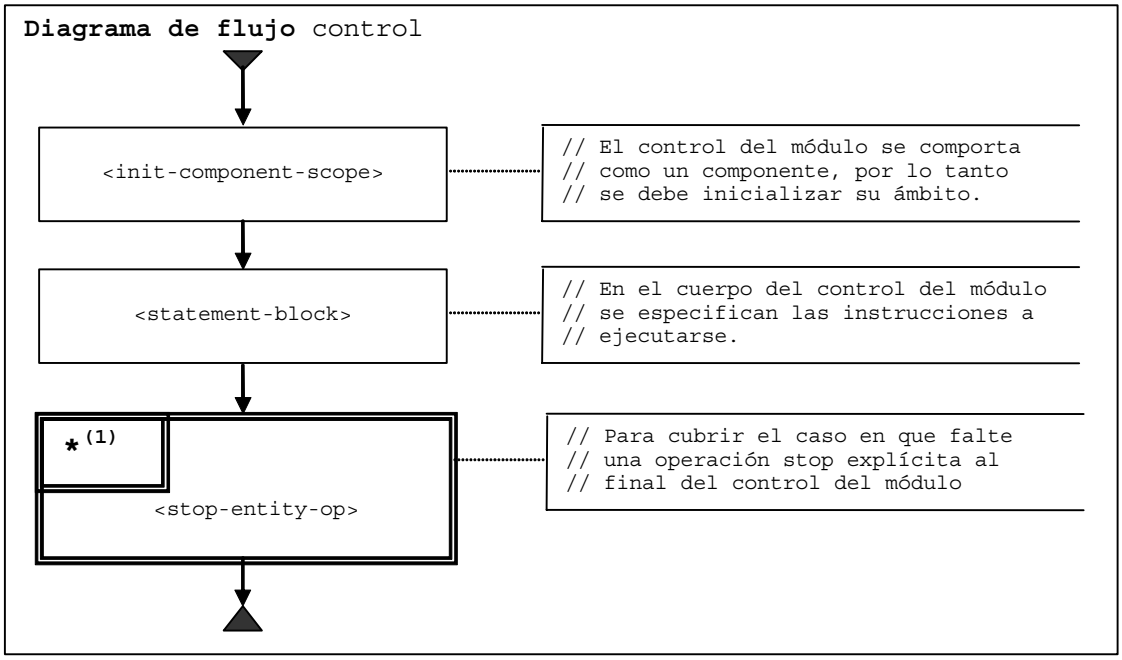


Figura 18/Z.143 – Representación del control del módulo mediante diagramas de flujo

8.2.3 Representación de casos de prueba mediante diagramas de flujo

Esquemáticamente, la estructura sintáctica de la definición de los casos de prueba de TTCN-3 es:

```
testcase <identifier> (<parameter>) <testcase-interface> <statement-block>
```

`<testcase-interface>` se refiere a los **`runs on`** (obligatorios) y a las cláusulas **`system`** (opcionales) en la definición del caso de prueba. La descripción del caso de prueba describe el comportamiento del MTC. La descripción de los casos de prueba mediante diagramas de flujo describe el comportamiento del MTC. La información suministrada por `<testcase-interface>` no es pertinente para el MTC, y aunque la instrucción **`execute`** la utiliza, no es necesario que aparezca en la representación con diagramas de flujo de los casos de prueba. Por lo tanto, la siguiente es la única información pertinente al representar el comportamiento del diagrama de flujo:

```
testcase <identifier> (<parameter>) <statement-block>
```

En la figura 19 se esquematiza la representación de los casos de prueba mediante diagramas de flujo. El nombre del diagrama de flujo, `<identifier>`, se refiere al nombre del caso de prueba representado. Los nodos del diagrama de flujo se asocian con comentarios que describen el significado de los diversos nodos. El nodo de referencia `<stop-entity-op>` cubre el caso en que no haya una operación **`stop`** explícita para el MTC, es decir, la semántica operacional supone que se añade implícitamente una operación **`stop`**.

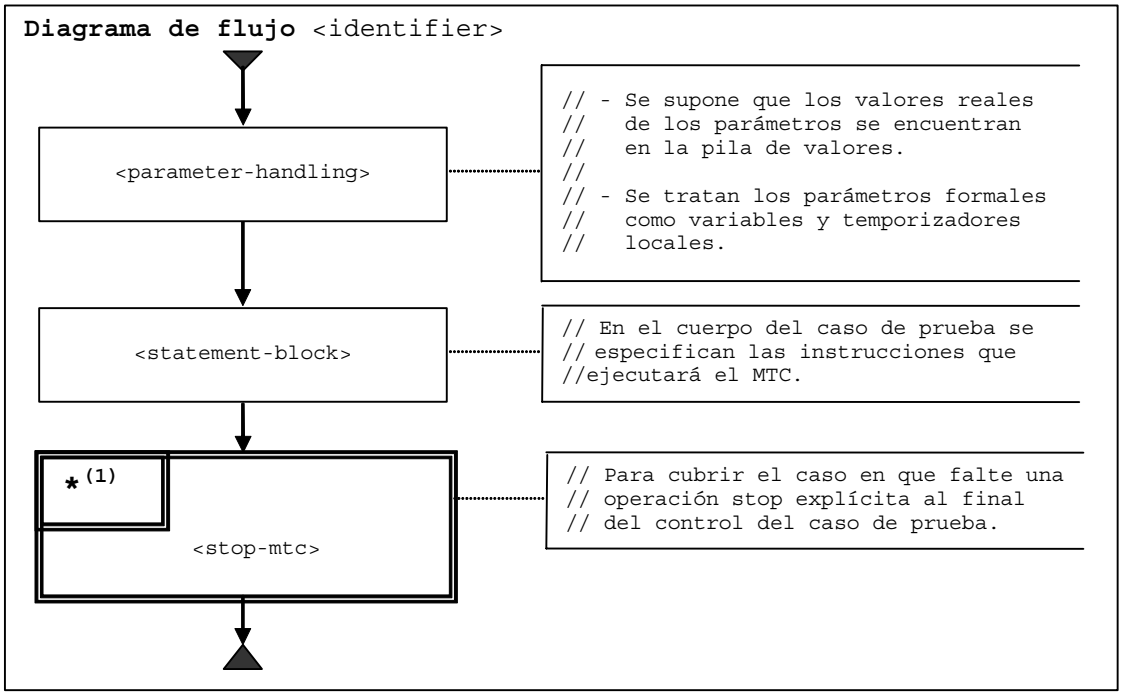


Figura 19/Z.143 – Representación de los casos de prueba mediante diagramas de flujo

8.2.4 Representación de funciones mediante diagramas de flujo

Esquemáticamente, la estructura sintáctica de la definición de las funciones de TTCN-3 es:

```
function <identifier> (<parameter>) [<function-interface>] <statement-block>
```

La `<function-interface>` opcional se refiere a los **`runs on`** y a las cláusulas **`return`** en la definición de la función. La información suministrada por `<function-interface>` no es relevante para la descripción del comportamiento, y aunque se utiliza para la verificación de la semántica, no es necesario representarla en el diagrama de flujo. Por lo tanto, únicamente la siguiente información es pertinente al representar el comportamiento del diagrama de flujo:

```
function <identifier> (<parameter>) <statement-block>
```

La semántica utilizará los nombres de las funciones para acceder a los diagramas de flujo que representan funciones.

En la figura 20 se esquematiza la representación de las funciones mediante diagramas de flujo. El nombre del diagrama de flujo, `<identifier>`, se refiere al nombre de la función representada. El nodo de referencia `<return-without-value>` cubre el caso en que no se especifique explícitamente una instrucción **`return`**, es decir, la semántica operacional supone que se añade implícitamente una operación **`return`**.

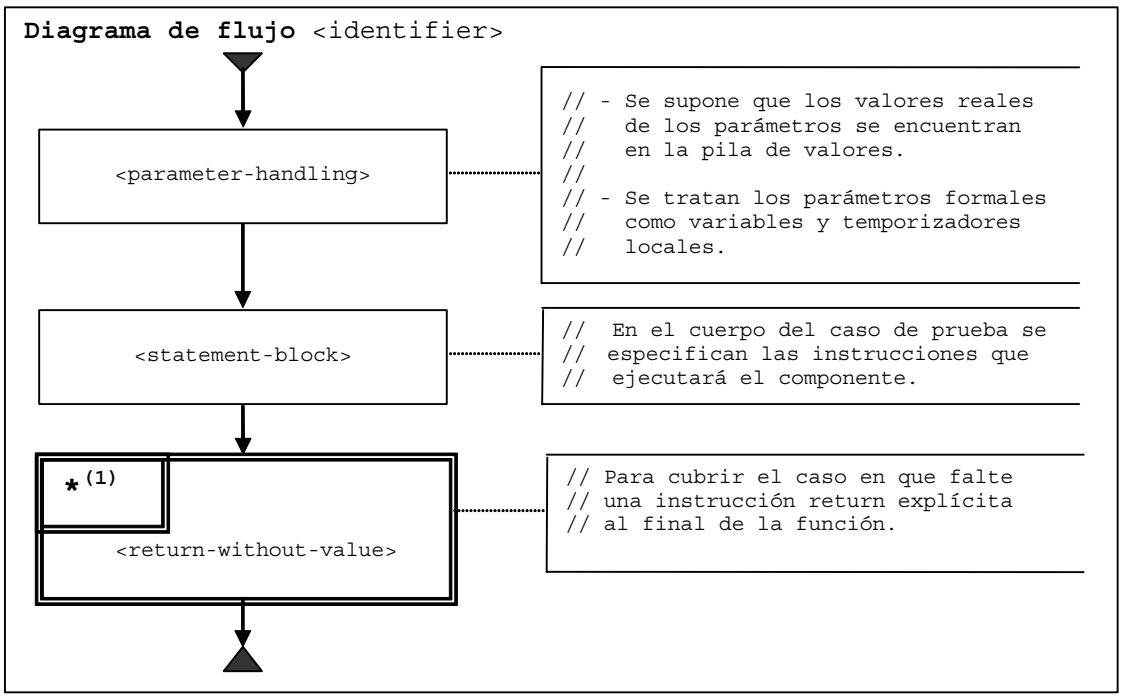


Figura 20/Z.143 – Representación de funciones mediante diagramas de flujo

8.2.5 Representación de altstep mediante diagramas de flujo

Esquemáticamente, la estructura sintáctica de los altstep de TTCN-3 es:

```
altstep <identifier> (<parameter>) [<altstep-interface>]
            <constant-variable-timer-declarations>
             { <receiving-branch> | <else-branch> }*
```

La <altstep-interface> opcional se refiere a la cláusula **runs on** en la definición del altstep. La información suministrada por < altstep-interface> no es relevante para la descripción del comportamiento, y aunque se utiliza para la verificación de la semántica, no es necesario representarla en el diagrama de flujo. Por lo tanto, únicamente la siguiente información es pertinente al representar el comportamiento del diagrama de flujo:

```
altstep <identifier> (<parameter>) [<altstep-interface>]
            <constant-variable-timer-declarations>
             { <receiving-branch> }*
             [ <else-branch> ]
```

NOTA – Únicamente se tienen en cuenta la primera rama else y las alternativas que van hasta la primera rama else. Las ramas posteriores a la primera rama else están fuera del alcance.

La semántica utilizará los nombres de los altstep para acceder a los diagramas de flujo que representan altstep.

En la figura 21 se esquematiza la representación de los altstep mediante diagramas de flujo. El nombre del diagrama de flujo, <identifier>, se refiere al nombre del altstep representado. El nodo de referencia <successful-altstep-termination> cubre el caso donde altstep se termina después de la selección y ejecución de una alternativa. El nodo de referencia <unsuccessful-altstep-termination> cubre el caso en que no se ejecute ninguna de las alternativas del altstep.

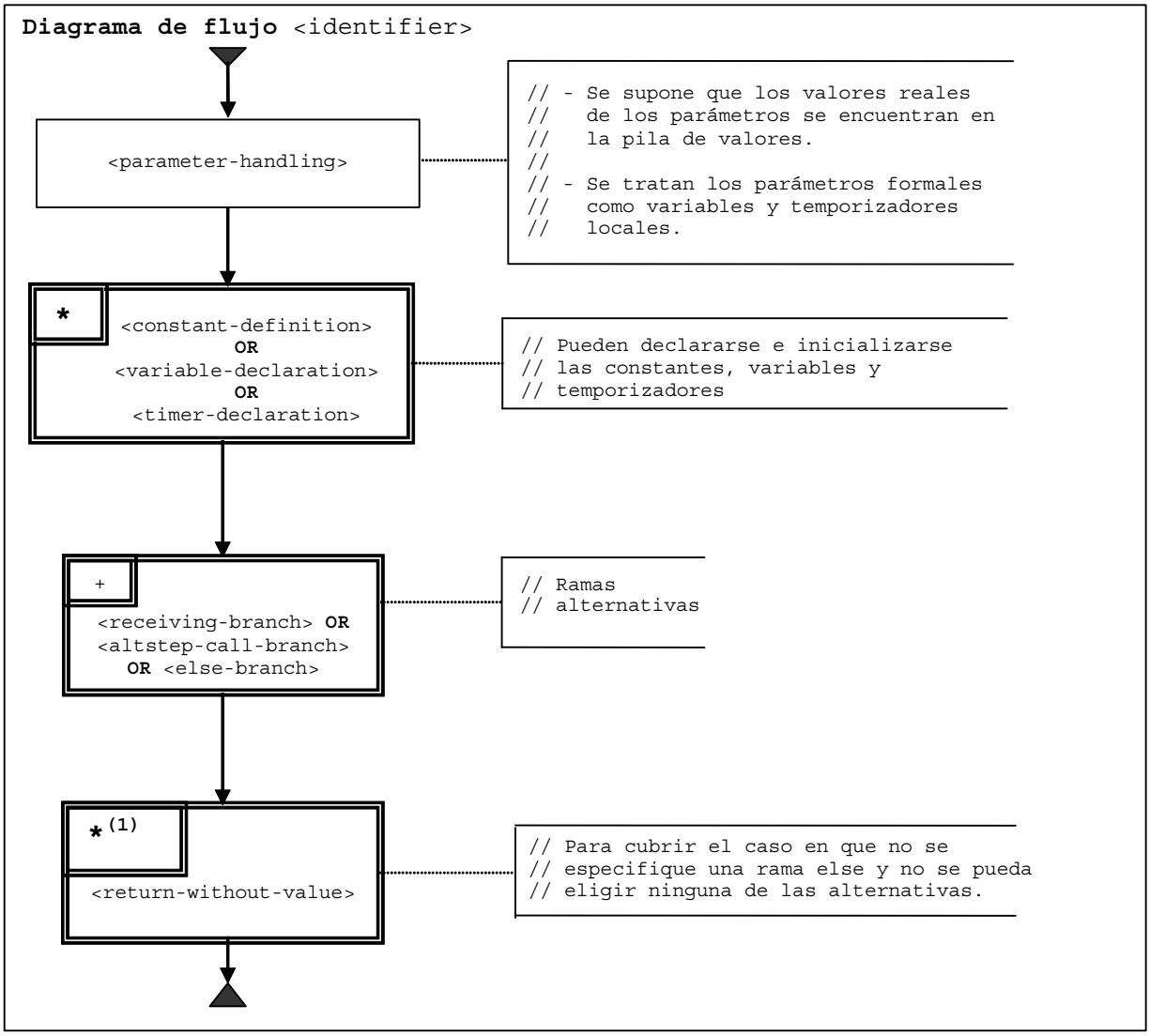


Figura 21/Z.143 – Representación de altstep mediante diagramas de flujo

8.2.6 Representación de definiciones de tipo de componente mediante diagramas de flujo

Esquemáticamente, la estructura sintáctica de las definiciones de tipo de componente de TTCN-3 es:

```
type component <identifier> <port-constant-variable-timer-declarations>
```

La semántica utilizará los nombres de los tipos de componente para acceder a los diagramas de flujo que representan tipos.

En la figura 22 se esquematiza la representación de las definiciones de tipo de componente mediante diagramas de flujo. El nombre del diagrama de flujo, `<identifier>`, se refiere al nombre del tipo de componente representado

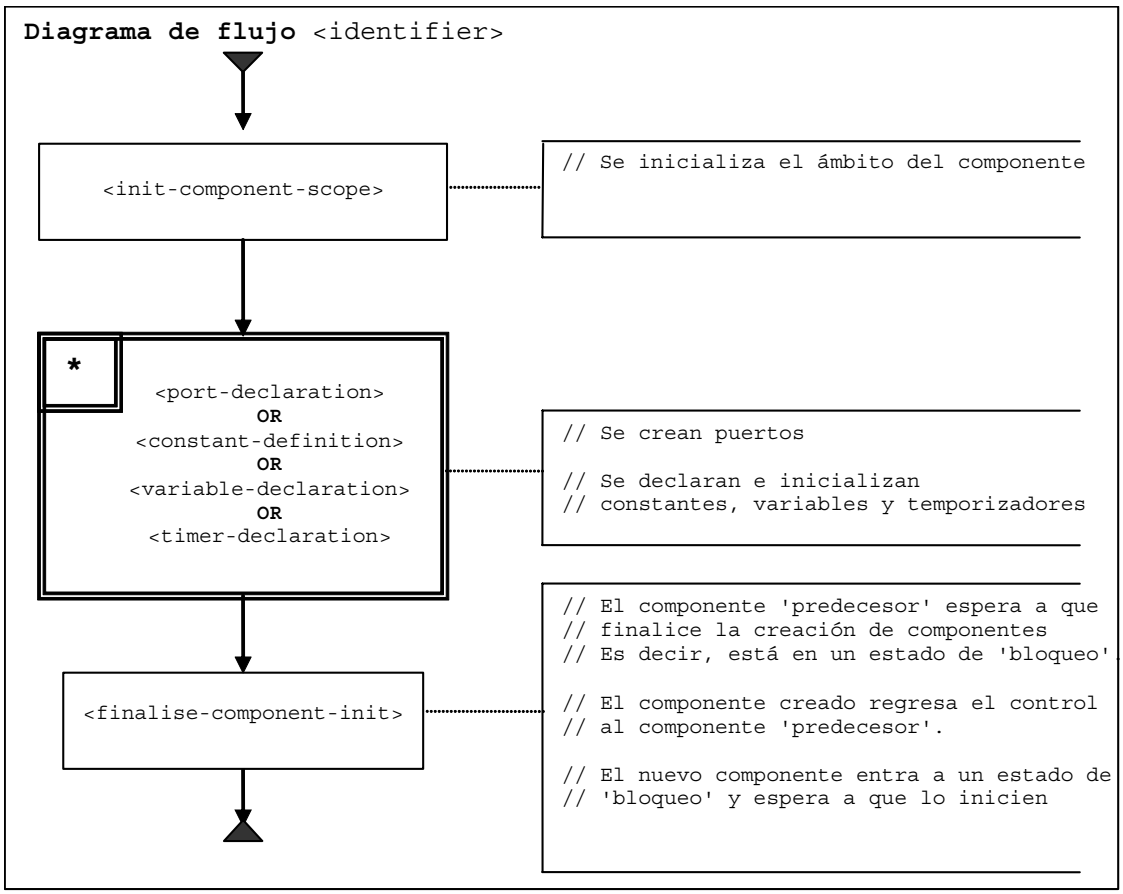


Figura 22/Z.143 – Representación de tipos de componente mediante diagramas de flujo

8.2.7 Recuperación del nodo inicial de los diagramas de flujo

La siguiente función es necesaria para recuperar la referencia del nodo inicial de los diagramas de flujo:

The GET-FLOW-GRAPH function: GET-FLOW-GRAPH (flow-graph-identifier)

La función devuelve una referencia al nodo inicial del diagrama de flujo con el nombre *flow-graph-identifier*. El *flow-graph-identifier* se refiere al nombre del módulo para el control, a nombres de casos de prueba, a nombres de funciones y a definiciones de tipo de componente.

8.3 Definiciones de estado de los módulos de TTCN-3

Para interpretar los diagramas de flujo que representan el comportamiento de TTCN-3, se utilizan los *estados del módulo*. Un estado de módulo es un estado estructurado compuesto por varios subestados que describen el estado de los componentes de prueba y de los puertos En esta cláusula se presentan los estados de los módulos, los estados de los componentes y los estados de los puertos. Adicionalmente, se definen las funciones usadas para recuperar información de los estados y para utilizarlos.

8.3.1 Estado del módulo

Tal y como se describe en la figura 23, el estado del módulo está compuesto por *ALL -ENTITY-STATES*, *ALL-PORT-STATES*, *MTC*, *TC-VERDICT*, *DONE* y *SNAP-ACTIVE*. *ALL-ENTITY-STATES* describe el estado del control del módulo y, durante el la ejecución de un caso de prueba, los estados de los componentes de prueba ejemplificados. *ALL-PORT-STATES*, la referencia *MTC* y el *TC-VERDICT* son pertinentes únicamente durante la ejecución del caso de prueba. *ALL-PORT-STATES* describe los estados de los diferentes puertos. *MTC* presenta una referencia al componente de prueba principal (MTC, *main test component*), *TC-VERDICT* almacena el veredicto global real del caso de prueba, *DONE* es una lista de todos los componentes de prueba que se detuvieron durante la ejecución del caso de prueba y *SNAP-ACTIVE* se utiliza como parte del estado puntual del MTC. *SNAP-ACTIVE* almacena el número de componentes de prueba que están activos en el momento en que el MTC determina el estado puntual. Se utiliza para evaluar las operaciones **`all component.done`** y **`all component.running`**.

NOTA 1 – El número de actualizaciones de TC-VERDICT es igual al número de componentes de prueba que han finalizado.

El comportamiento del control del módulo (*M-CONTROL*, en la figura 23) se procesa de la misma forma que un componente de prueba normal y su estado es el primer elemento de *ALL-ENTITY-STATES* en el estado del módulo.

ALL-ENTITY-STATES					ALL-PORT-STATES				MTC	TC-VERDICT	DONE	SNAP-ACTIVE
M-CONTROL	ES_1	...	ES_n		P_1	…	P_n					

Figura 23/Z.143 – Estructura del estado de un módulo

NOTA 2 – Se puede considerar que los estados de los puertos forman parte de los estados de las entidades. Los demás componentes pueden ver los puertos mediante **`connect`** y **`map`**, por lo tanto, la presente semántica operacional ubica los puertos en el nivel superior del estado del módulo.

8.3.1.1 Acceso al estado del módulo

MTC, *TC-VERDICT* y *SNAP-ACTIVE* forman parte del estado del módulo y se tratan como variables globales, es decir, se pueden utilizar las palabras clave *MTC* y *TC-VERDICT* para recuperar y modificar el valor del correspondiente estado de módulo.

NOTA 1 – Existe únicamente un estado de módulo durante la interpretación de un módulo de TTCN-3. Por lo tanto, se puede considerar que las palabras clave *MTC* y *TC-VERDICT* son los identificadores globales únicos para el procedimiento de evaluación.

Para manejar las listas *ALL-ENTITY-STATES*, *ALL-PORT-STATES* y *DONE*, se pueden utilizar las operaciones de lista *add*, *append*, *delete*, *member*, *first*, *length*, *next*, *random* y *change*. Éstas tienen el siguiente significado:

- *myList*.*add*(*item*) añade *item* como primer elemento de la lista *myList*;
- *myList*.*append*(*item*) anexa *item* como último elemento de la lista *myList*;
- *myList*.*delete*(*item*) suprime *item* de la lista *myList*;
- *myList*.*member*(*item*) devuelve **`true`** si *item* es un elemento de la lista *myList*, en caso contrario, devuelve **`false`**;
- *myList*.*first*() devuelve el primer elemento de *myList*;
- *myList*.*length*() devuelve la longitud de la lista *myList*;
- *myList*.*next*(*item*) devuelve el elemento que sigue a *item* en *myList*, o **`NULL`** si *item* es el último elemento de *myList*;
- *MyList*.*random*(<*condition*>) devuelve un elemento aleatorio de *myList*, que cumpla con la condición booleana <*condition*> o **`NULL`**, si ninguno de los elementos de *myList* cumplen <*condition*>;
- *MyList*.*change*(<*operation*>) permite que se pueda efectuar <*operation*> sobre todos los elementos de *myList*.

NOTA 2 – Las operaciones random y change no son operaciones corrientes de listas. Se incluyen a fin de explicar el significado de las palabras clave **`all`** y **`any`** en las operaciones de TTCN-3.

8.3.2 Estados de entidad

Los estados de entidad se utilizan para describir los estados reales del control del módulo y de los componentes de prueba. En el estado de módulo, los estados de entidad se manejan en la lista *ENTITY-STATES*. En la figura 24 se presenta la estructura de los estados de entidad.

<identifier>	STATUS	CONTROL-STACK	DEFAULT-LIST	DEFAULT-POINTER	VALUE-STACK	E-VERDICT	TIMER-GUARD	DATA-STATE	TIMER-STATE	SNAP-DONE

Figura 24/Z.143 – Estructura de los estados de entidad

<*identifier*> es un identificador único de una entidad, es decir, el control del módulo del componente de prueba en el sistema de prueba. Este tipo de identificador único se crea implícitamente para el control del módulo, el **`mtc`** y el sistema de prueba **`system`** cuando se inicia la ejecución de un módulo o se da inicio a la ejecución de un caso de prueba por medio de la instrucción **`execute`**. Se utiliza el identificador para identificar las entidades del sistema de prueba y hacer referencia a ellas. Por ejemplo en el caso de operaciones **`send`** con cláusulas **`to`** u operaciones **`receive`** con cláusulas **`from`**.

STATUS describe si el control del módulo o un componente de prueba se encuentra **ACTIVE**, **SNAPSHOT**, **REPEAT** o **BLOCKED**. Se bloquea el control del módulo durante la ejecución de los casos de prueba. Los componentes de prueba podrían bloquearse durante la creación de otros componentes de prueba, es decir, durante la ejecución de una operación **create**. El estado **SNAPSHOT** indica que el componente se encuentra activo, pero en la fase de evaluación de un estado puntual. El estado **REPEAT** indica que el componente está activo y ejecutando una instrucción alt que debería volverse a evaluar debido a una instrucción **repeat**.

CONTROL-STACK es una pila de referencias a nodos de diagramas de flujo. El elemento superior de *CONTROL-STACK* es el siguiente nodo del diagrama de flujo que ha de interpretarse. Es necesaria esta pila para modelar las solicitudes de funciones de forma adecuada.

La *DEFAULT-LIST* es una lista de opciones por defecto activas, es decir, una lista de apuntadores que se refiere a los nodos iniciales de opciones por defecto activas. La lista se construye en el orden inverso de activación, en el que la primera opción por defecto en activarse figura como último elemento de la lista.

Durante la ejecución del mecanismo de opciones por defecto, el *DEFAULT-POINTER* se refiere a la siguiente opción por defecto que debe evaluarse en caso de que la opción por defecto actual no finalice exitosamente.

VALUE-STACK es una pila de valores de todos los posibles tipos, que permite un almacenamiento intermedio de resultados intermedios o finales de operaciones, funciones e instrucciones. Por ejemplo, en *VALUE-STACK* se incluirán el resultado de la evaluación de una expresión o el resultado de la operación **mtc**. Además de los valores de todos los tipos de datos conocidos en el módulo, se define el valor especial **MARK** para que forme parte del alfabeto de la pila. Al salir de una unidad de ámbito, se utiliza **MARK** para borrar el contenido de la pila *VALUE-STACK*.

E-VERDICT almacena el veredicto efectivo local de un componente de prueba. Se hace caso omiso de *E-VERDICT* si una entidad de estado representa el control del módulo.

TIMER-GUARD representa el temporizador especial necesario para vigilar el tiempo de ejecución de los casos de prueba y la duración de las operaciones call. *TIMER-GUARD* se modela como una vinculación de temporizador (véanse 8.3.2.4 y la figura 28)

Se considera que *DATA-STATE* es una lista de listas de vinculaciones de variables. La estructura de la lista de listas revela unidades de ámbito anidadas como consecuencia de solicitudes anidadas de funciones. Cada lista de la lista de listas de vinculaciones de variables describe las variables conocidas y sus valores en una unidad de ámbito determinada. Al ingresar y al salir de una unidad de ámbito se añade o suprime una lista de vinculaciones de variables de la lista *DATA-STATE*. En 8.3.2.2 se presenta una descripción de la parte *DATA-STATE* del estado de una entidad.

Se considera que *TIMER-STATE* es una lista de listas de vinculaciones de temporizadores. La estructura de la lista de listas revela unidades de ámbito anidadas como consecuencia de solicitudes anidadas de funciones. Cada lista de la lista de listas de vinculaciones de temporizadores describe los temporizadores conocidos y su estado en una unidad de ámbito determinada Al ingresar y al salir de una unidad de ámbito se añade o se suprime una lista de vinculaciones de variables de la lista *timer-state*. A En 8.3.2.4 se presenta una descripción de la parte *timer-state* del estado de una entidad.

SNAP-DONE soporta la semántica del estado puntual de los componentes de prueba. Cuando se precisa un estado puntual, se asigna a *SNAP-DONE* una copia de la lista *DONE* del estado del módulo, es decir, *SNAP-DONE* es una lista de identificadores de componente de los componentes detenidos.

8.3.2.1 Acceso a los estados de entidad

<*identifier*> es el identificador único del estado de una entidad, el cual se puede utilizar para acceder al componente representado por el estado de la entidad y las diversas partes del estado de la entidad.

Las partes del estado de entidad, *STATUS*, *DEFAULT-POINTER*, *E-VERDICT* y *TIMER-GUARD* se tratan como variables globalmente visibles, es decir que se pueden recuperar o modificar los valores de *STATUS*, *DEFAULT-POINTER* y *E-VERDICT* utilizando la notación de "punto". Por ejemplo, *myEntity*.*STATUS*, *myEntity*.*DEFAULT-POINTER* y *myEntity*.*E-VERDICT*, donde *myEntity* hace referencia al estado de una entidad.

NOTA – De ahora en adelante, se supone que se puede utilizar la notación de "punto" empleando referencias e identificadores únicos. Por ejemplo, en *myEntity*.*STATUS*, *myEntityState* podría ser un apuntador al estado de una entidad o el valor del campo <*identifier*>.

Se puede hacer referencia al *CONTROL-STACK*, *DEFAULT-LIST* y *VALUE-STACK* del estado de la entidad *myEntity* mediante la notación de "punto", así: *myEntity*.*CONTROL-STACK*, *myEntity*.*DEFAULT-LIST* y *myEntity*.*VALUE-STACK*.

Se puede acceder a *CONTROL-STACK* y a *VALUE-STACK* y manejarlas haciendo uso de las operaciones de pila *push*, *pop*, *top*, *clear* y *clear-until*. Los siguientes son los significados de estas operaciones de pila:

- myStack.push(item) incluye item en la pila myStack;

- myStack.pop() extrae el elemento en la cima de la pila myStack;
- myStack.top() devuelve el elemento en la cima de la pila myStack o **NULL** si myStack se encuentra vacía;
- myStack.clear() despeja myStack, es decir, extrae todos los elementos de myStack;
- myStack.clear-until(item) extrae elementos de la pila myStack hasta encontrar el elemento item o hasta que myStack se vacíe.

Se puede acceder a *DEFAULT-LIST* y manejarla haciendo uso de las operaciones de pila *add*, *append*, *delete*, *member*, *first*, *length*, *next*, *random* y *change*. En 8.3.1.1 se define el significado de dichas operaciones.

Para crear un nuevo estado de entidad, se supone que se tiene acceso a la función *NEW-ENTITY*:

- *NEW-ENTITY (entityIdentifier, flow-graph-node-reference)*;

Crea un nuevo estado de entidad y devuelve su referencia. Los siguientes son los valores del nuevo estado de entidad:

- <identifier> se fija a *entityIdentifier* y será un identificador global único;
- *STATUS* se fija a **ACTIVE**;
- *flow-graph-node-reference* es el único elemento (superior) en *CONTROL-STACK*;
- *DEFAULT-LIST* es una lista vacía;
- *DEFAULT-POINTER* tiene el valor **NULL**;
- *VALUE-STACK* es una pila vacía;
- *E-VERDICT* se fija a **none**;
- *TIMER-GUARD* es una nueva vinculación de temporizador (véase 8.3.2.4) con el nombre GUARD, estado **IDLE** y sin duración por defecto;
- *DATA-STATE* es una lista vacía;
- *TIMER-STATE* es una lista vacía;
- *SNAP-DONE* es una lista vacía.

Cuando se recorre un diagrama de flujo, el valor de la pila *CONTROL-STACK* cambia con frecuencia y de una misma forma: se extrae el elemento en la cima de la pila *CONTROL-STACK* y el nodo sucesor del elemento extraído se ingresa en esa misma pila. Esta serie de operaciones de pila se encapsula en la función *NEXT-CONTROL*:

```
myEntity.NEXT-CONTROL(myBool) {
      successorNode := myEntity.CONTROL-STACK.NEXT(myBool).top();
      myEntity.CONTROL-STACK.pop();
      myEntity.CONTROL-STACK.push(successorNode);
}
```

8.3.2.2 Estado de datos y vinculación de variables

Tal y como se muestra en la figura 25, el estado de datos *DATA-STATE* del estado de una entidad consiste en una lista de listas de vinculaciones de variables. Cada lista de vinculaciones de variables define las vinculaciones de variables en una unidad de ámbito determinada. La acción de añadir una nueva lista de vinculaciones de variables corresponde a ingresar a una nueva unidad de ámbito, como por ejemplo cuando se solicita una función. La acción de suprimir una lista de vinculaciones de variables corresponde a abandonar una unidad de ámbito, como cuando una función ejecuta la instrucción **return**.

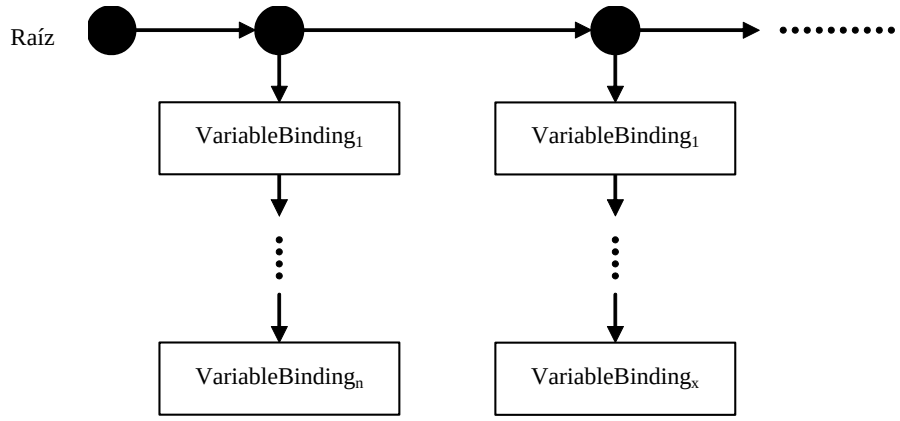


Figura 25/Z.143 – Estructura de la parte *DATA-STATE* del estado de una entidad

En la figura 26 se muestra la estructura de una vinculación de variable. Cada variable tiene un nombre, una <*location*> (ubicación) y un *VALUE* (valor). *VAR NAME* identifica una variable en una unidad de ámbito. <*location*> es un identificador único del sitio en que se almacena el valor de la variable. La parte *VALUE* de la vinculación de una variable describe el valor efectivo de la variable.

NOTA – Se proporcionarán automáticamente identificadores únicos de ubicación cuando se declare cada variable.

Figura 26/Z.143 – Estructura de la vinculación de una variable

Se hace una distinción entre el nombre de la variable y su ubicación, a fin de modelar de forma adecuada las solicitudes de funciones y la ejecución de casos de prueba con paso de parámetros por valor y por referencia:

a) Se tratan los parámetros pasados por valor de la misma forma en que se declara una variable nueva, es decir, se anexa una nueva vinculación de variable a la lista de vinculaciones de variable del ámbito de la función solicitada o del caso de prueba ejecutado. La nueva vinculación de variable utiliza el nombre formal de parámetro *VAR-NAME*, recibe una nueva ubicación y obtiene el valor que se pasa a la función o caso de prueba.

b) Los parámetros pasados por referencia también implican una nueva vinculación de variable en el ámbito de la función solicitada o del caso de prueba ejecutado. La nueva vinculación de variable también utiliza el nombre formal de parámetro *VAR-NAME*, pero no recibe una nueva ubicación ni un valor nuevo. La vinculación de variable obtiene copias de <*location*> y *VALUE* de la variable que se pasa por referencia.

Cuando se actualiza el valor de una variable, como cuando se hace una asignación a una variable, se emplea el nombre de la variable para identificar una ubicación y se actualizan al mismo tiempo todas las vinculaciones de variables con la misma ubicación. Por lo tanto, al abandonar la unidad de ámbito, se puede suprimir la lista de variables pertenecientes a dicha unidad de ámbito, sin posteriores actualizaciones. Como resultado del proceso de actualización, las variables que se pasan por referencia tienen automáticamente el valor correcto.

8.3.2.3 Acceso a los estados de datos

Puede recuperarse el valor de una variable utilizando la notación de "punto" *myEntity.myVar*.*VALUE*, donde *myEntity* se refiere al estado de una unidad y *myVar* es el valor de una variable.

Se definen las siguientes funciones a fin de poder manejar las variables y el ámbito de las variables:

a) La función *VAR-SET*: *myEntity.VAR-SET* (*myVar*, *myValue*)

fija a *myVal* la parte *VALUE* de la variable *myVar* en el ámbito efectivo de la entidad *myEntity*. Adicionalmente, también se fijarán a *myVal* la parte *VALUE* de todas las variables con la misma ubicación que *myVar*.

b) La función *INIT-VAR*: *myEntity.INIT-VAR* (*myVar*, *myVal*)

crea una nueva vinculación de variable para la variable *myVar* con el valor inicial *myVal* en la unidad de ámbito real de la entidad *myEntity*. Si se usa la palabra clave **NONE** en vez de *myVal*, es porque se crea una variable con un valor inicial no definido. Se crea automáticamente un nuevo valor único <*location*>.

c) La función *GET-VAR-LOC*: *myEntity.GET-VAR-LOC* (*myVar*)

recupera la ubicación de la variable *myVar* perteneciente a *myEntity*.

d) La función *INIT-VAR-LOC*: *myEntity.INIT-VAR-LOC* (*myVar*, *myLoc*)

crea una nueva vinculación de variable *myVar* con la ubicación *myLoc* en la unidad de ámbito real de *myEntity*. La variable se inicializará tomando el valor de otra variable con la ubicación *myLoc*.

NOTA – Las variables con una misma ubicación son resultado del paso de parámetros por referencia. Debido a la forma en que se tratan los parámetros pasados por referencia, tal como se describe en 8.3.2.2, todas las variables con una misma ubicación tendrán valores idénticos durante toda su existencia.

e) La función *INIT-VAR-SCOPE*: *myEntity.INIT-VAR-SCOPE* ()

inicializa un nuevo ámbito de variable en el estado de datos de la entidad *myEntity*, es decir, se anexa una lista vacía en la primera posición de la lista de listas de las vinculaciones de variables.

f) La función *DEL-VAR-SCOPE*: *myEntity.DEL-VAR-SCOPE* ()

suprime un ámbito de variable del estado de datos *myEntity*, es decir, se suprime la primera lista de la lista de listas de de las vinculaciones de variables.

8.3.2.4 Estado de temporizador y vinculación de temporizador

Conforme se indica en las figuras 27 y 25, son similares el estado de temporización *TIMER-STATE* y el estado de datos *DATA-STATE* de un estado vacío. Ambos son una lista de listas de vinculaciones y cada lista de vinculaciones define las vinculaciones válidas en un ámbito determinado. Añadir una nueva lista corresponde a ingresar a un nuevo ámbito y suprimir una lista de listas corresponde a abandonar una unidad de ámbito.

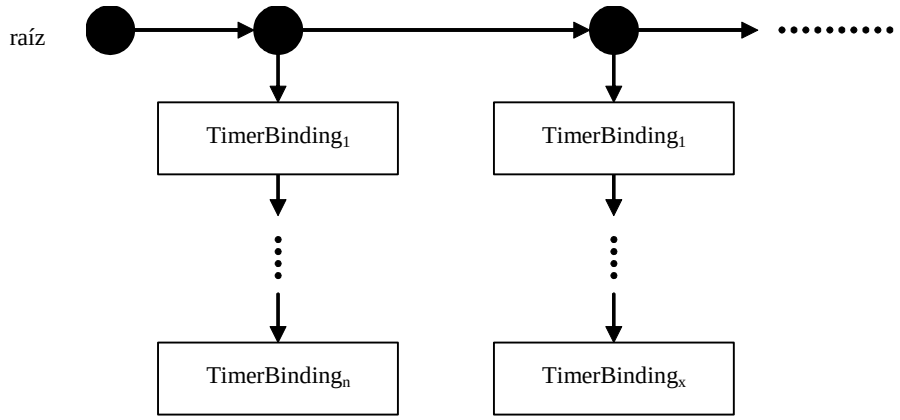


Figura 27/Z.143 – Estructura de la parte *TIMER-STATE* de un estado de entidad

En la figura 28 se presenta la estructura de las vinculaciones de temporizador. Los significados de *TIMER-NAME* y de <*location*> es similar al significado de *VAR-NAME* y de <*location*> en las vinculaciones de variable (figura 26).

TIMER-NAME	<location>	STATUS	DEF-DURATION	ACT-DURATION	TIME-LEFT	SNAP-VALUE	SNAP-STATUS

Figura 28/Z.143 – Estructura de las vinculaciones de temporizador

STATUS indica si el temporizador está activo, inactivo o si ha expirado. Los valores correspondientes de *STATUS* son **IDLE**, **RUNNING** y **TIMEOUT**. *DEF-DURATION* describe la duración por defecto del temporizador. En *ACT-DURATION* se almacena el tiempo efectivo transcurrido desde que se inició el temporizador. *TIME-LEFT* describe el tiempo efectivo que debe transcurrir para que el temporizador expire.

NOTA – Si un temporizador se declara sin duración por defecto, el valor de *DEF-DURATION* no está definido. *ACT-DURATION* y *TIME-LEFT* se fijan en 0.0 si se detiene un temporizador o si expira. Si un temporizador se inicia sin duración, se copia el valor de *DEF-DURATION* en *ACT-DURATION*. Ocurre un error dinámico, si se inicia un temporizador sin definir su duración.

SNAP-VALUE y *SNAP-STATUS* son necesarios para soportar la semántica de estado puntual de TTCN-3. Cuando se determina el estado puntual, *SNAP-VALUE* obtiene el valor efectivo de *ACT-DURATION* – *TIME-LEFT*, y *SNAP-STATUS* obtiene el mismo valor que *STATUS*. La evaluación del estado puntual se basará únicamente en los valores de *SNAP-VALUE* y *SNAP-STATUS*.

Los temporizadores podrán pasarse a las funciones únicamente por referencia. En otras palabras, el mecanismo es similar al mecanismo descrito para las variables en 8.3.2.2. Esto significa que se crea una nueva vinculación de temporizador (con el nombre de parámetro formal) a la que se copian <*location*>, *STATUS*, *DEF-DURATION*, *ACT-DURATION*, *TIME-LEFT*, *SNAP-VALUE* y *SNAP-STATUS* del temporizador que se pasa por referencia. Cuando se efectúa la actualización de un temporizador, se actualizan al mismo tiempo todas las vinculaciones de temporizador que tengan el mismo valor <*location*>.

8.3.2.5 Acceso a los estados de temporizador

Se pueden recuperar los valores de *STATUS*, *DEF-DURATION*, *ACT-DURATION*, *TIME-LEFT*, *SNAP-VALUE* y *SNAP-STATUS* del temporizador *myTimer*, utilizando la notación de punto:

- myEntity.myTimer.STATUS;
- myEntity.myTimer.DEF-DURATION;
- myEntity.myTimer.ACT-DURATION;
- myEntity.myTimer.TIME-LEFT;
- myEntity.myTimer.SNAP-VALUE;
- myEntity.myTimer.SNAP-STATUS.

myEntity en la notación de punto se refiere a un estado de entidad que representa el estado de un componente de prueba o control de módulo al que pertenece el temporizador *myTimer*.

Debe emplearse la operación genérica *TIMER-SET* para modificar los valores de *STATUS*, *DEF-DURATION*, *ACT-DURATION*, *TIME-LEFT*, *SNAP-VALUE* y *SNAP-STATUS* del temporizador *timer-name*, por ejemplo:

- *myEntity*.*TIMER-SET*(*myTimer*, *STATUS*, *myVal*).

fija el valor de *STATUS* del temporizador *myTimer* del ámbito efectivo de *myEntity* al valor *myVal*. Adicionalmente, también se fijarán a *myVal* el *STATUS* de todos los temporizadores con la misma ubicación que *myTimer*. También puede utilizarse la función *TIMER-SET* para modificar los valores de *DEF-DURATION*, *ACT-DURATION*, *TIME-LEFT*, *SNAP-VALUE* y *SNAP-STATUS.*

Se deben definir las siguientes funciones para poder manejar los temporizadores, el ámbito de temporizador y el estado puntual:

a) La función *INIT-TIMER*: *myEntity*.*INIT-TIMER* (*myTimer*, *myDuration*)

crea una nueva vinculación de temporizador para el temporizador *myTimer* con la duración por defecto, *myDuration*, en el ámbito real de la entidad *myEntity*. Si se usa la palabra clave **NONE** en vez de *myDuration*, es porque se crea un temporizador con un valor inicial no definido.

b) La función *GET-TIMER-LOC*: *myEntity*.*GET-TIMER-LOC* (*myTimer*)

recupera la ubicación del temporizador *myTimer* que pertenece a *myEntity*.

c) La función *INIT-TIMER-LOC*: *myEntity*.*INIT-TIMER-LOC* (*myTimer*, *myLocation*)

crea una nueva vinculación de temporizador para el temporizador *myTimer* con la ubicación *myLocation*, en la unidad de ámbito real de *myEntity*. Se inicializará el temporizador con los valores de *STATUS*, *DEF-DURATION*, *ACT-DURATION* y *TIME-LEFT* de otro temporizador con la ubicación <*location*>.

NOTA – Los temporizadores con una misma ubicación resultan del paso de parámetros por referencia. Debido a la forma en que se tratan los parámetros por referencia, que se describe en 8.3.2.3, todos los temporizadores con una misma ubicación tendrán valores idénticos de *STATUS*, *DEF-DURATION*, *ACT-DURATION* y *TIME-LEFT* durante toda su existencia.

d) La función *INIT-TIMER-SCOPE*: *myEntity*.*INIT-TIMER-SCOPE* ()

inicializa un nuevo ámbito de temporizador en el estado de temporizador de la entidad *myEntity*. En otras palabras, se anexa una lista vacía en la primera posición de la lista de listas de vinculaciones de temporizador.

e) La función *DEL-TIMER-SCOPE*: *myEntity*.*DEL-TIMER-SCOPE* ()

suprime un ámbito de temporizador del estado de temporizador de la entidad *myEntity*. En otras palabras, se suprime la primera lista de la lista de listas de vinculaciones de temporizadores.

f) La función *SNAP-TIMER*: *myEntity*.*SNAP-TIMER* ()

actualiza *SNAP-VALUE* y *SNAP-STATUS*, en todos los temporizadores que pertenecen a *myEntity* es decir:

```
myEntity.SNAP-TIMERS () {
 for all myTimer in TIMER-STATE {
     myEntity.myTimer.SNAP-VALUE := myEntity.myTimer.ACT-DURATION -
     myEntity.myTimer.TIME-LEFT;
     myEntity.myTimer.SNAP-STATUS := myEntity.myTimer.STATUS;
 }
```

8.3.3 Estados de puerto

Se utilizan los estados de puerto para describir los estados efectivos de los puertos. En un estado de módulo, los estados de puerto se manejan en la lista *ALL-PORT-STATES* (véase la figura 23). En la figura 29 se presenta la estructura de los estados de puerto. *PORT-NAME* se refiere al nombre de puerto que el componente de prueba *OWNER*, a quien pertenece el puerto, utiliza para identificar el puerto. *STATUS* proporciona el estado efectivo del puerto. Los puertos pueden estar **STARTED** (iniciado) o **STOPPED** (detenido).

NOTA – Se identifican inequívocamente los puertos de los sistemas de prueba mediante el componente de prueba a quien pertenecen, <owner> y mediante el nombre del puerto, <port-name>, local desde el punto de vista de <owner>.

La *CONNECTIONS-LIST* del estado de un puerto lleva un registro de las conexiones entre los diversos puertos del sistema de prueba. En 8.3.3.1 se explica este mecanismo.

En *VALUE-QUEUE* del estado de un puerto de almacenan los mensajes, solicitudes, respuestas y excepciones recibidas por este puerto y que todavía no se han agotado.

SNAP-VALUE soporta el mecanismo de estado puntual de TTCN-3. Al determinarse un estado puntual, se copia el primer elemento de *VALUE-QUEUE* en *SNAP-VALUE*. *SNAP-VALUE* tomará el valor **NULL** si *VALUE-QUEUE* está vacía o si *STATUS* vale **STOPPED**.

PORT-NAME	OWNER	STATUS	CONNECTIONS-LIST	VALUE-QUEUE	SNAP-VALUE

Figura 29/Z.143 – Estructura de los estados de puerto

8.3.3.1 Tratamiento de conexiones entre puertos

Se efectúa la conexión entre dos componentes de prueba conectando dos de sus puertos mediante la operación **connect**. De esta forma, los componentes podrán luego utilizar su nombre de puerto local para referirse a la cola distante. Como se muestra en la figura 30, la pareja compuesta por *REMOTE-ENTITY* y *REMOTE-PORT-NAME* representa a *connection* en los estados de las dos colas conectadas. *REMOTE-ENTITY* es el identificador único del componente de prueba al que pertenece el puerto distante. *REMOTE-PORT-NAME* se refiere al nombre local utilizado por la *REMOTE-ENTITY* para referirse a la cola. TTCN-3 soporta conexiones de uno a varios puertos, por lo que todas las conexiones de un puerto se organizan en una lista.

NOTA 1 – En la lista de conexiones también se tratan conexiones efectuadas con operaciones **map**. La operación **map**, **map**(*PTC1*:*MyPort*, **system**.*PCO1*), crea una nueva conexión (**system**, *PCO1*) en el estado de puerto *MyPort* que pertenece a *PTC1*. El extremo distante al que se conecta *PCO1*, reside dentro del SUT. Su comportamiento no se trata en la presente semántica.

NOTA 2 – La semántica operacional interpreta que la palabra clave **system** es una dirección ficticia. La conexión (**system**, *myPort*) en la lista de conexiones de un puerto indica que el puerto corresponde al puerto *myPort* en la interfaz del sistema de prueba.

REMOTE-ENTITY	REMOTE-PORT-NAME

Figura 30/Z.143 – Estructura de las conexiones

8.3.3.2 Tratamiento de los estados de los puertos

Se puede acceder y manejar la cola de valores del estado de los puertos utilizando las operaciones habituales de colas, *enqueue*, *dequeue*, *first* y *clear*. Al utilizarse las funciones *GET-PORT* o *GET-REMOTE*, se hace referencia a la cola a la que se accederá.

NOTA 1 – Las operaciones *enqueue*, *dequeue*, *first* y *clear* tienen el siguiente significado:

- *myQueue*.*enqueue*(*item*) pone el elemento *item* en el ultimo puesto de *myQueue*;
- *myQueue*.*dequeue*() suprime el primer elemento de *myQueue*;
- *myQueue*.*first*() devuelve el primer elemento de *myQueue* o **NULL** si *myQueue* está vacía;
- *myQueue*.*clear*() suprime todos los elementos de *myQueue*.

Se utilizan las siguientes funciones para manejar los estados de los puertos:

a) La función *NEW-PORT*: *NEW-PORT* (*myEntity*, *myPort*)

crea un puerto nuevo y devuelve su referencia. El nuevo puerto pertenece a *myEntity* y tiene el nombre *myPort*, es decir al puerto identificado por el componente de prueba *myEntity* y el nombre del puerto *myPort*. El estado del nuevo puerto es **STARTED**. *CONNECTIONS-LIST* y *VALUE-QUEUE* están vacías. *SNAP-VALUE* tiene el valor **NULL** (es decir, la cola de entrada del nuevo puerto está vacía).

b) La función *GET-PORT*: *GET-PORT* (*myEntity*, *myPort*)

devuelve una referencia al puerto identificado por el componente de prueba *myEntity* al que pertenece el puerto, así como el nombre del puerto, *myPort*.

c) La función *GET-REMOTE-PORT*: *GET-REMOTE-PORT* (*myEntity*, *myPort*, *myRemoteEntity*)

devuelve la referencia al puerto que pertenece al componente de prueba *myRemoteEntity* y está conectado al puerto identificado por *myEntity* y *myPort*. Si el puerto distante corresponde a un puerto de la interfaz del sistema de prueba, se devuelve la dirección ficticia **SYSTEM**.

NOTA 2 – *GET-REMOTE-PORT* devuelve **NULL** si no hay ningún puerto distante o si no se puede identificar inequívocamente el puerto distante. Si la entidad remota no se conoce o no se requiere, puede utilizarse el valor especial **NONE** como valor del parámetro *myRemoteEntity*, es decir, existe únicamente una conexión uno a uno para este puerto.

d) El *STATUS* de un puerto se trata como una variable. Se puede hacer referencia a ésta cualificando *STATUS* con una solicitud *GET-PORT*:

GET-PORT(*myEntity*, *myPort*).*STATUS*

e) La función *ADD-CON*: *ADD-CON* (*myEntity*, *myPort*, *myRemoteEntity*, *myRemotePort*)

añade la conexión (*myRemoteEntity*, *myRemotePort*) a la lista de conexiones del puerto *myPort*, que pertenece a *myEntity*.

f) La función *DEL-CON*: *DEL-CON* (*myEntity*, *myPort*, *myRemoteEntity*, *myRemotePort*)

suprime la conexión (*myRemoteEntity*, *myRemotePort*) de la lista de conexiones del puerto *myPort*, que pertenece a *myEntity*.

g) La función *SNAP-PORTS*: *SNAP-PORTS* (*myEntity*)

actualiza *SNAP-VALUE* de todos los puertos que pertenecen a *myEntity*, es decir:

```
SNAP-PORTS (myEntity) {
   for all ports p          /* in the module state */ {
      if (p.OWNER == myEntity) {
         if (p.STATUS == STOPPED) {
               p. SNAP-VALUE := NULL;
         }
         else {
               p. SNAP-VALUE := p.first()
         }
      }
   }
}
```

8.3.4 Funciones generales para manejar los estados de los módulos

La semántica operacional supone que se pueden utilizar las siguientes funciones para manejar los estados de los módulos.

NOTA 1 – Existe únicamente un estado de módulo durante la interpretación de un módulo TTCN-3. Se supone que los componentes del estado del módulo se almacenan en variables globales y no en un objeto de datos complejo. Por consiguiente, se supone que las siguientes funciones se ejecutan en variables globales y no se refieren a un objeto del estado de módulo específico.

a) La función *DEL-ENTITY*: *DEL-ENTITY*(*myEntity*)

suprime una entidad con el identificador único *myEntity*. La supresión comprende:

– la supresión del estado de entidad de *myEntity;*

– la supresión de todos los puertos que pertenecen a *myEntity*;

– la supresión de todas las conexiones en que participa *myEntity*.

b) La función *UPDATE-REMOTE-REFERENCES*:

UPDATE-REMOTE-REFERENCES (*source*, *target*)

UPDATE-REMOTE-REFERENCES actualiza en ambas entidades variables y temporizadores que tengan la misma ubicación. Los valores que se utilizarán para la actualización son los valores de variables y temporizadores que pertenecen a *source*.

NOTA 2 – Se utiliza *UPDATE-REMOTE-REFERENCES* durante la finalización de los casos de prueba. Permite actualizar variables del control del módulo, que se pasan a los casos de prueba como parámetros por referencia.

8.4 Mensajes, solicitudes de procedimiento, respuestas y excepciones

El intercambio de información entre componentes de prueba y entre componentes de prueba y el SUT se relaciona con *mensajes*, *solicitudes de procedimiento*, *respuestas a solicitudes de procedimiento* y *excepciones*. A efectos de la comunicación, estos elementos tienen que ser construidos, codificados y decodificados. La codificación concreta, es decir, la conversión de tipos de datos de TTCN-3 a bits y bytes, y la decodificación, es decir la conversión de bits y bytes a tipos de datos TTCN-3, no se tratan en la semántica operacional. En la presente Recomendación los *mensajes*, los *procedimientos de solicitud*, las *respuestas a solicitudes de procedimiento* y las *excepciones* se tratan a nivel teórico.

8.4.1 Mensajes

Los mensajes se relacionan con la comunicación centrada en mensajes. Las entidades que se comunican pueden intercambiar los valores de todos los tipos de datos (de los previamente definidos y de los definidos por el usuario). Como se muestra en la figura 31, la semántica operacional trata un mensaje como un objeto estructurado compuesto por las partes *emisor*, *tipo* y *valor*. La parte *emisor* identifica la entidad emisora del mensaje, la parte *tipo* especifica el tipo de mensaje y la parte *valor* define el valor del mensaje.

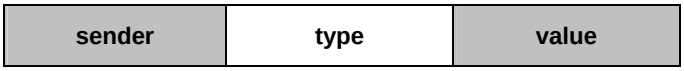


Figura 31/Z.143 – Estructura de los mensajes

NOTA – La semántica operacional sólo presenta un modelo de los conceptos de TTCN-3. La determinación de si la información del *emisor* es o tiene que ser enviada y/o recibida, así como la manera de hacerlo, dependen de la implementación del sistema de prueba; por ejemplo, en algunos casos la información del emisor puede ser una parte de la parte de valor de un mensaje, y por lo tanto no es una parte separada de la estructura del mensaje.

8.4.2 Solicitudes y respuestas de procedimiento

Las solicitudes de procedimiento y las respuestas a los procedimientos se relacionan con la comunicación centrada en procedimientos. Se definen como valores de un registro en el que los componentes representan los parámetros. La semántica operacional trata también las solicitudes de procedimiento y las respuestas a solicitudes de procedimiento como valores en tipos estructurados. En las figuras 32 y 33 se presentan la estructura de las solicitudes de procedimiento y la estructura de las respuestas.

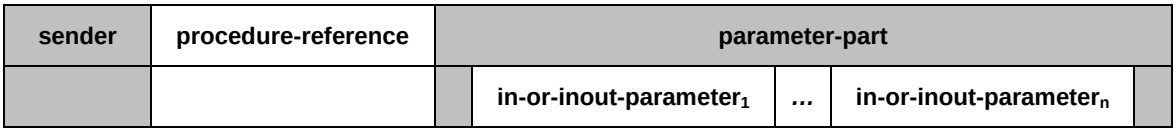


Figura 32/Z.143 – Estructura de las solicitudes de procedimiento

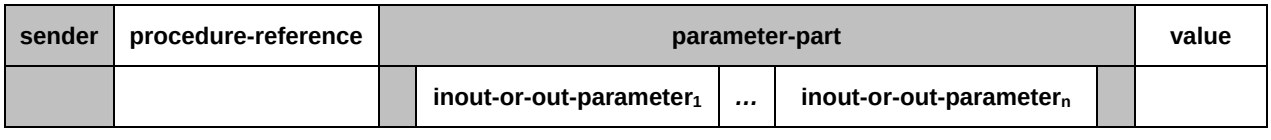


Figura 33/Z.143 – Estructura de las respuestas a las solicitudes de procedimiento

Las partes *emisor* y *referencia de procedimiento* tienen el mismo significado en ambas figuras. La parte *emisor* se refiere a la entidad emisora de una solicitud o a la respuesta a una solicitud de procedimiento. La *referencia de procedimiento* se refiere al procedimiento al que pertenecen la solicitud y la respuesta. La *parte de parámetros* de la solicitud de procedimiento en la figura 32 se refiere a los parámetros **entrada** y **entrada**/**salida** y la *parte de parámetros* de la respuesta en la figura 33 se refiere a los parámetros **entrada**/**salida** y **salida** del procedimiento al que pertenecen la solicitud y la respuesta. Además, la respuesta tiene una parte *valor* para los valores devueltos en la respuesta a un procedimiento.

NOTA 1 – Como se indica en la nota anterior (véase 8.4.1), la semántica operacional sólo presenta un modelo para los conceptos de TTCN-3. La determinación de si la información descrita en las figuras 32 y 33 es o tiene que ser enviada y/o recibida, depende de la implementación del sistema de prueba.

NOTA 2 – Para las solicitudes de procedimiento, los parámetros **salida** no son pertinentes y se omiten en la figura 32. Para las respuestas a solicitudes de procedimiento, los parámetros **in** no son pertinentes y se omiten en la figura 33.

NOTA 3 – Los tipos de parámetro y el tipo del valor en respuesta se pueden siempre calcular unívocamente a partir de la definición de firma asociada.

8.4.3 Excepciones

Las excepciones también se relacionan con la comunicación centrada en procedimientos. En la figura 34 se muestra la estructura de las excepciones, que está compuesta por cuatro partes: La parte *emisor* identifica al emisor de la excepción; la parte *referencia de procedimiento* hace referencia al procedimiento al que pertenece la excepción; la parte *tipo* identifica el tipo de excepción y la parte *valor* contiene el valor de la excepción. La firma de procedimiento indicada en la parte referencia de procedimiento define la lista de los tipos permitidos de excepción. Las excepciones recibidas serán conformes con uno de los tipos listados. Por lo general puede ser de cualquier tipo de datos TTCN-3, previamente definido o definido por el usuario.

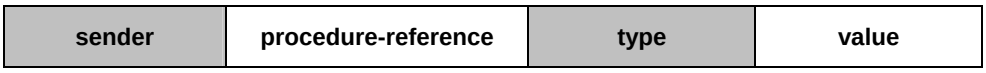


Figura 34/Z.143 – Estructura de las excepciones

8.4.4 Construcción de mensajes, solicitudes de procedimiento, respuestas y excepciones

Las operaciones utilizadas para enviar un mensaje, una solicitud de procedimiento, una respuesta a una solicitud de procedimiento o una excepción son: **send**, **call**, **reply** y **raise**. Todas estas operaciones de envío se construyen de la siguiente forma:

```
<port-name>.<sending-operation>(<send-specification>) [to <receiver>]
```

<port-name> y <sending-operation> definen el puerto y la operación utilizada para enviar un elemento. En las conexiones uno a varios, es necesario definir una entidad <receiver>. Se construye el elemento que se ha de enviar utilizando <send-specification>. La especificación de envío puede utilizar valores concretos, referencias de plantilla, valores de variable, constantes, expresiones, funciones, etc. para construir y codificar el tren que se ha de enviar.

La semántica operacional supone que existe la función genérica *CONSTRUCT-ITEM* (Construir elemento):

CONSTRUCT-ITEM (*myEntity*, <sending-operation>, <send-specification>)

devuelve un mensaje, una solicitud de procedimiento, una respuesta a una solicitud de procedimiento o una excepción, dependiendo de <sending-operation> y <send-specification> (tanto <sending-operation> como <send-specification> se refieren a las partes correspondientes de la operación de envío de TTCN-3). La referencia de entidad, *myEntity*, es el emisor del elemento que se ha de enviar. Se supone que esta información del *emisor* también forma parte del elemento que se ha de enviar (véanse las figuras 31 a 34).

8.4.5 Concordancia de mensajes, solicitudes de procedimiento, respuestas y excepciones

Las operaciones para recibir un mensaje, una solicitud de procedimiento, una respuesta a una solicitud de procedimiento o una excepción son: **receive**, **getcall**, **getreply** y **catch**. Todas estas operaciones de recepción se construyen de la siguiente forma:

```
<port-name>.<receiving-operation>(<matching-part>) [from <sender>] [<assignment-part>]
```

<port-name> y <receiving-operation> definen el puerto y la operación utilizados para recibir un elemento. En las conexiones uno a varios, se puede emplear una cláusula **from** para seleccionar una entidad emisora en particular, <sender>. El elemento que se ha de recibir debe cumplir las condiciones especificadas en <matching-

part>, es decir, tiene que concordar. <matching-part> puede usar valores concretos, referencias de plantilla, valores de variable, constantes, expresiones, funciones, etc. para especificar las condiciones de concordancia.

La semántica operacional supone que existe la función genérica *MATCH-ITEM*:

MATCH-ITEM (*myItem*, <matching-part>, <sender>)

devuelve **true** si *myItem* cumple las condiciones de <matching-part> y si <sender> envió *myItem*, en caso contrario, devuelve **false**.

8.4.6 Recuperación de información de los elementos recibidos

La información de los mensajes recibidos, solicitudes de procedimiento, respuestas a solicitudes de procedimiento y excepciones puede recuperarse de la parte <assignment-part> (véase 8.4.5) de las funciones de recepción **receive**, **getcall**, **getreply** y **catch.** La parte <assignment-part> describe la forma en que los parámetros de las solicitudes y respuestas de procedimiento, los valores devueltos codificados en respuestas, mensajes, excepciones y el identificador de la entidad <sender> se asignan a variables.

La semántica operacional supone que existe la función genérica *RETRIEVE-INFO* (recuperar información):

RETRIEVE-INFO (*myItem*, <assignment-part>)

todos los valores que han de recuperarse conforme a la parte <assignment-part> se recuperan y se asignan a las variables enumeradas en la parte de asignación. Las asignaciones se efectúan por medio de la operación *VAR-SET*, es decir, se actualizan al mismo tiempo las variables que tienen la misma ubicación.

8.5 Registros de solicitud de funciones, altstep y casos de prueba

Las funciones, los altstep y los casos de prueba se solicitan (o ejecutan) empleando el nombre y una lista de parámetros efectivos. Los parámetros efectivos proporcionan referencias para los parámetros de referencia y valores concretos para el parámetro de valor, definido por los parámetros formales en la definición de la función o del caso de prueba. La semántica operacional trata las funciones, los altstep y los casos de prueba utilizando los *registros de llamada* que se describen en la figura 35. El valor de *BEHAVIOUR-ID* es el nombre de una función o caso de prueba, los parámetros por valor proporcionan los valores concretos <$parId_1$> … <$parId_n$> para los parámetros formales <$parId_1$> … <$parId_n$>. Los parámetros por referencia proporcionan referencias a las ubicaciones de variables y temporizadores existentes. Antes de que pueda ejecutarse una función o un caso de prueba, se debe construir un registro de solicitud apropiado.

identificación de comportamiento	**parámetro de valor**					**parámetro de referencia**				
		parId$_1$	**…**	**parId$_n$**			**parId$_1$**	**…**	**parId$_n$**	
		valor$_1$	**…**	**valor$_n$**			**loc$_1$**	**…**	**loc$_n$**	

Figura 35/Z.143 – Estructura de los registros de solicitud

8.5.1 Tratamiento de los registros de solicitud

Para recuperar el nombre de la función o del caso de prueba y los valores reales de los parámetros, puede utilizarse la notación de punto, por ejemplo; *myCallRecord*.$parId_n$ o *myCallRecord*.*behaviour-id*, donde *myCallRecord* es un apuntador a un registro de solicitud.

Para construir una solicitud, se supone que se cuenta con la función *NEW-CALL-RECORD* (nuevo registro de solicitud):

NEW-CALL-RECORD(myBehaviour)

crea un nuevo registro de solicitud para la función o caso de prueba *myBehaviour* y devuelve un apuntador al nuevo registro. Los campos de parámetros del nuevo registro de solicitud tienen valores no definidos.

myEntity.INIT-CALL-RECORD(myCallRecord)

crea variables y temporizadores que se usan para manejar los parámetros por valor y por referencia en el ámbito efectivo del componente de prueba o control de módulo, *myEntity*. Se inicializan las variables para el tratamiento de los parámetros por valor con los valores correspondientes proporcionados en el registro de solicitud. Las variables y temporizadores para el tratamiento de los

parámetros por referencia reciben la ubicación que se proporciona. Además, obtienen el valor de una variable o temporizador de otra unidad de ámbito del componente en que se creó el registro de solicitud.

8.6 Procedimiento de evaluación de los módulos TTCN-3

8.6.1 Fases de evaluación

El procedimiento de evaluación de los módulos TTCN-3 está compuesto por:

1) *la fase de inicialización*;
2) *la fase de actualización*;
3) *la fase de selección*; y
4) *la fase de ejecución*.

Las fases 2), 3) y 4) se repiten hasta que finalice el control del módulo. El procedimiento de evaluación se describe por medio de una mezcla de texto informal, seudo código y las funciones descritas en las cláusulas previas.

8.6.1.1 Fase I: Inicialización

La fase de inicialización incluye las siguientes acciones:

a) **Declaración e inicialización de variables**

– *INIT-FLOW-GRAPHS*(); // Inicialización del tratamiento del diagrama de flujo.
// *INIT-FLOW-GRAPHS* se explica en 8.6.2

– *Entity* := **NULL**; // Se usará *Entity* para referirse a un estado de entidad.
// Los estados de entidad representan al control del
// módulo o un componente de prueba.

NOTA – Las siguientes variables globales *ALL-ENTITY-STATES*, *ALL-PORT-STATES*, *MTC*, *TC-VERDICT* y *DONE* conforman el estado del módulo manipulado durante la interpretación de los módulos TTCN-3 (véase 8.3.1).

– *ALL-ENTITY-STATES* := **NULL**;
– *ALL-PORT-STATES* := **NULL**;
– *MTC* := **NULL**;
– *TC-VERDICT* := **none**;
– *DONE* := **NULL**;
– *SNAP-DONE* := 0;

b) **Creación e inicialización del control de módulo**

– *Entity:*= *NEW-ENTITY* (*GET-UNIQUE-ID*(),*GET-FLOW-GRAPH* (<*moduleId*>));
// Se crea e inicializa un nuevo estado de módulo con el
// nodo inicial del diagrama de flujo que representa el
// comportamiento del control de módulo con el nombre
// <*moduleId*>. *GET-UNIQUE-ID* se explica en 8.6.2.

– *Entity*.*INIT-VAR-SCOPE*(); // Nuevo ámbito de variable
– *Entity*.*INIT-TIMER-SCOPE*(); // Nuevo ámbito de temporizador
– *Entity*.*VALUE-STACK*.*push*(**MARK**); // Se introduce una marca en la pila de valores
– *ALL-ENTITY-STATES*.*append*(*Entity*); // Se pone la nueva entidad en el estado de módulo.

8.6.1.2 Fase II: Actualización

La fase de actualización se relaciona con todas las acciones que están fuera del entorno de la semántica operacional pero que influyen en la interpretación de los módulos de TTCN-3. La fase de actualización comprende las siguientes acciones:

a) **Progresión del tiempo**: Se actualizan todos los temporizadores en funcionamiento. Es decir, los valores *TIME-LEFT* de los temporizadores en funcionamiento (posiblemente) se disminuyen, y, si debido a la actualización expira un temporizador, se actualizan las correspondientes vinculaciones de temporizador, es decir, *TIME-LEFT* se fija a 0.0 y *STATUS* se fija a **`TIMEOUT;`**

NOTA 1 – La actualización de temporizadores incluye la actualización de todos los temporizadores *TIMER-GUARD* en funcionamiento en los estados de módulo. Los temporizadores *TIMER-GUARD* se utilizan para controlar la ejecución de casos de prueba y de operaciones de solicitud.

b) **Comportamiento del SUT**: Los mensajes, solicitudes de procedimiento distantes y (posiblemente) excepciones recibidas del SUT se introducen en las colas de los puertos en las que se efectuarán las recepciones correspondientes.

NOTA 2 – La presente semántica operacional no establece hipótesis sobre la progresión del tiempo y el comportamiento del SUT.

8.6.1.3 Fase III: Selección

La fase de selección incluye las siguientes dos acciones:

a) **Selección**: Seleccionar una entidad no bloqueada, es decir, una entidad con el valor de *STATUS* **ACTIVE** o **SNAPSHOT**;

b) **Almacenamiento**: Almacenar el identificador de la entidad seleccionada en la variable global *Entity*.

8.6.1.4 Fase IV: Ejecución

La fase de ejecución incluye las siguientes dos acciones:

a) **Paso de ejecución de la entidad seleccionada**: Ejecutar el nodo superior del diagrama de flujo en la *CONTROL-STACK* de *Entity*;

b) **Comprobación del criterio de finalización**: Detener la ejecución si el módulo de control ha finalizado, es decir, la lista de estados de entidad está vacía; en caso contrario, continuar con la fase II.

NOTA – Puede considerarse que el paso de ejecución de la entidad seleccionada es una solicitud de procedimiento. La comprobación del criterio de finalización se hace cuando termina el paso de ejecución, es decir, se devuelve el control.

8.6.2 Funciones globales

El procedimiento de evaluación utiliza las funciones globales *INIT-FLOW-GRAPHS* y *GET-UNIQUE-ID*:

a) Se supone que *INIT-FLOW-GRAPHS* es la función que inicializa el tratamiento de los diagramas de flujo. El tratamiento puede incluir la creación de diagramas de flujo y el tratamiento de los punteros a los diagramas de flujo y a los nodos de los diagramas de flujo.

b) Se supone que *GET-UNIQUE-ID* es una función que devuelve un identificador único cada vez que la solicitan. El identificador único puede tener la forma de una variable de contador que se incrementa y devuelve cada vez que se solicita *GET-UNIQUE-ID*.

El seudo código utiliza las siguientes cláusulas para describir la ejecución de los nodos de los diagramas de flujo que utilizan las funciones *CONTINUE-COMPONENT*, *RETURN*, *****DYNAMIC-ERROR*****:

a) *CONTINUE-COMPONENT* hace que el componente de prueba real continúe la ejecución usando el nodo que se encuentra en la cima de la pila de control, es decir, no se devuelve el control al procedimiento de evaluación de módulo descrito en esta cláusula.

b) *RETURN* devuelve el control al procedimiento de evaluación de módulo descrito en esta cláusula. *RETURN* es la última acción que efectúa el "*paso de ejecución de la entidad seleccionada"* de la fase de *ejecución*.

c) *****DYNAMIC-ERROR***** indica que ocurrió un error dinámico. El procedimiento de tratamiento de error propiamente dicho está fuera del ámbito de la semántica operacional. Si ocurre un error dinámico, se busca que todo el comportamiento subsiguiente del caso de prueba sea indefinido. En dicho caso, se liberarán todos los recursos adjudicados al caso de prueba y se asignará el veredicto **error** al caso de prueba. Esto se modela mediante el segmento de diagrama de flujo <dynamic-error> (véase cláusula 9.18b)

NOTA – La aparición de un error dinámico está relacionada con el comportamiento de la prueba. Un error dinámico especificado por la semántica operacional indica un problema en la utilización de la TTCN-3, por ejemplo, uso indebido o condición de competencia.

d) *APPLY-OPERATOR* se usa como función genérica para describir la evaluación de operadores (como por ejemplo, +, *, / o –) en las expresiones (véase 9.18.4).

9 Segmentos de diagrama de flujo para las construcciones de TTCN-3

La semántica operacional utiliza diagramas de flujo para representar el comportamiento de TTCN-3. En 8.2 se describe el algoritmo para la construcción de diagramas de flujo que representan este comportamiento. Está basado en plantillas de diagramas de flujo y en segmentos de diagramas de flujo que deben utilizarse para la construcción de diagramas de flujo concretos del control del módulo, casos de prueba, altstep, funciones y definiciones de tipo de componente,

definidos en los módulos de TTCN-3. En la presente cláusula se enuncian las definiciones de las plantillas de los segmentos de diagrama de flujo. Se presentan en orden alfabético (del inglés) y no en un orden lógico.

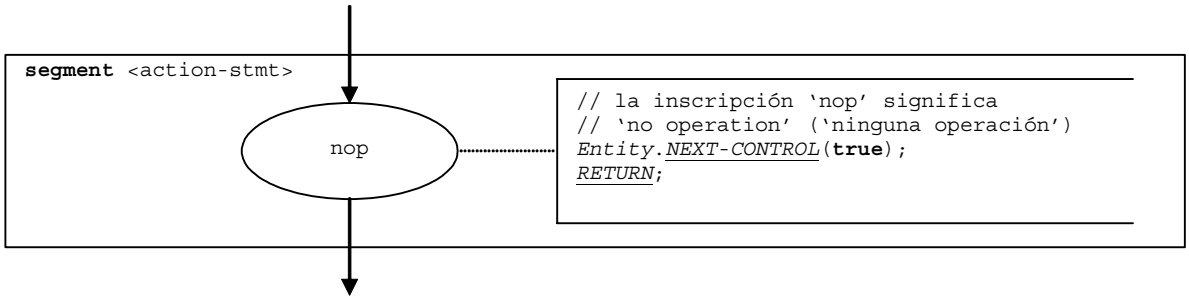
Las definiciones de segmentos de diagrama de flujo se presentan en la forma de figuras. Los nodos de diagrama de flujo se presentan a la izquierda de las figuras y los comentarios asociados a los nodos y a las líneas de flujo se muestran a la derecha. Se presentan comentarios descriptivos de los nodos de referencia, y a los nodos básicos se asocian comentarios en seudo código. El seudo código describe cómo se interpreta el nodo básico, es decir, cambia el estado del módulo. Utiliza las funciones definidas en la cláusula 8 y las variables globales declaradas e inicializadas en el proceso de evaluación de módulos TTCN-3 (véase 8.6). En la cláusula 8 se presenta una visión panorámica de todas las funciones y palabras clave que utiliza el seudo código.

9.1 Instrucción action

La siguiente es la estructura sintáctica de las instrucciones **`action`**:

```
action (<informal description>)
```

El segmento de diagrama de flujo <action-stmt> de la figura 36 define la instrucción **`action`**.



NOTA – El parámetro <descripción informal> de la instrucción **`action`** es irrelevante para la semántica operacional, por lo tanto no se representa en el segmento de diagrama de flujo.

Figura 36/Z.143 – Segmento de diagrama de flujo <action-stmt>

9.2 Instrucción activate

La siguiente es la estructura sintáctica de la instrucción **`activate`**:

```
activate(<altstep-name>([<act-par-desc1>, ... , <act-par-descn>]))
```

<altstep-name> indica el nombre del altstep que se activa como comportamiento por defecto, y <act-par-descr$_1$>, ... , <act-par-descr$_n$> describen los valores reales de los parámetros del altstep en el momento en que este se activa.

Se supone que para cada <act-par-desc$_1$> se conoce el correspondiente identificador formal de parámetro <f-par-Id$_1$>, es decir, la estructura sintáctica enunciada puede extenderse así:

```
activate(<altstep-name>((<f-par-Id1>,<act-par-desc1>), ... , (<f-par-Idn>,<act-par-descn>)))
```

El segmento de diagrama de flujo <activate-stmt> de la figura 37 define la ejecución de la instrucción activate. La ejecución consta de tres pasos. En el primer paso se crea un registro de solicitud para el altstep <function-name>. En el segundo paso se calculan los valores del parámetro efectivo y se asignan al campo correspondiente del registro de solicitud. En el tercer paso se coloca el registro de solicitud como primer elemento de la lista *DEFAULT-LIST* de la entidad que activa el comportamiento por defecto.

NOTA – Para los altstep que se activan como comportamiento por defecto sólo se permiten parámetros por valor. En la figura 37, el manejo de los parámetros por valor se describe mediante el segmento de diagrama de flujo <value-par-calculation>, que se define en 9.24.1.

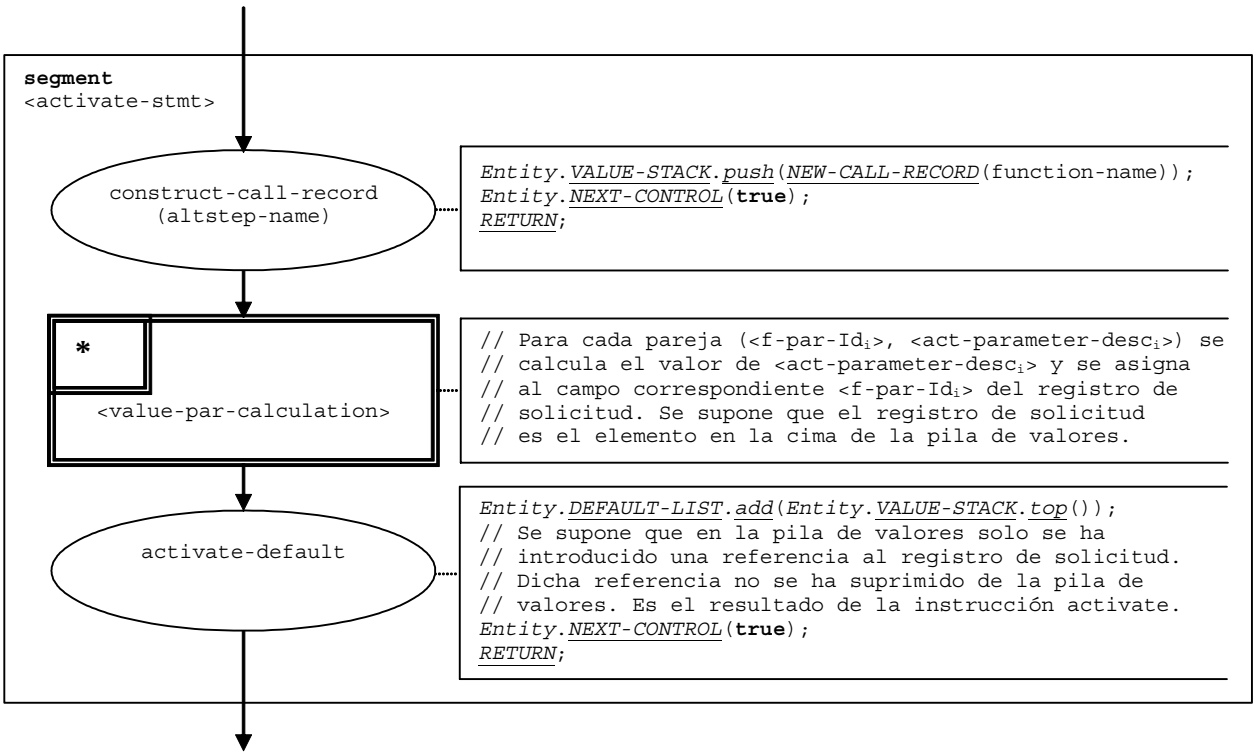


Figura 37/Z.143 – Segmento de diagrama de flujo <activate-stmt>

9.3 Instrucción alt

La instrucción **alt** es la instrucción más complicada e importante de TTCN-3. Tiene en cuenta la semántica del estado puntual y especifica la ramificación provocada por la recepción de mensajes, respuestas, solicitudes y excepciones, debidos a la expiración de temporizadores y a la finalización de componentes. Adicionalmente, la evocación del mecanismo por defecto de TTCN-3 también está relacionada con la instrucción **alt**.

En la figura 38 se muestra la representación de la instrucción **alt**. En el segmento de diagrama de flujo <receiving-branch> están ocultas las diferentes alternativas provocadas por la recepción de mensajes, respuestas, solicitudes y excepciones, debidos a la expiración de temporizadores y a la finalización de componentes.

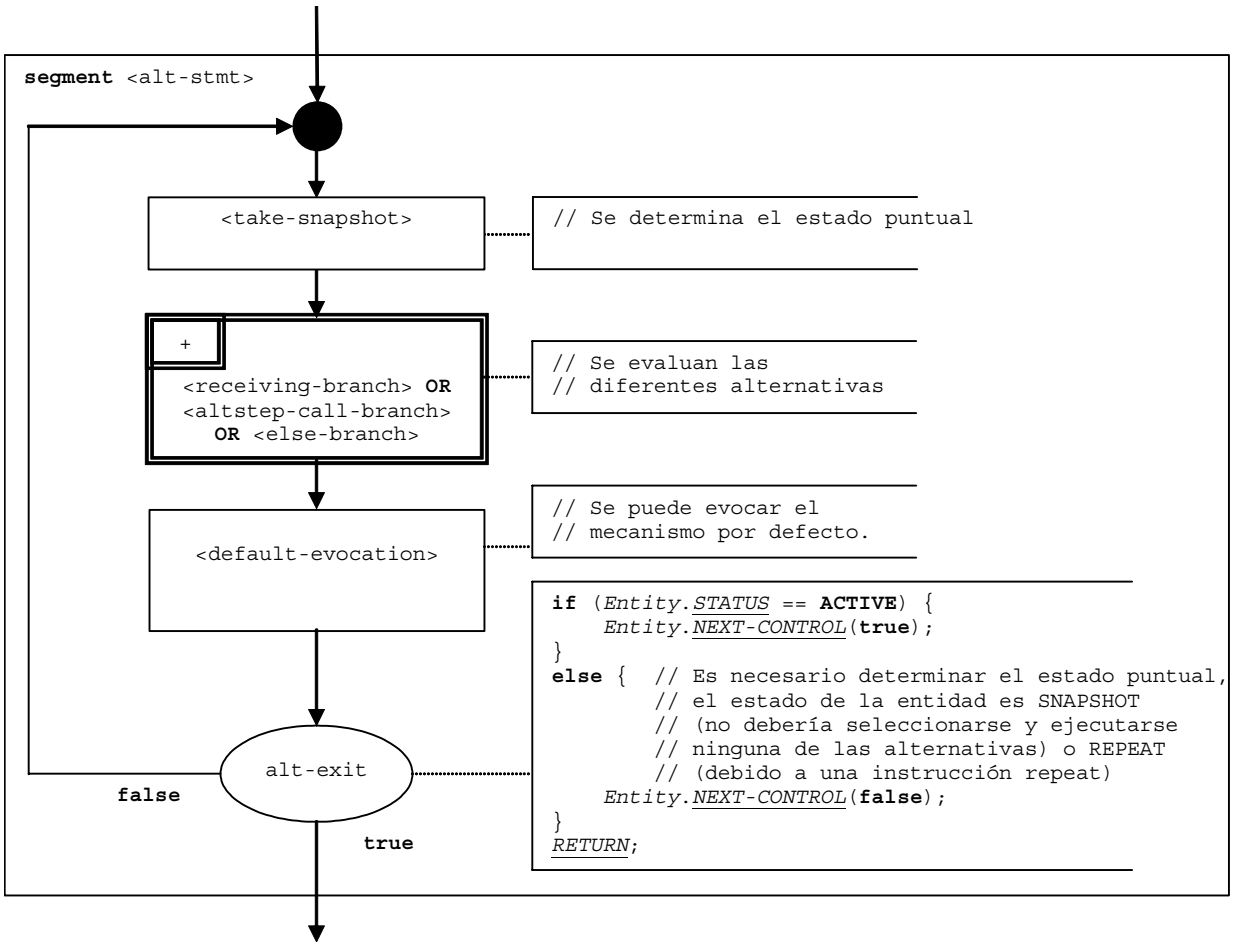


Figura 38/Z.143 – Segmento de diagrama de flujo <alt-stmt>

9.3.1 Segmento de diagrama de flujo <take-snapshot>

El segmento de diagrama de flujo `<take-snapshot>` que se muestra en la figura 39 describe el procedimiento para determinar el estado puntual. Como resultado se registran los valores de los puertos, temporizadores y componentes detenidos.

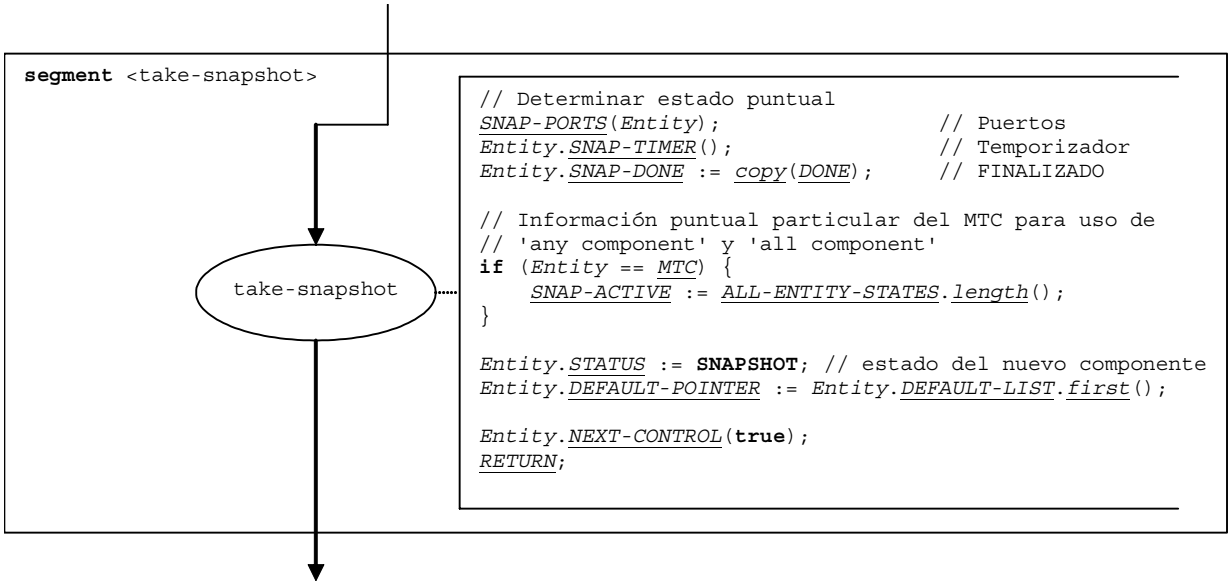


Figura 39/Z.143 – Segmento de diagrama de flujo <take-snapshot>

9.3.2 Segmento de diagrama de flujo <receiving-branch>

En la figura 40 se muestra la ejecución del segmento de diagrama de flujo <receiving-branch>.

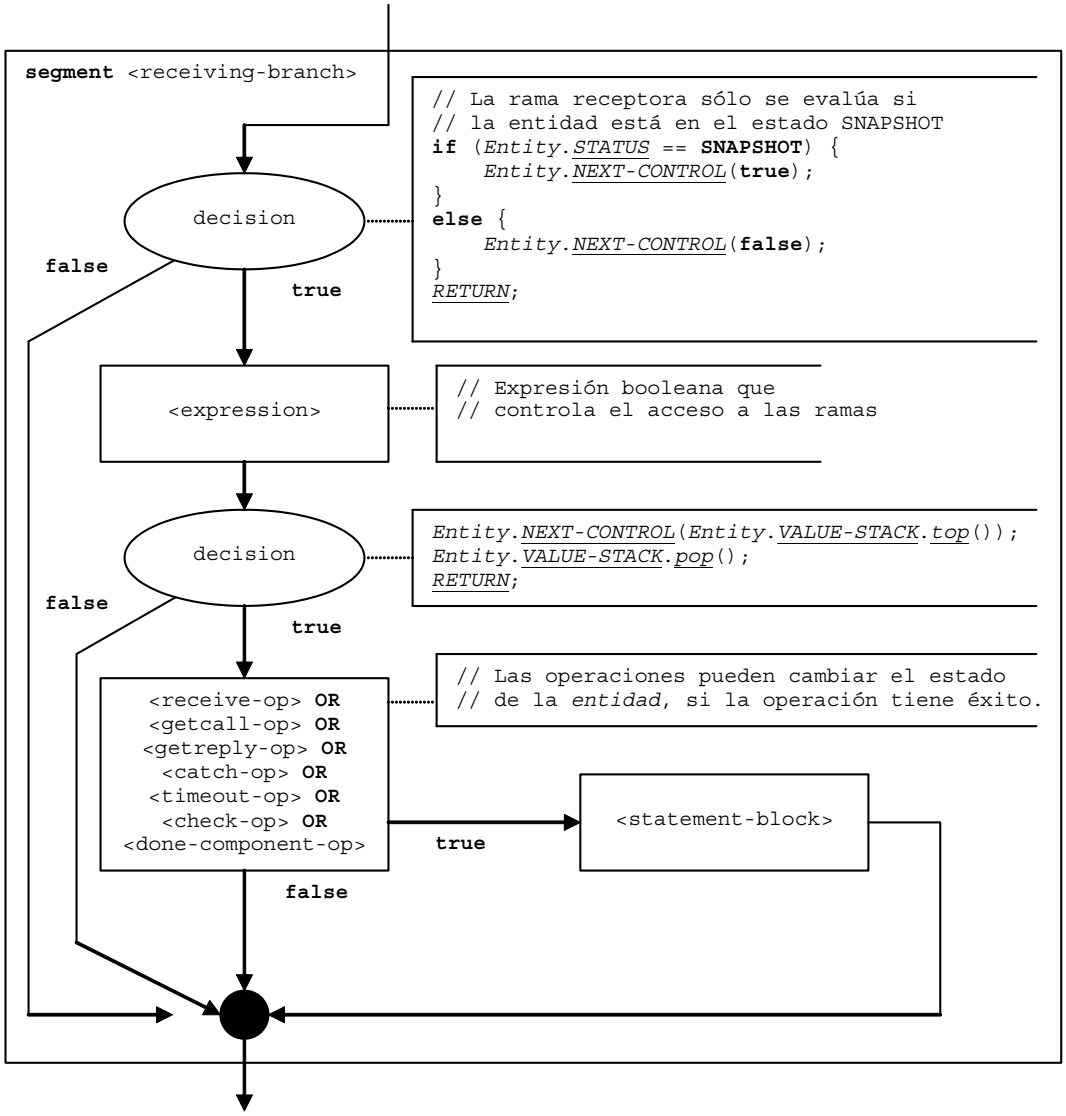


Figura 40/Z.143 – Segmento de diagrama de flujo <receiving-branch>

9.3.3 Segmento de diagrama de flujo <altstep-call-branch>

La invocación de un altstep dentro de una instrucción **`alt`** se describe mediante el segmento de diagrama de flujo `<altstep-call-branch>` de la figura 41.

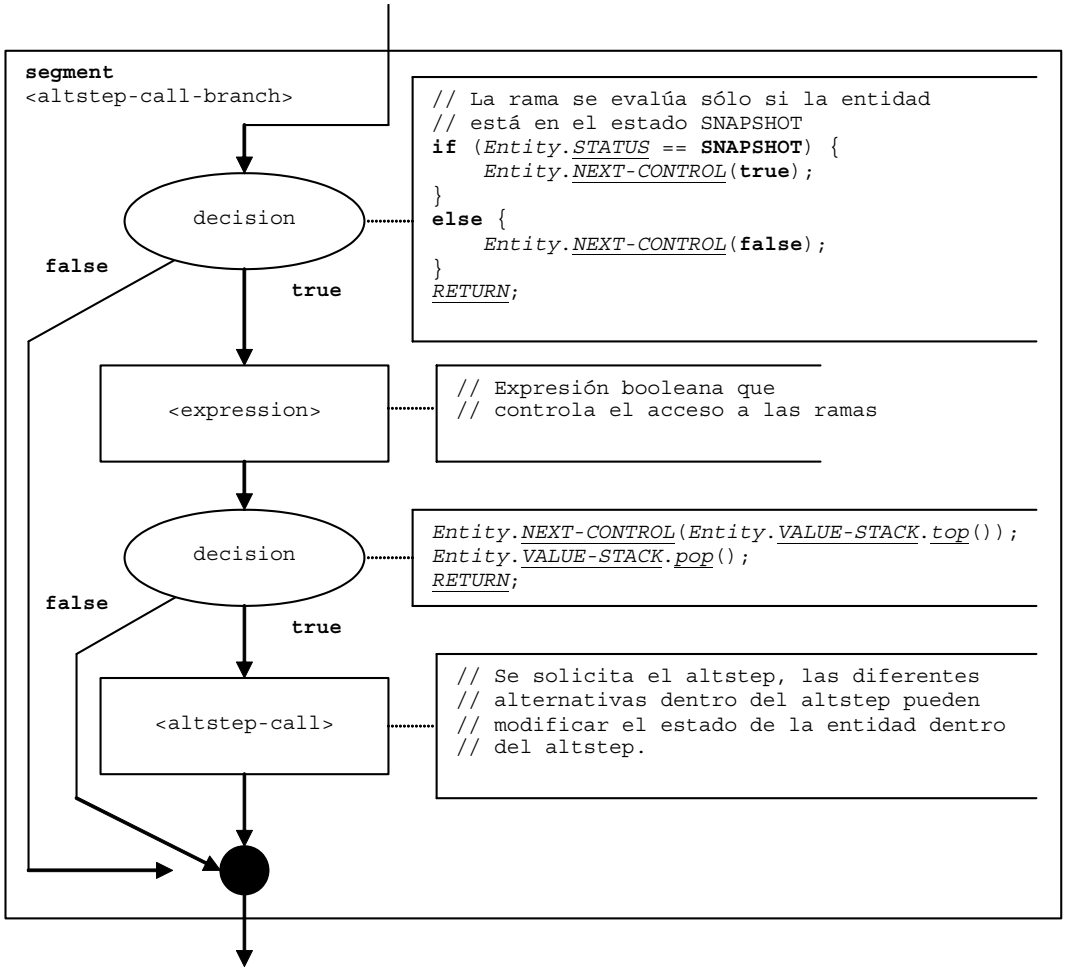


Figura 41/Z.143 – Segmento de diagrama de flujo <altstep-call-branch>

9.3.4 Segmento de diagrama de flujo <else-branch>

El segmento de diagrama de flujo `<else-branch>` de la figura 42 describe la ejecución de una rama **`else`** dentro de una instrucción **`alt`**.

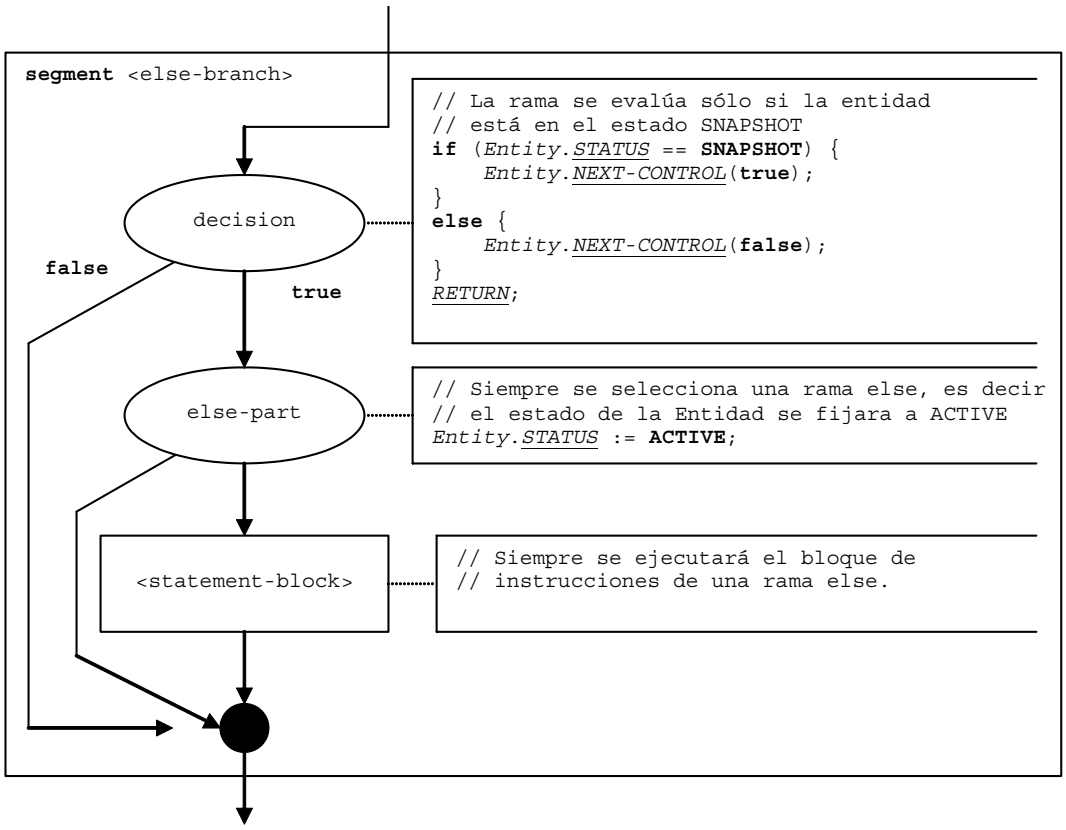


Figura 42/Z.143 – Segmento de diagrama de flujo <else-branch>

9.3.5 Segmento de diagrama de flujo <default-evocation>

El segmento de diagrama de flujo <default-evocation> de la figura 43 describe la invocación del comportamiento por defecto al final de las instrucciones **alt**.

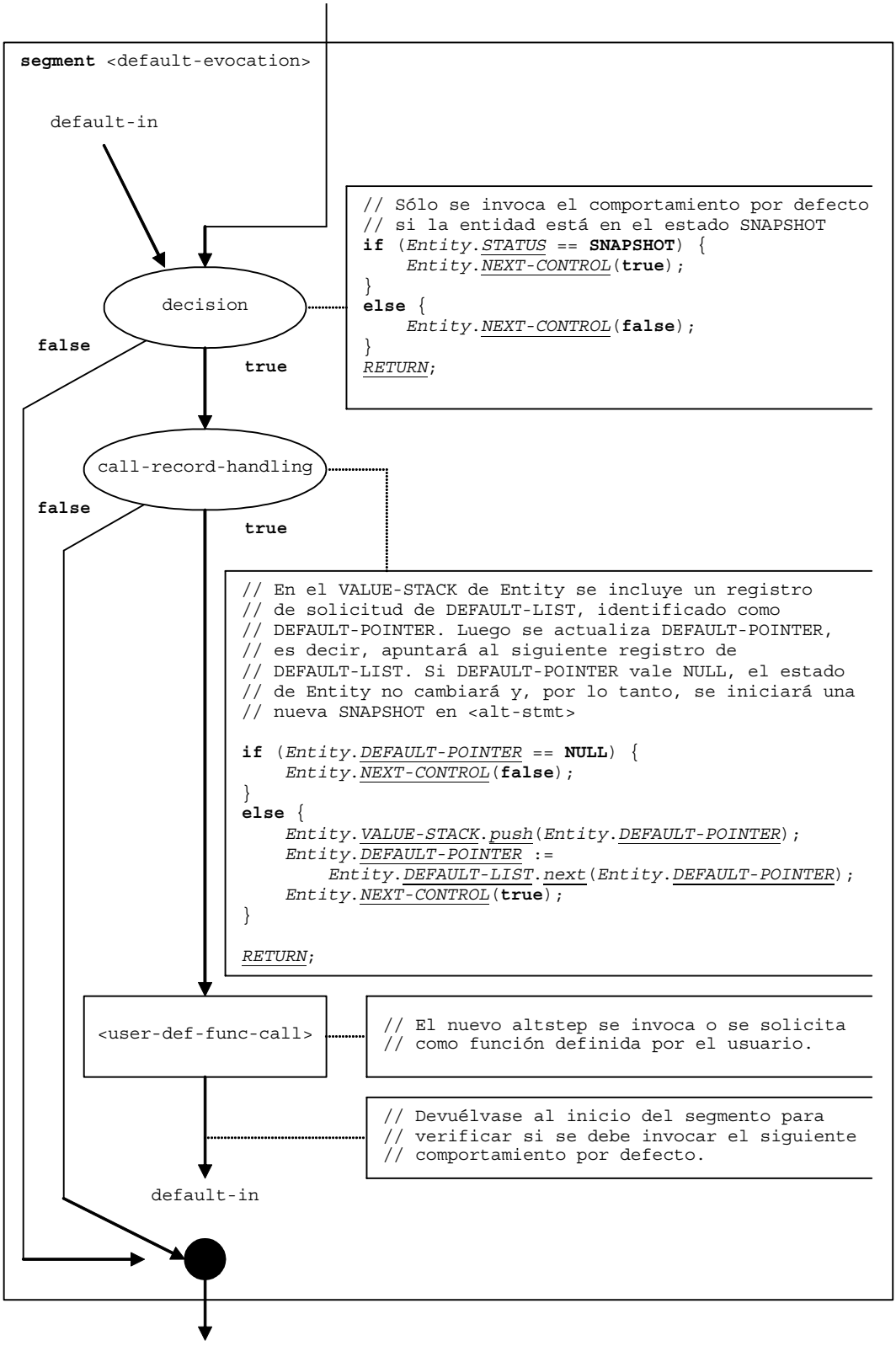


Figura 43/Z.143 – Segmento de diagrama de flujo <default-evocation>

9.4 Solicitud de altstep

Según se indica en la figura 44, las solicitudes de altstep se tratan como solicitudes de función.

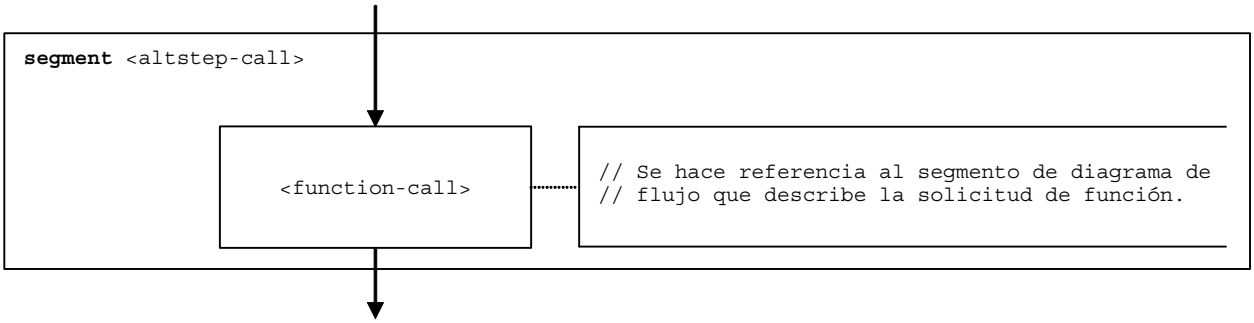


Figura 44/Z.143 – Segmento de diagrama de flujo <altstep-call>

9.5 Instrucción assignment (de asignación)

La siguiente es la estructura sintáctica de las instrucciones **assignment**:

```
<varId> := <expression>
```

El valor de la expresión <expression> se asigna a la variable <varId>. El segmento de diagrama de flujo <assignment-stmt> de la figura 45 define la ejecución de las instrucciones assignment.

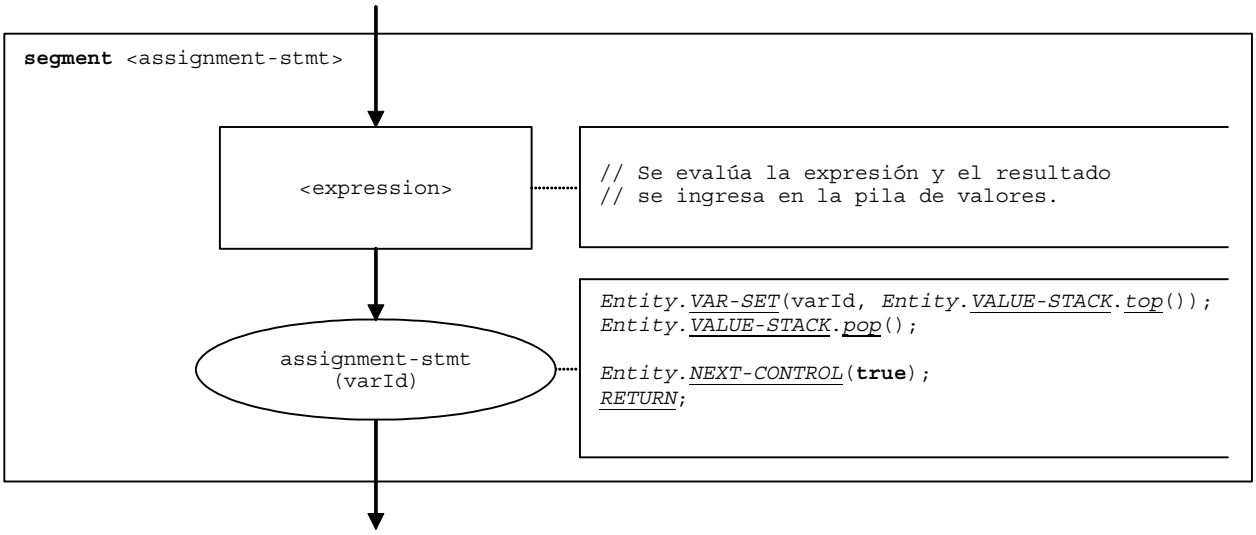


Figura 45/Z.143 – Segmento de diagrama de flujo <assignment-stmt>

9.6 Operación call (solicitud)

La siguiente es la estructura sintáctica de las operaciones **call**:

```
<portId>.call (<callSpec> [<blocking-info>]) [to <component-expression>] [<call-reception-part>]
```

El parámetro opcional <blocking-info> consiste de la palabra clave **nowait** o de la duración de una excepción de temporización. <component-expression> en la cláusula **to** se refiere a la entidad receptora. Puede ser un valor variable o el valor devuelto de una función. El parámetro opcional <call-reception-part> denota las posibles recepciones en caso de una operación **call** bloqueante.

La semántica operacional distingue entre operaciones **solicitud** *bloqueantes* y *no bloqueantes*. Una **solicitud** es no bloqueante si se utiliza la palabra clave **nowait** en la operación **call**, o si el procedimiento solicitado es no bloqueante, es decir, se define utilizando la palabra clave **noblock**. Las **solicitudes** bloqueantes poseen una parte <call-reception-part>.

El segmento de diagrama de flujo <call-op> de la figura 46 define la ejecución de las operaciones **call**. Permite hacer distinción entre las solicitudes bloqueantes y las no bloqueantes.

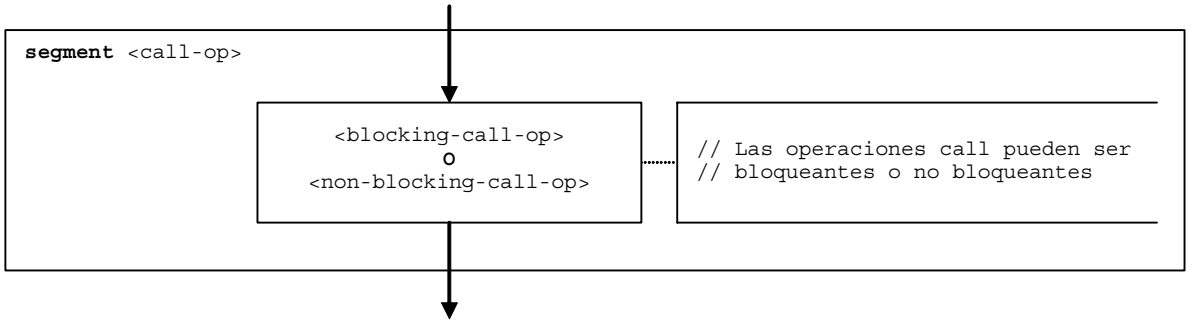


Figura 46/Z.143 – Segmento de diagrama de flujo <call-op>

Para las operaciones de solicitud bloqueantes y no bloqueantes puede especificarse una entidad receptora en la forma de expresión. En las figuras 47 y 48 se muestran las posibilidades.

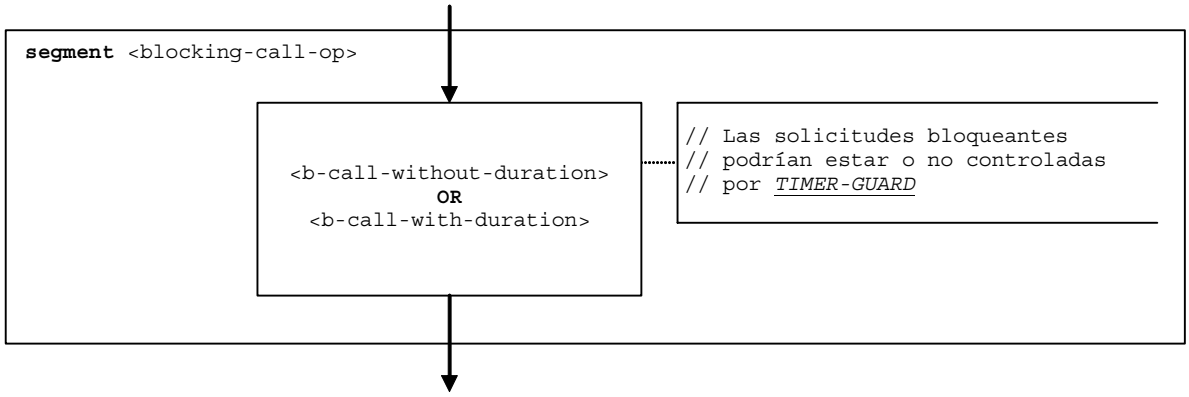


Figura 47/Z.143 – Segmento de diagrama de flujo <blocking-call-op>

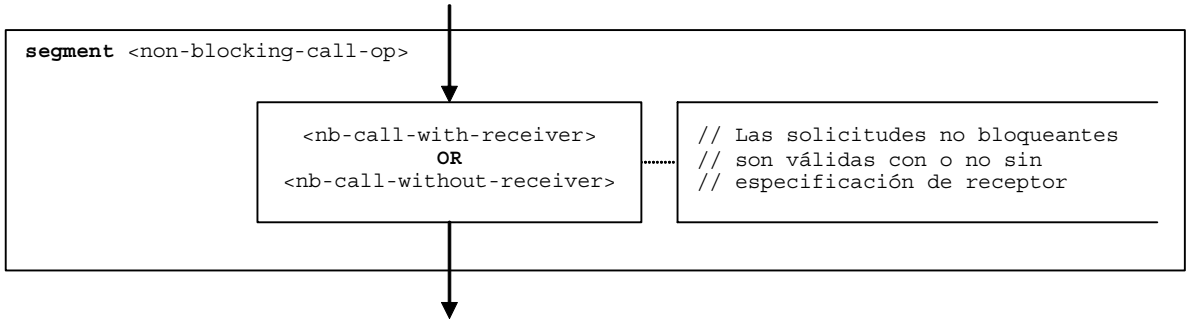


Figura 48/Z.143 – Segmento de diagrama de flujo <non-blocking-call-op>

9.6.1 Segmento de diagrama de flujo <nb-call-with-receiver>

El segmento de diagrama de flujo <nb-call-with-receiver> de la figura 49 define la ejecución de una operación **call** no bloqueante en la que se especifica el receptor en la forma de una expresión.

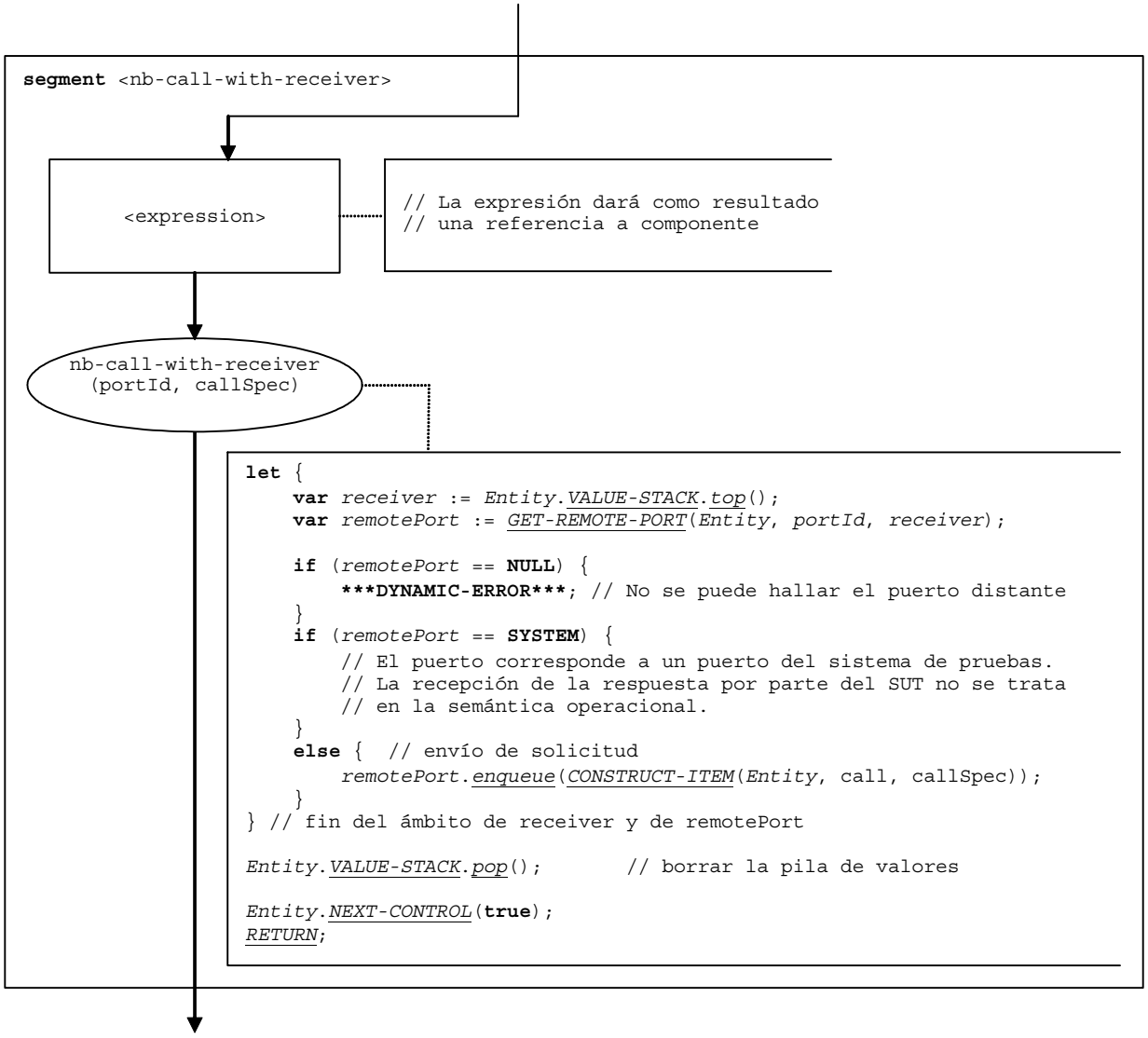


Figura 49/Z.143 – Segmento de diagrama de flujo <nb-call-with-receiver>

9.6.2 Segmento de diagrama de flujo <nb-call-without-receiver>

El segmento de diagrama de flujo <nb-call-without-receiver> de la figura 50 define la ejecución de una operación **call** no bloqueante sin cláusula **to**.

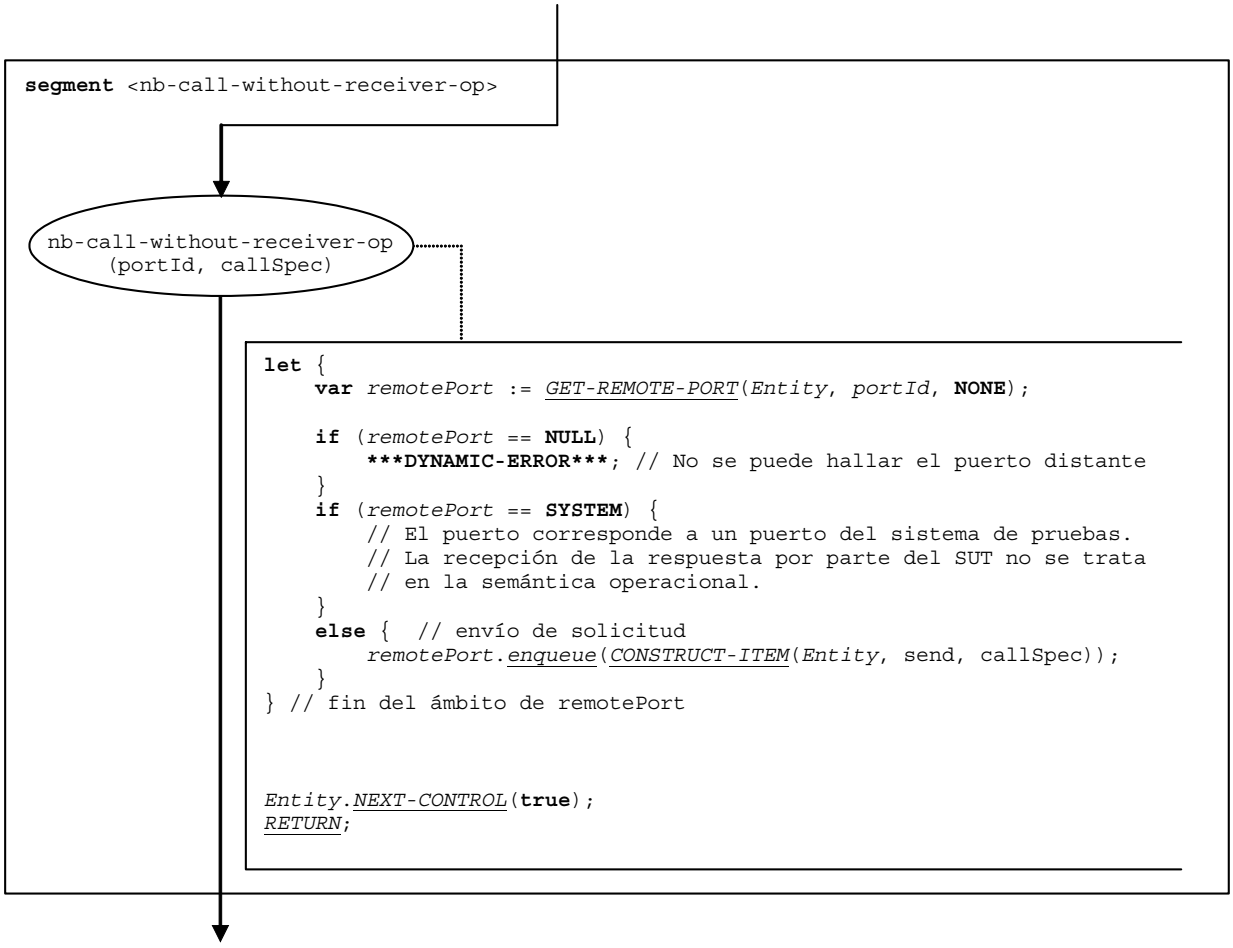


Figura 50/Z.143 – Segmento de diagrama de flujo <nb-call-without-receiver>

9.6.3 Segmento de diagrama de flujo <b-call-without-duration>

Se modelan las solicitudes bloqueantes mediante una solicitud no bloqueante seguida por el cuerpo de la solicitud encargada de tratar las respuestas y excepciones. El segmento de diagrama de flujo <b-call-without-duration> que se muestra en la figura 51 describe la ejecución de las solicitudes bloqueantes en las que no se define una duración como límite de tiempo.

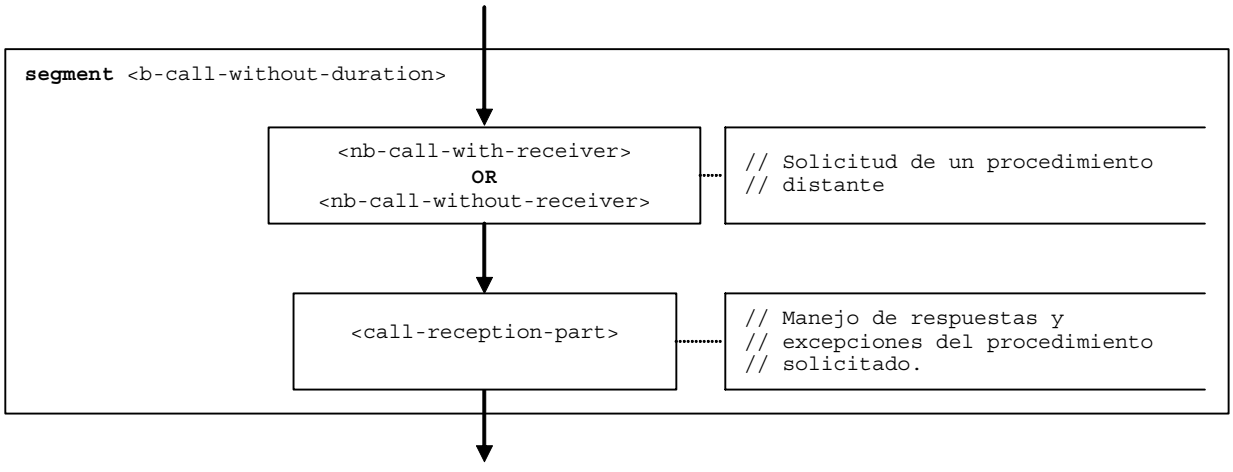


Figura 51/Z.143 – Segmento de diagrama de flujo <b-call-without-duration>

9.6.4 Segmento de diagrama de flujo <b-call-with-duration>

El segmento de diagrama de flujo <b-call-with-duration> (véase la figura 52) describe la ejecución de una solicitud bloqueante en las que se define una duración como límite de tiempo.

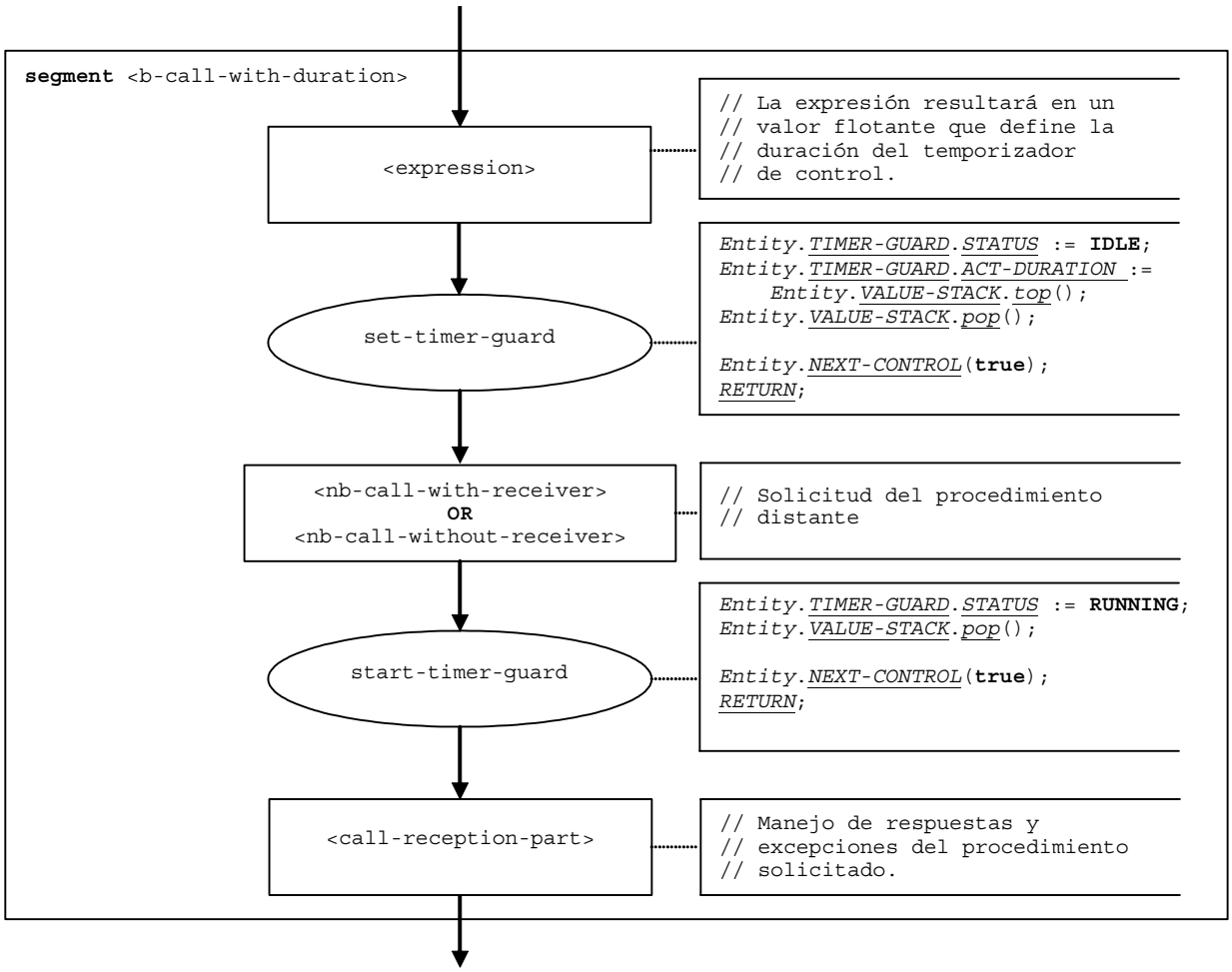


Figura 52/Z.143 – Segmento de diagrama de flujo <b-call-with-duration>

9.6.5 Segmento de diagrama de flujo <call-reception-part>

El segmento de diagrama de flujo `<call-reception-part>` (véase la figura 53) describe el manejo de respuestas, excepciones y la excepción de temporización de una operación **`call`** bloqueante.

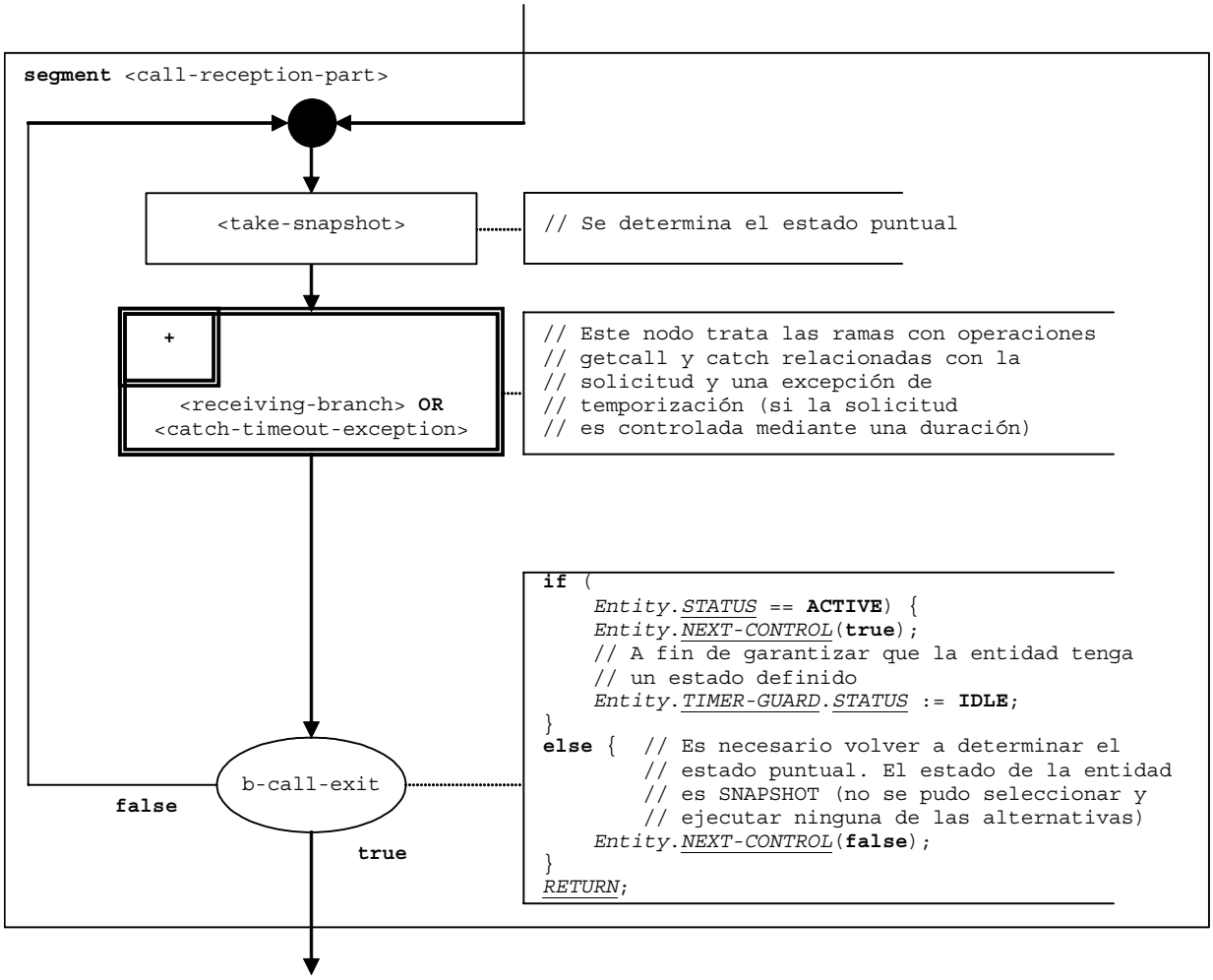


Figura 53/Z.143 – Segmento de diagrama de flujo <call-reception-part>

9.6.6 Segmento de diagrama de flujo <catch-timeout-exception>

El segmento de diagrama de flujo `<catch-timeout-exception>` (véase la figura 54) se utiliza para tratar la excepción de fin del periodo temporización de las operaciones de solicitud bloqueantes controladas por una duración.

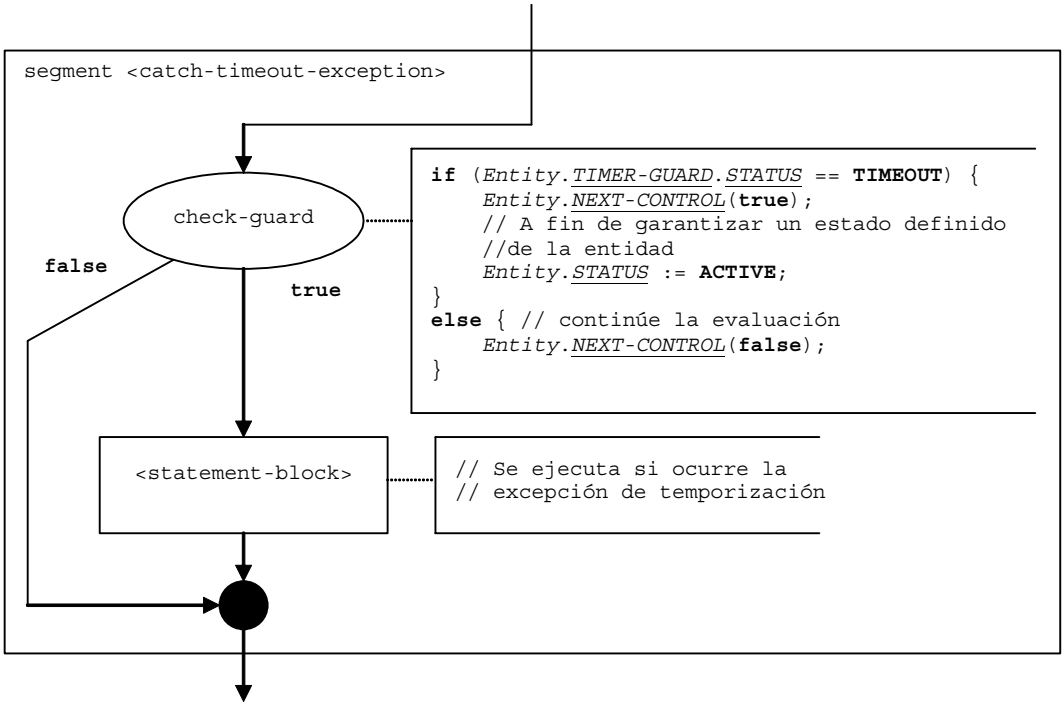


Figura 54/Z.143 – Segmento de diagrama de flujo <catch-timeout-exception>

9.7 Operación catch

La siguiente es la estructura sintáctica de las operaciones **catch**:

```
<portId>.catch (<matchingSpec>) [from <component_expression>] -> [<assignmentPart>]
```

Salvo por la palabra clave **`catch`**, esta estructura sintáctica es idéntica a la estructura sintáctica de las operaciones **`receive`**. Por lo tanto, la semántica operacional trata la operación **`catch`** de la misma manera que la operación **`receive`**. Esto se muestra también en el segmento de diagrama de flujo `<catch-op>` (figura 55), que define la ejecución de las operaciones **`catch`**. La figura hace referencia a segmentos de diagrama de flujo relacionados con la operación **`receive`** (véase 9.37).

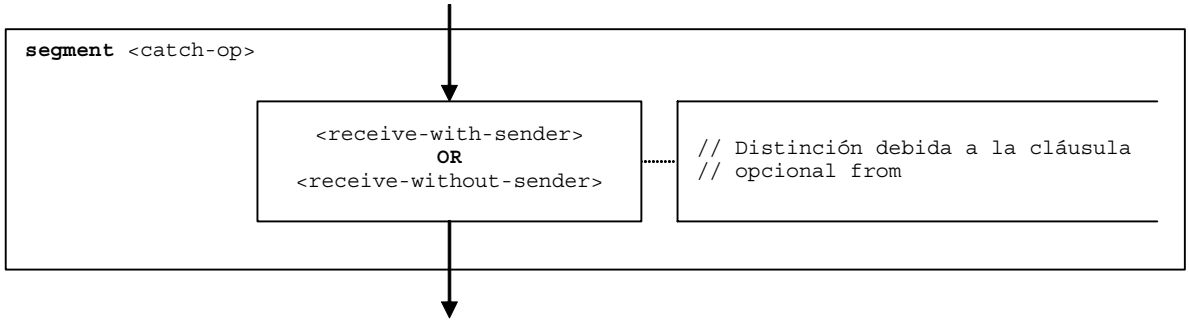


Figura 55/Z.143 – Segmento de diagrama de flujo <catch-op>

9.8 Operación check (verificar)

La siguiente es la estructura sintáctica de las operaciones **check**:

```
<portId>.check( receive|getcall|catch|getreply (<matchingSpec>)
            [from <component-expression>]) [-> <assignmentPart>]
```

La <component-expression> opcional de la cláusula **from** se refiere a la entidad emisora. Puede ser un valor variable o el valor devuelto de una función, es decir, se supone que es una expresión. El parámetro opcional <assignmentPart> indica la asignación de la información recibida siempre y cuando la información recibida corresponda con la especificación para correspondencia <matchingSpec> y con la cláusula opcional **from**.

La semántica operacional trata las operaciones **receive**, **getcall**, **catch** y **getreply** de una misma manera, es decir, las describe haciendo referencia a los mismos segmentos de diagrama de flujo <receive-with-sender> y <receive-without-sender>. La operación de verificación también trata las diferentes operaciones de una misma manera. Por lo tanto, el segmento de diagrama de flujo <check-op> de la figura 56, que define la ejecución de las operaciones **check**, también hace referencia sólo dos a segmentos de diagrama de flujo. La única diferencia con los segmentos de diagrama de flujo <receive-with-sender> y <receive-without-sender> es que no se eliminan los elementos recibidos tras la comparación.

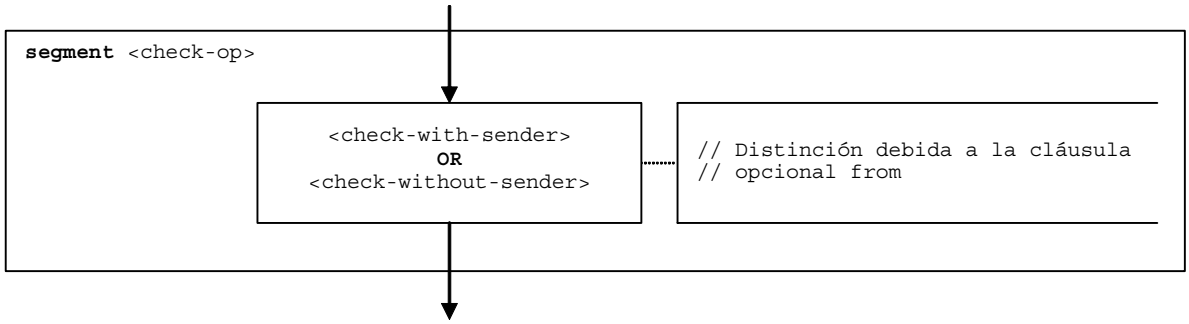


Figura 56/Z.143 – Segmento de diagrama de flujo <check-op>

9.8.1 Segmento de diagrama de flujo <check-with-sender>

El segmento de diagrama de flujo `<check-with-sender>` de la figura 57 define la ejecución de las operaciones **`check`** en las que se especifica el emisor mediante una expresión.

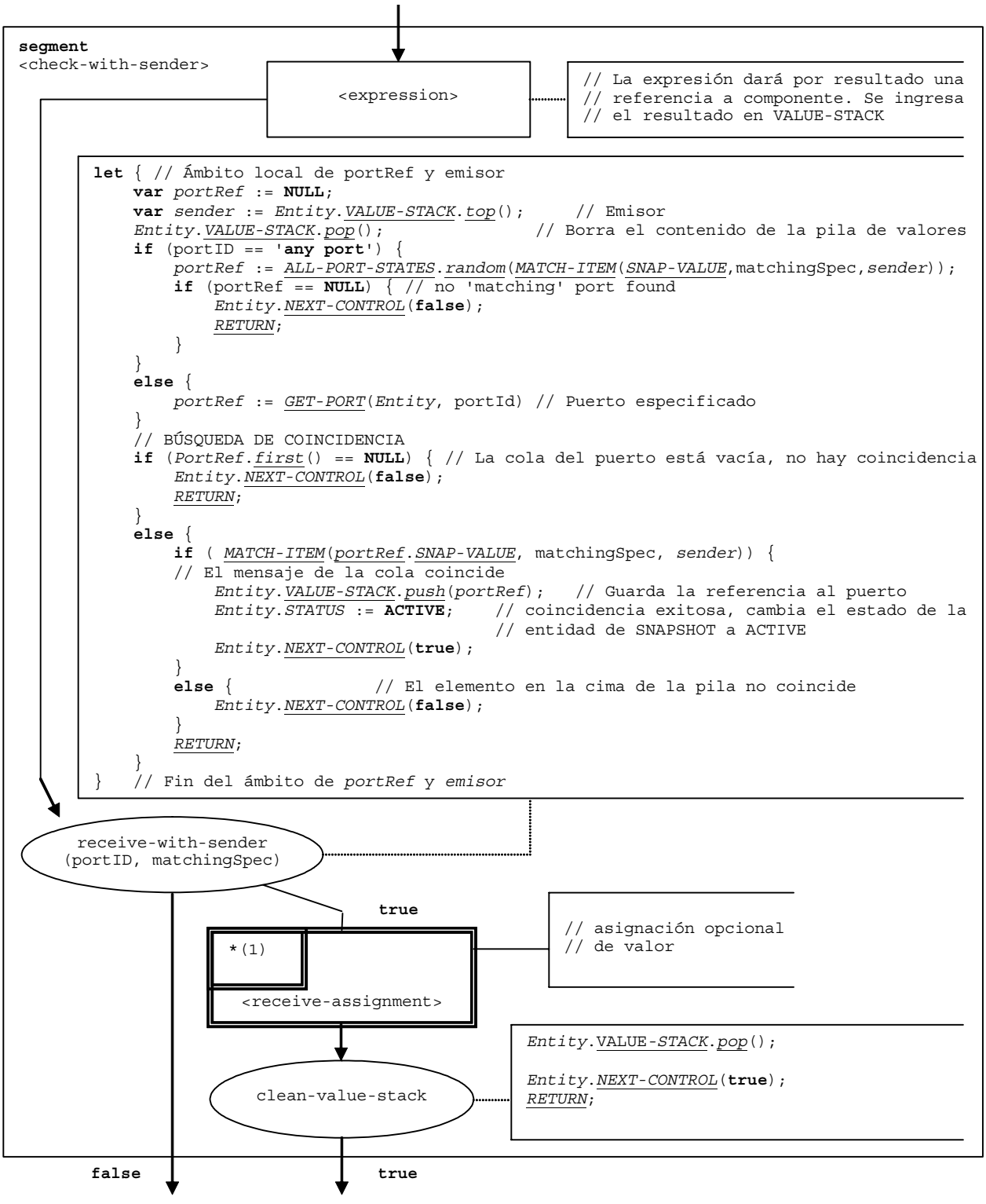


Figura 57/Z.143 – Segmento de diagrama de flujo <check-with-sender>

9.8.2 Segmento de diagrama de flujo <check-without-sender>

El segmento de diagrama de flujo <check-without-sender> de la figura 58 define la ejecución de las operaciones **check** sin cláusula **from**.

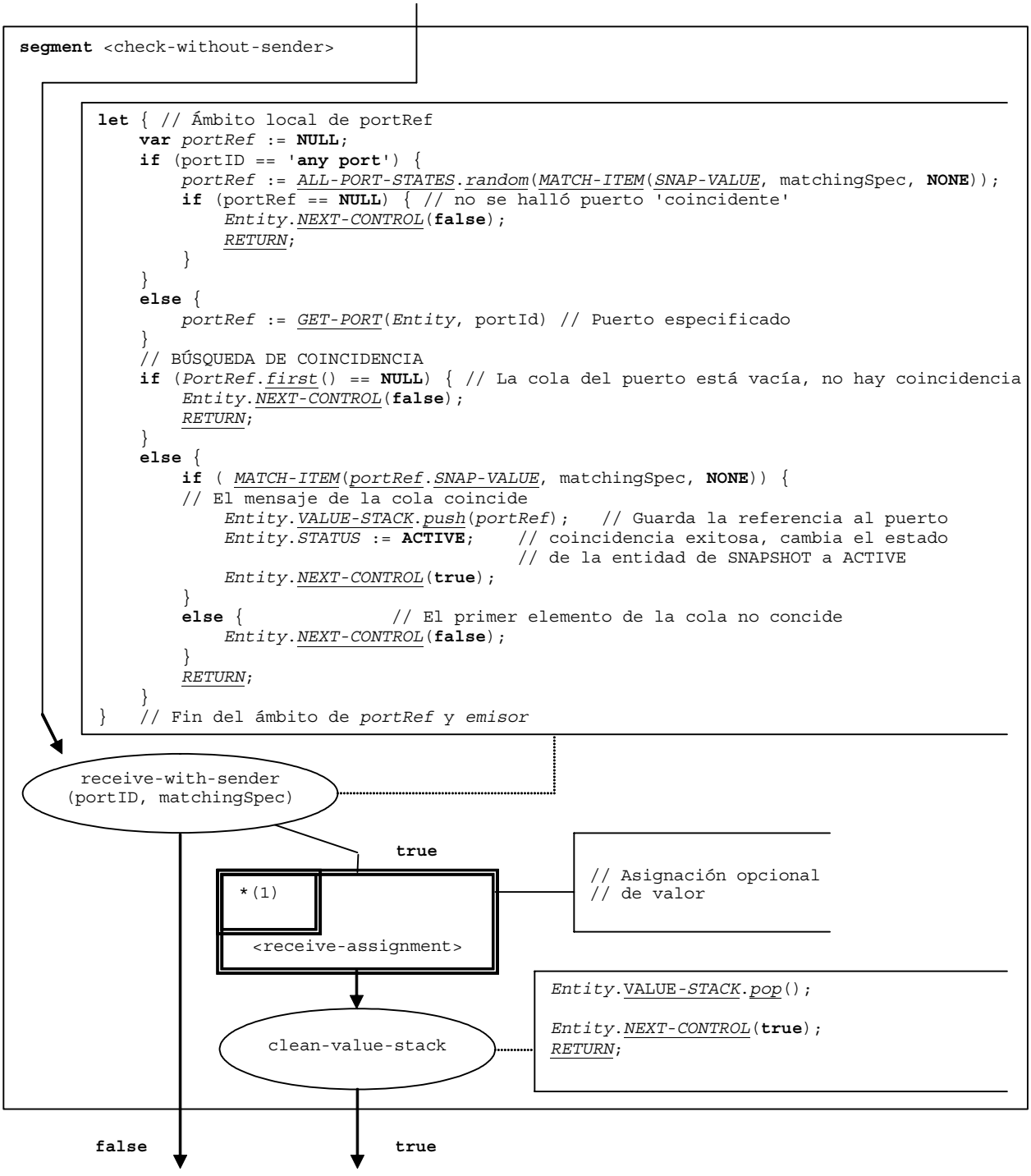


Figura 58/Z.143 – Segmento de diagrama de flujo <check-without-sender>

9.9 Operación de puerto clear

La siguiente es la estructura sintáctica de las operaciones de puerto **clear**:

```
<portId>.clear
```

El segmento de diagrama de flujo <clear-port-op> de la figura 59 define la ejecución de las operaciones de puerto **clear**.

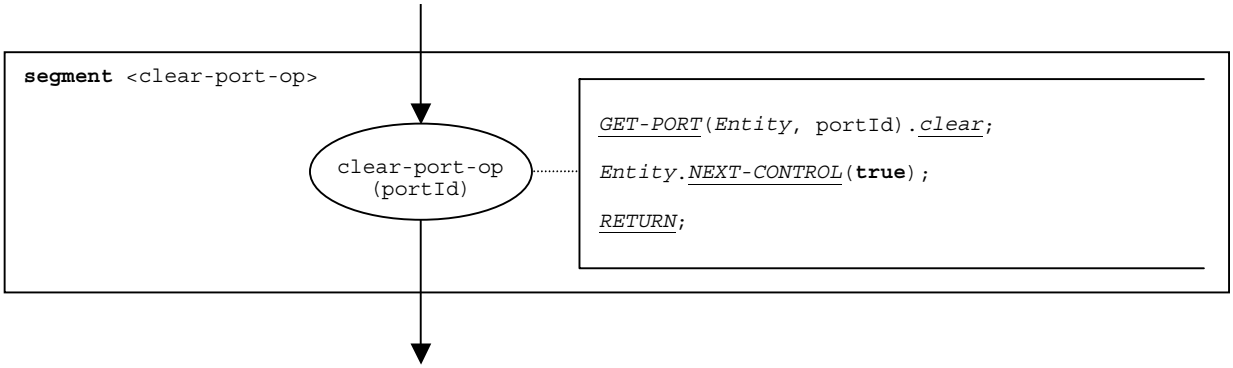


Figura 59/Z.143 – Segmento de diagrama de flujo <clear-port-op>

9.10 Operación connect

La siguiente es la estructura sintáctica de las operaciones **connect**:

```
connect(<component-expression1>:<portId1>, <component-expression2>:<portId2>)
```

Se considera que los identificadores <portId1> y <portId2> son identificadores de puerto de los componentes de prueba correspondientes. Se referencian los componentes a los que pertenecen los puertos, haciendo uso de las referencias de componente <component-expression$_1$> y <component-expression$_2$>. Las referencias pueden almacenarse en variables o pueden devolverse en funciones, es decir, son expresiones que dan como resultado referencias de componente. Se utiliza la pila de valores para almacenar las referencias de componente.

El segmento de diagrama de flujo <connect-op> de la figura 60 define la ejecución de las operaciones **connect**. En la descripción del diagrama de flujo, la primera expresión que se evalúa se refiere a <component-expression$_1$> mientras que la segunda se refiere a <component-expression$_2$>, es decir, <component-expression$_2$> se encuentra en la cima de la pila de valores cuando se ejecuta el nodo connect-op.

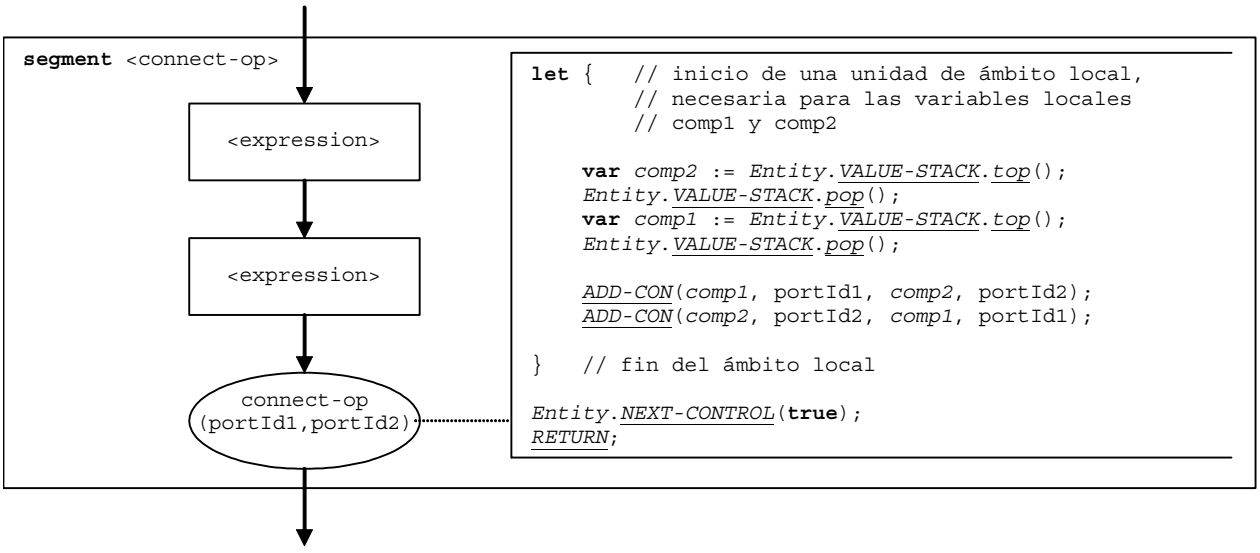


Figura 60/Z.143 – Segmento de diagrama de flujo <connect-op>

9.11 Definición constant

La siguiente es la estructura sintáctica de las definiciones de **constante**:

```
const <constType> <constId> := <constType-expression>
```

Se considera que el valor de una constante es una expresión que resulta en un valor del tipo de la constante.

NOTA – Se sustituyen las constantes globales por sus valores en un paso de procesamiento previo a la aplicación de esta semántica (véase 9.2). Las constantes locales se manejan como declaraciones de variables con inicialización. Se verificará la utilización correcta de las constantes, es decir, no habrá constantes al lado izquierdo de una asignación durante el análisis de semántica estática de los módulos de TTCN-3.

El segmento de diagrama de flujo <constant-definition> de la figura 61 define la ejecución de una declaración de constante en la que el valor de la constante se presenta como una expresión.

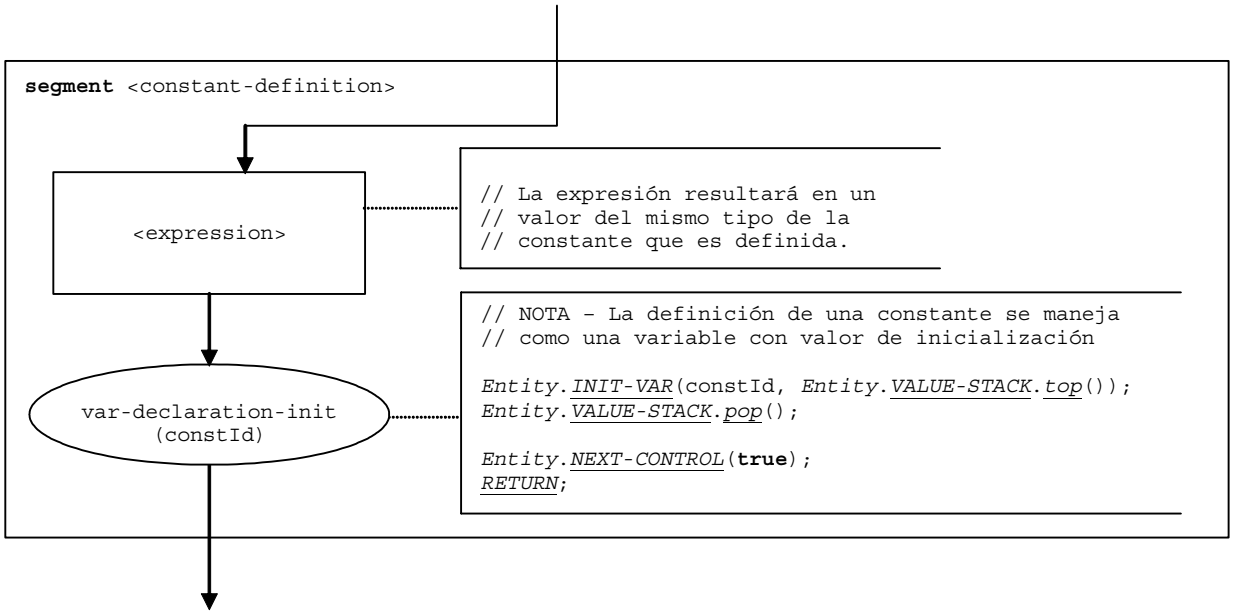


Figura 61/Z.143 – Segmento de diagrama de flujo <constant-definition>

9.12 Operación create

La siguiente es la estructura sintáctica de las operaciones **create**:

```
<componentTypeId>.create
```

El segmento de diagrama de flujo <create-op> de la figura 62 define la ejecución de las operaciones **create**.

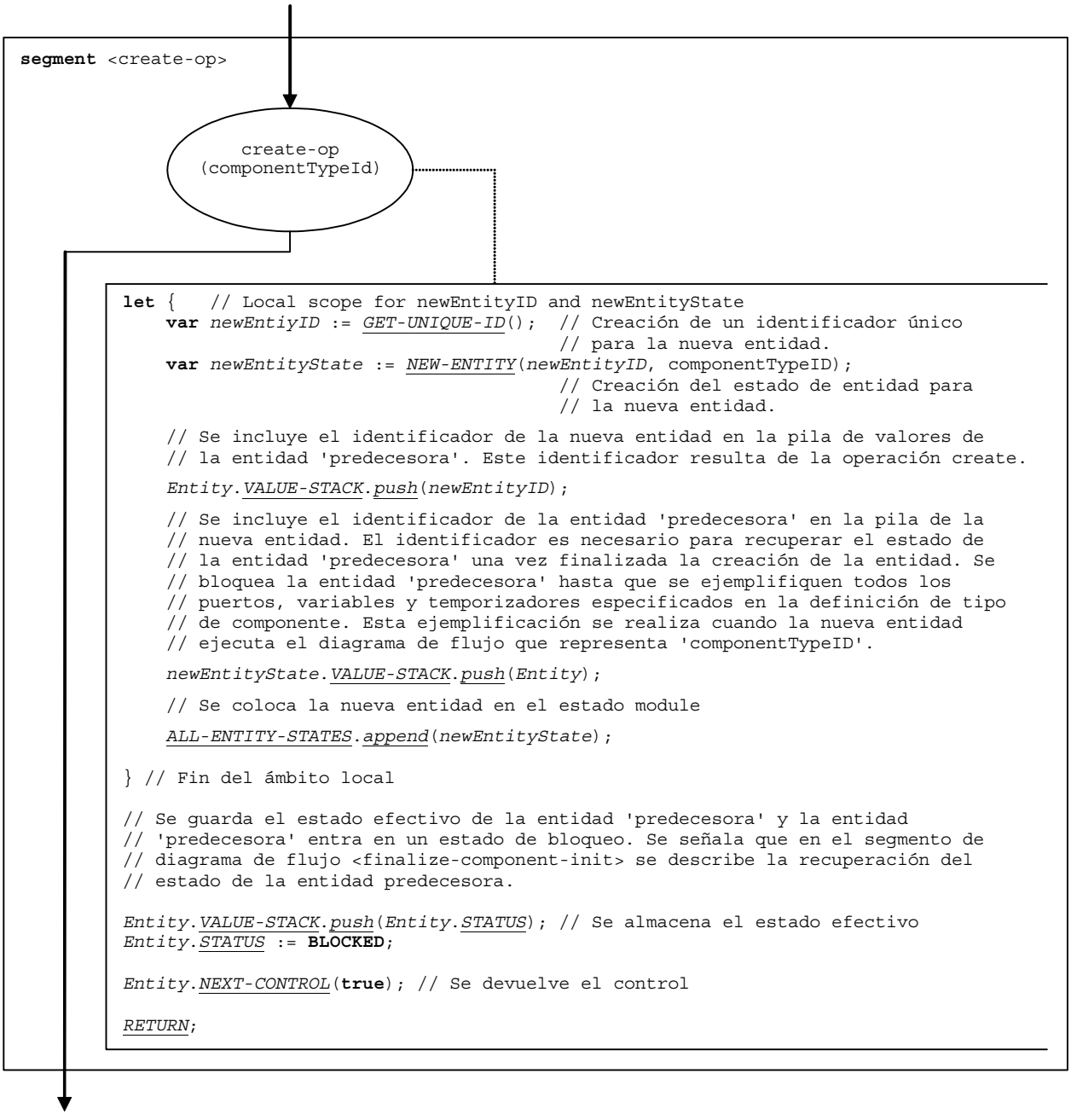


Figura 62/Z.143 – Segmento de diagrama de flujo <create-op>

9.13 Instrucción deactivate

La siguiente es la estructura sintáctica de las instrucciones **deactivate**:

```
deactivate [(<default-expression>)]
```

La instrucción **deactivate** especifica la desactivación de uno o de todas las condiciones por defecto de la entidad que ejecuta la instrucción **deactivate**. Si se desactiva una de las condiciones por defecto, el parámetro opcional <default-expression> dará por resultado una referencia por defecto que identifica la condición por defecto que

se ha de desactivar. Si se solicita la instrucción **deactivate** sin **<default-expression>**, se desactivan todas las condiciones activas por defecto.

El segmento de diagrama de flujo <deactivate-stmt> de la figura 63-a define la ejecución de la instrucción **deactivate**.

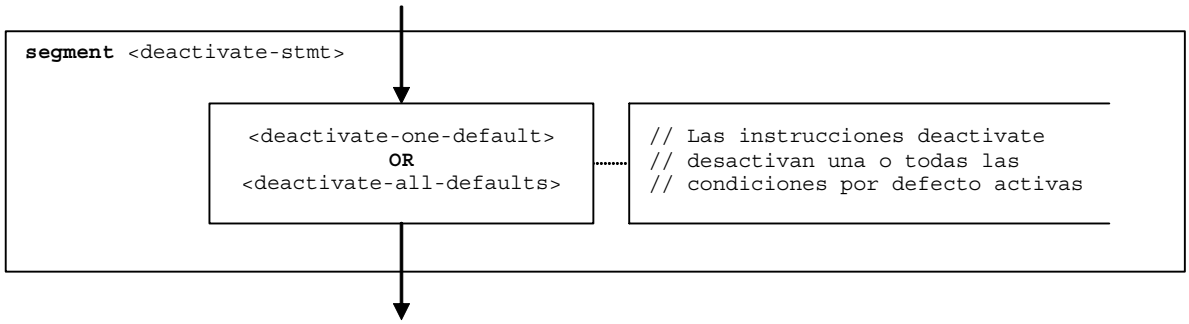


Figura 63-a/Z.143 – Segmento de diagrama de flujo <deactivate-stmt>

9.13.1 Segmento de diagrama de flujo <deactivate-one-default>

El segmento de diagrama de flujo <deactivate-one-default> de la figura 63-b especifica la desactivación de una condición por defecto activa. El valor de la expresión <default-expression> dará como resultado la referencia por defecto. La expresión puede darse en la forma de un valor variable o de una función que devuelve un valor. La instrucción **deactivate** suprime la condición por defecto especificada de la *DEFAULT-LIST* de la entidad que ejecuta la instrucción **deactivate**.

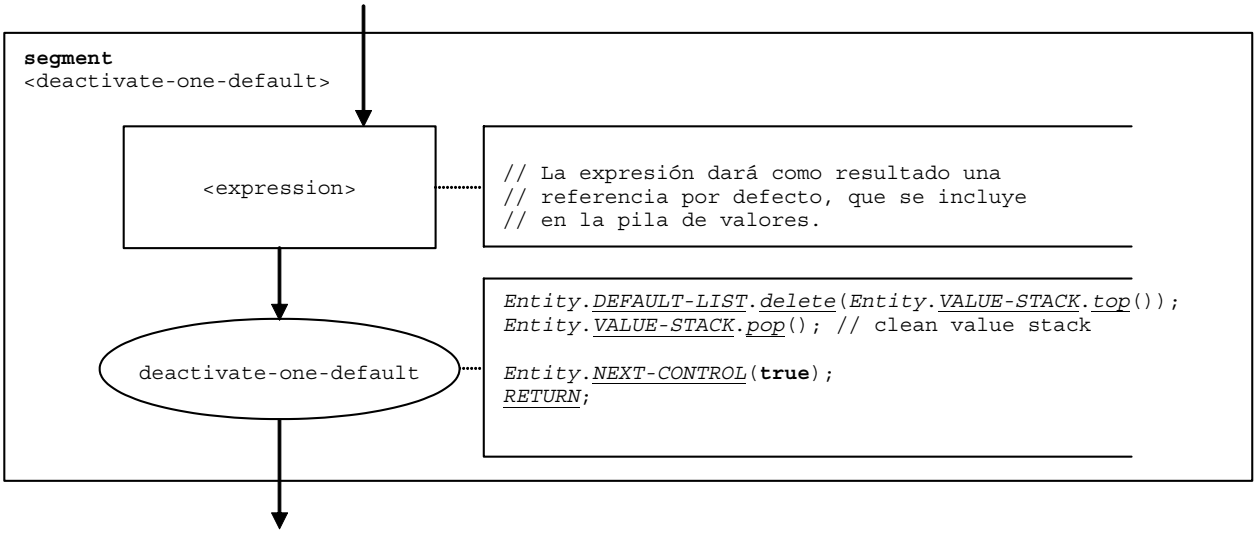


Figura 63-b/Z.143 – Segmento de diagrama de flujo <deactivate-one-default>

9.13.2 Segmento de diagrama de flujo <deactivate-all-defaults>

El segmento de diagrama de flujo <deactivate-all-defaults> de la figura 63-c especifica la desactivación de todas las opciones por defecto activas. La instrucción de desactivación borra la *DEFAULT-LIST* de la entidad que ejecuta la instrucción **deactivate**.

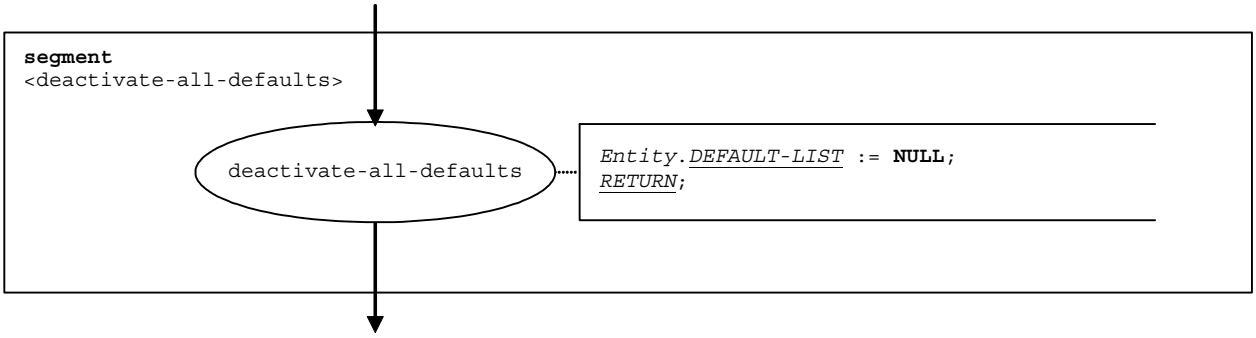


Figura 63-c/Z.143 – Segmento de diagrama de flujo <deactivate-all-defaults>

9.14 Operación disconnect

La siguiente es la estructura sintáctica de las operaciones **disconnect**:

```
disconnect(<component-expression1>:<portId1>,
    <component-expression2>:<portId2>)
```

Se considera que los identificadores <portId1> y <portId2> son identificadores de puerto de los componentes de prueba correspondientes. Se hace referencia a los componentes a los que pertenecen los puertos mediante las referencias de componente <component-expression$_1$> y <component-expression$_2$>. Las referencias pueden almacenarse en variables o pueden devolverse en funciones, es decir, son expresiones que resultan en referencias de componente. Se utiliza la pila de valores para almacenar las referencias de componente.

El segmento de diagrama de flujo <disconnect-op> que se muestra en la figura 64 define la ejecución de las operaciones **disconnect**. La primera expresión a evaluarse en el segmento de diagrama de flujo se refiere a <component-expression$_1$> y la segunda expresión a <component-expression$_2$>, es decir, <component-expression$_2$> se encuentra en la cima de la pila de valores cuando se ejecuta el nodo disconnect-op.

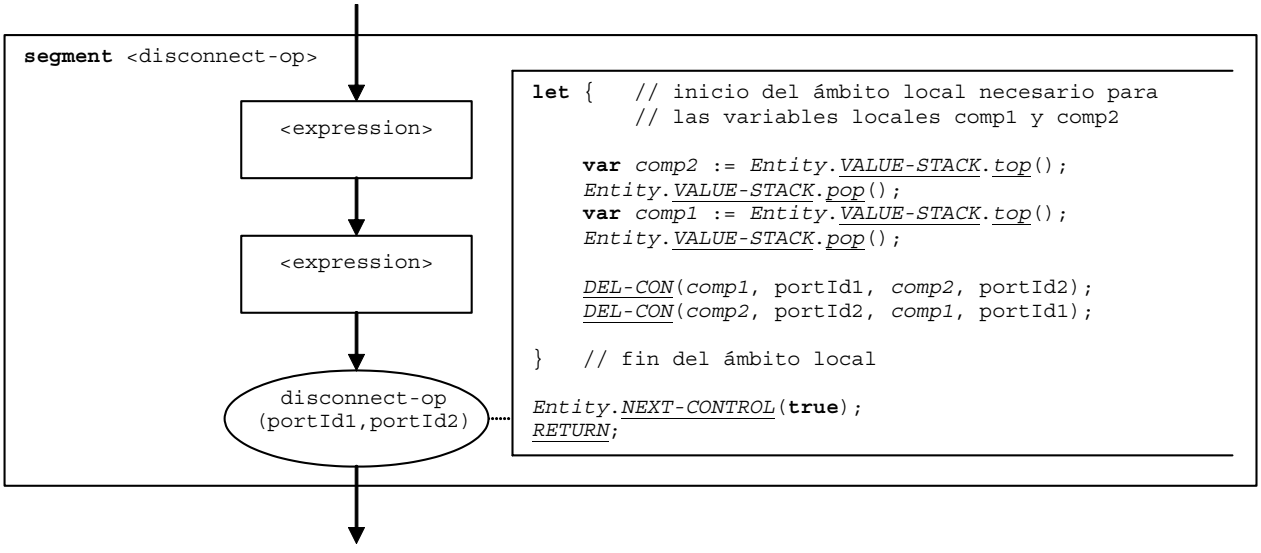


Figura 64/Z.143 – Segmento de diagrama de flujo <disconnect-op>

9.15 Instrucción do-while

La siguiente es la estructura sintáctica de las instrucciones **do-while**:

```
do <statement-block>
while (<boolean-expression>)
```

El segmento de diagrama de flujo <do-while-stmt> que se muestra en la figura 65 define la ejecución de las instrucciones **do-while**.

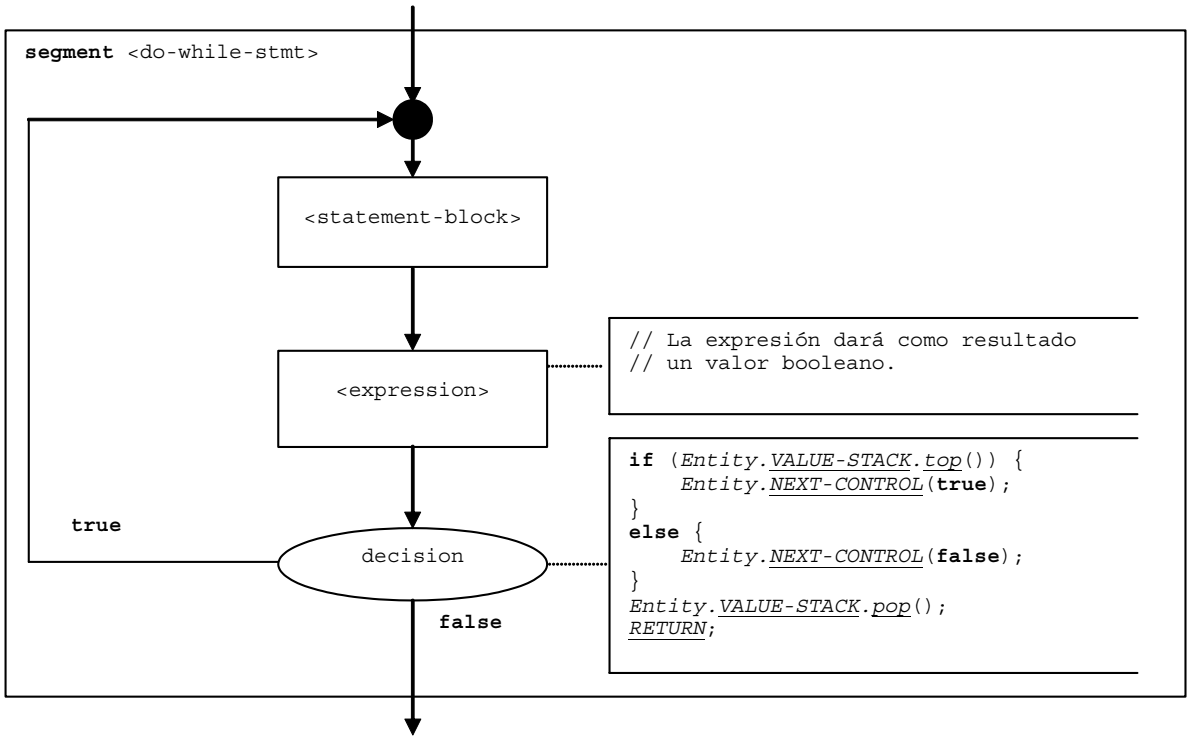


Figura 65/Z.143 – Segmento de diagrama de flujo <do-while-stmt>

9.16 Operación de componente done

La siguiente es la estructura sintáctica de las operaciones de componente **done**:

```
<component-expression>.done
```

La operación de componente **done** verifica si un componente está en ejecución o si se ha detenido. Dependiendo de si el componente verificado está en ejecución o se ha detenido, la operación **done** decide cómo ha de seguir el flujo de control. Se identifica el componente que se ha de verificar mediante una referencia de componente. Puede almacenarse la referencia en una variable o puede devolverse por una función, es decir, es una expresión. En aras de simplicidad, se considera que las palabras clave '**all component**' y '**any component**' son expresiones especiales.

El segmento de diagrama de flujo <done-op> de figura 66 define la ejecución de las operaciones de componente **done**.

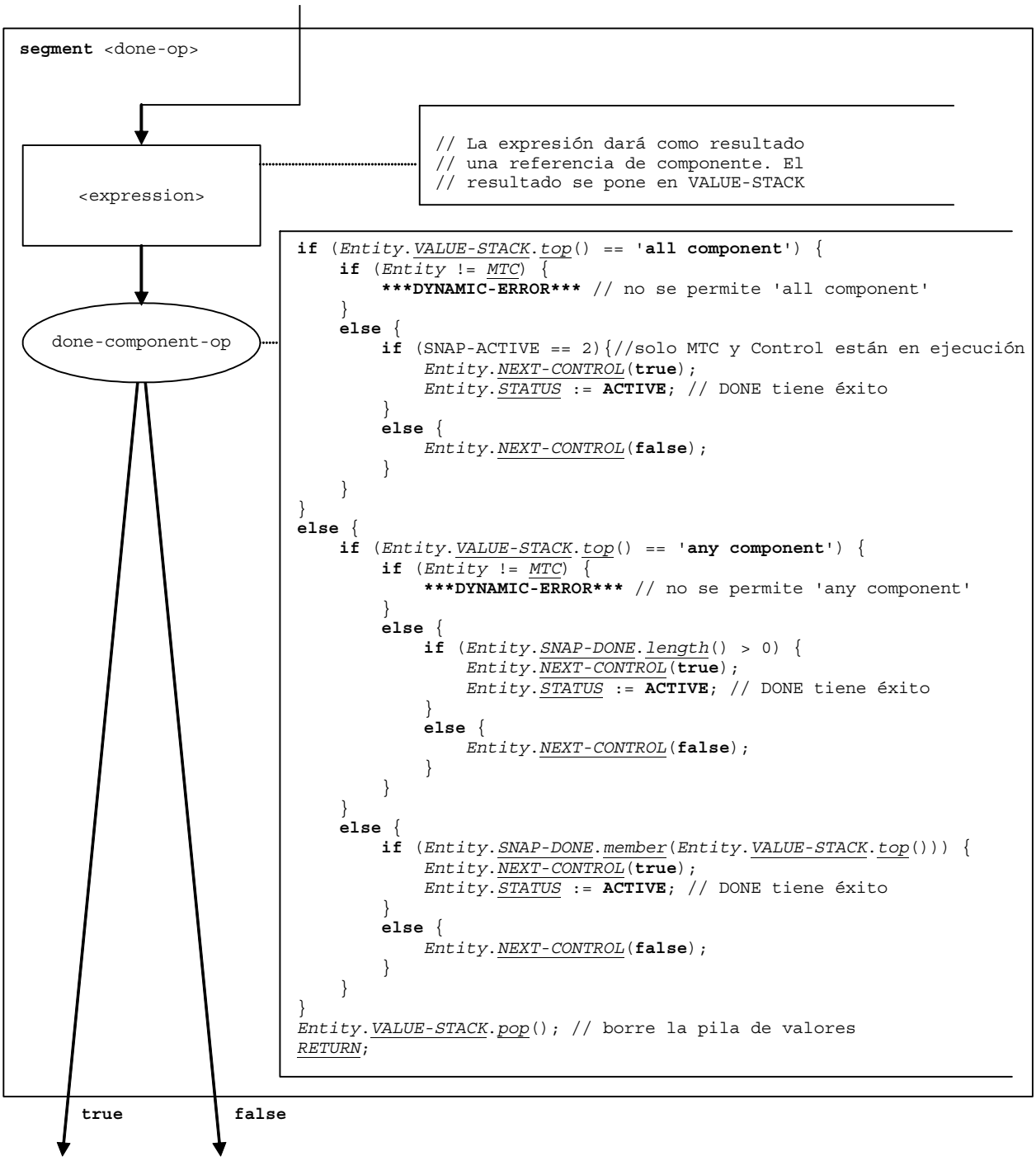


Figura 66/Z.143 – Segmento de diagrama de flujo <done-component-op>

9.17 Instrucción execute

La siguiente es la estructura sintáctica de las instrucciones **execute**:

execute(<testCaseId>([<act-par$_1$>, ... , <act-par$_n$>)]) [, <float-expression>])

La instrucción **execute** describe la ejecución del caso de prueba <testCaseId> con los parámetros efectivos (opcionales) <act-par$_1$>, … , <act-par$_n$>. Opcionalmente, se podría vigilar la instrucción execute durante un tiempo dado que se provee en la forma de una expresión que da como resultado un valor tipo **float**. Si el caso de prueba no devuelve un veredicto en el tiempo especificado, ocurre una excepción de temporización, se detiene el caso de prueba y se devuelve un veredicto de **error**.

NOTA – La semántica operacional modela la detención del caso de prueba mediante la detención del MTC. En realidad, podrían ser más adecuados otros mecanismos.

Si no ocurre una excepción de temporización, se crea el MTC, el ejemplar de control (que representa la parte de control del módulo TTCN-3) se bloquea hasta que finalice el caso de prueba y se otorga el control de flujo al MTC durante el resto de la ejecución del caso de prueba. El control de flujo regresa a la instancia de control una vez finalice el MTC.

El segmento de diagrama de flujo `<execute-stmt>` de la figura 67 define la ejecución de las instrucciones **`execute`**.

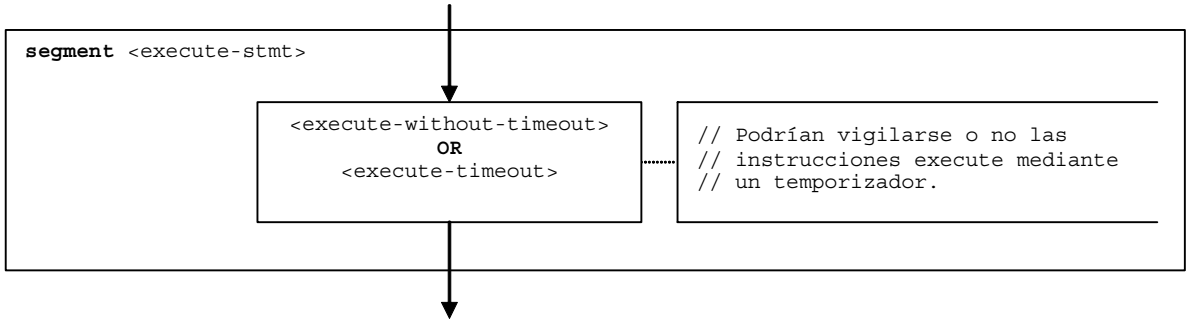


Figura 67/Z.143 – Segmento de diagrama de flujo <execute-stmt>

9.17.1 Segmento de diagrama de flujo <execute-without-timeout>

La ejecución de los casos de prueba se inicia con la creación del **`mtc`**. Se inicia el mtc con el comportamiento definido en la definición del caso de prueba. Luego, el control de módulo espera hasta que finalice el caso de prueba. Pueden describirse la creación e inicio del MTC utilizando las instrucciones **`create`** y **`start`**:

```
var mtcType MyMTC := mtcType.create;
MyMTC.start(TestCaseName(P1...Pn));
```

El segmento de diagrama de flujo `<execute-without-timeout>` de la figura 68 define la ejecución de las instrucciones **`execute`** sin que ocurra una excepción de temporización, mediante la utilización de los segmentos de diagrama de flujo de las operaciones **`create`** y **`start`**.

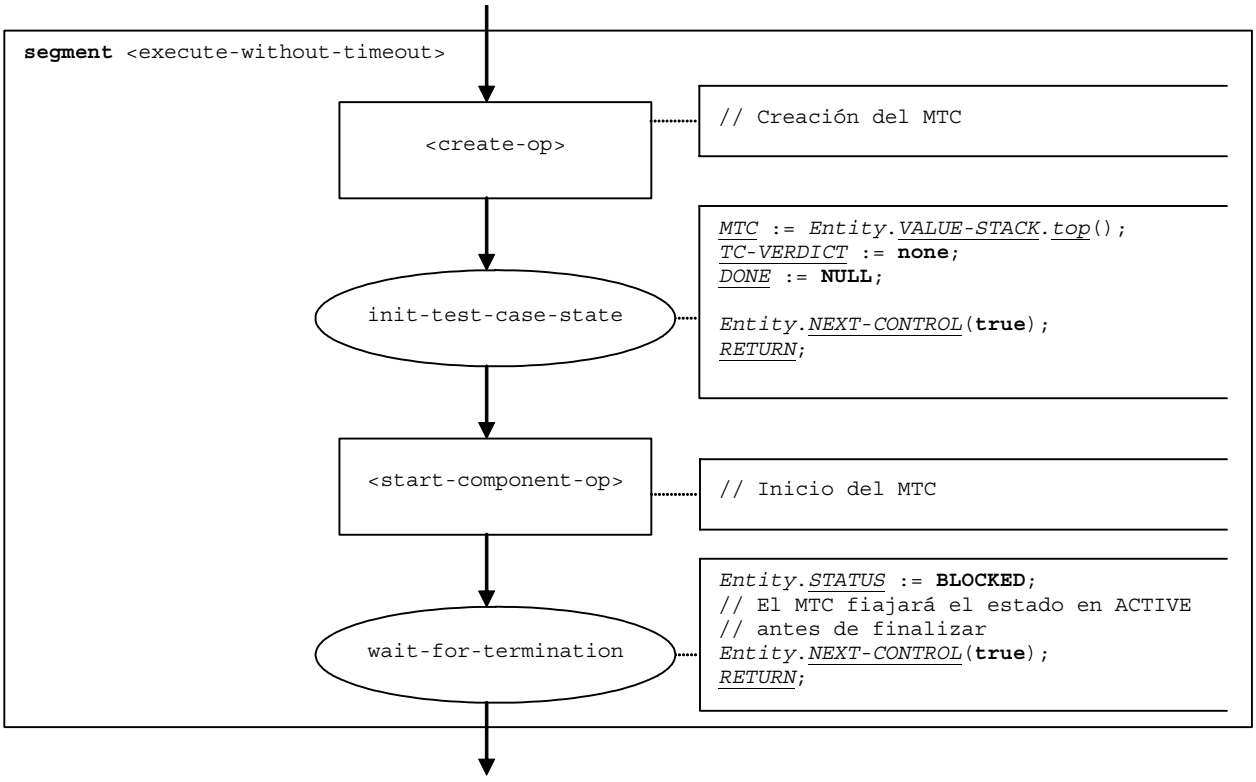


Figura 68/Z.143 – Segmento de diagrama de flujo <execute-without-timeout>

9.17.2 Segmento de diagrama de flujo <execute-timeout>

El segmento de diagrama de flujo `<execute-timeout>` de la figura 69 define la ejecución de una instrucción **`execute`** vigilada por un valor de temporizador. El segmento de diagrama de flujo también modela la creación e inicio del MTC mediante una operación **`create`** y **`start`**. Adicionalmente, *TIMER-GUARD* vigila la finalización.

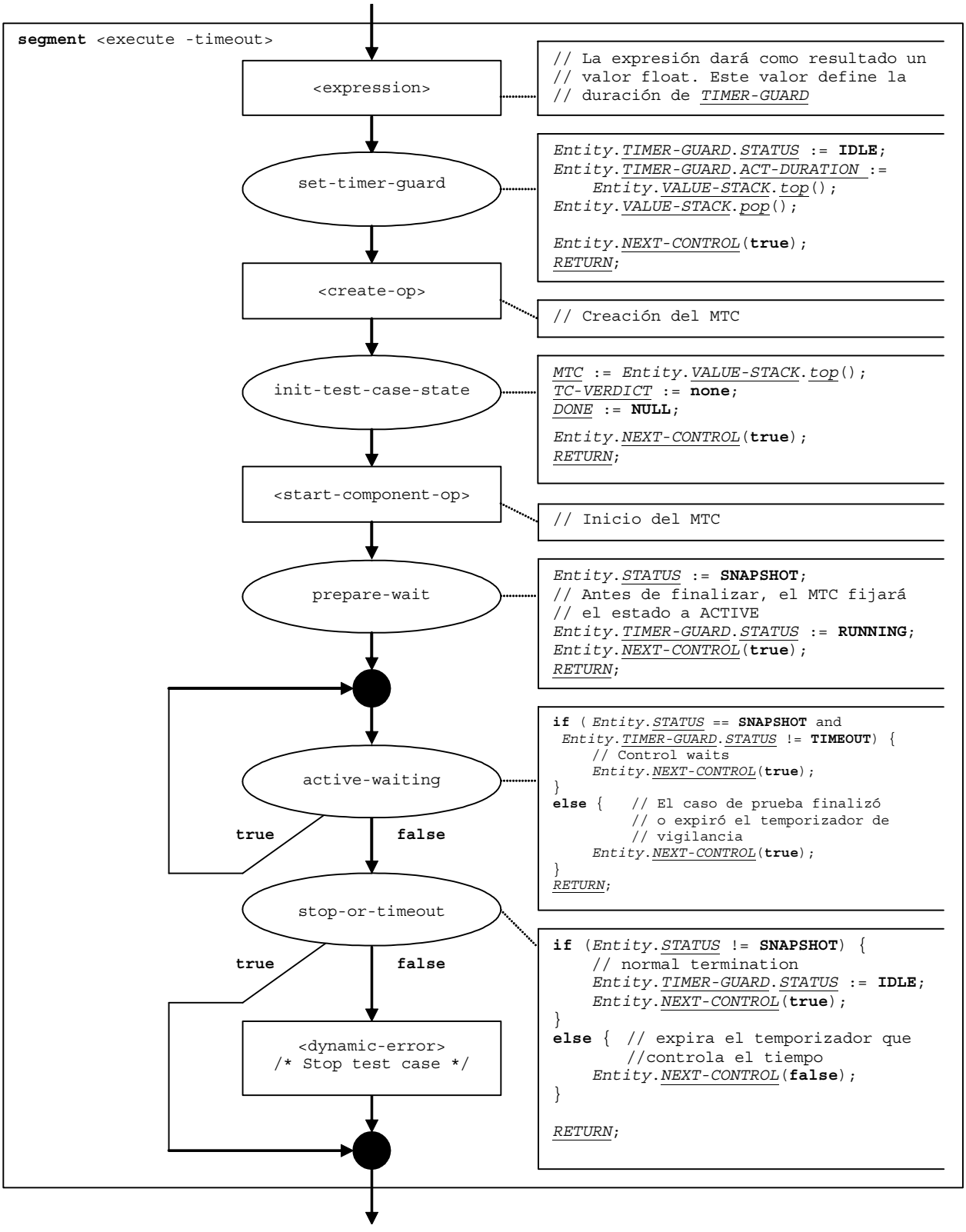


Figura 69/Z.143 – Segmento de diagrama de flujo <execute-timeout>

9.18 Expresiones

Para el manejo de las expresiones es necesario distinguir los siguientes cuatro casos:

a) la expresión es un valor literal (o una constante);

b) la expresión es una variable;

c) la expresión es un operador aplicado a uno o varios operandos;

d) la expresión es una solicitud de operación o de función.

La siguiente es la estructura sintáctica de las expresiones:

```
<lit-val> | <var-val> | <func-op-call> | <operand-appl>
```

donde:

`<lit-val>`	indica un valor literal;
`<var-val>`	indica un valor variable;
`<func-op-call>`	indica una solicitud de función o de operación;
`<operator-appl>`	indica la aplicación de operadores aritméticos como +, –, **`not`**, etc.

El segmento de diagrama de flujo `<expression>` de la figura 70 define la ejecución de una expresión.

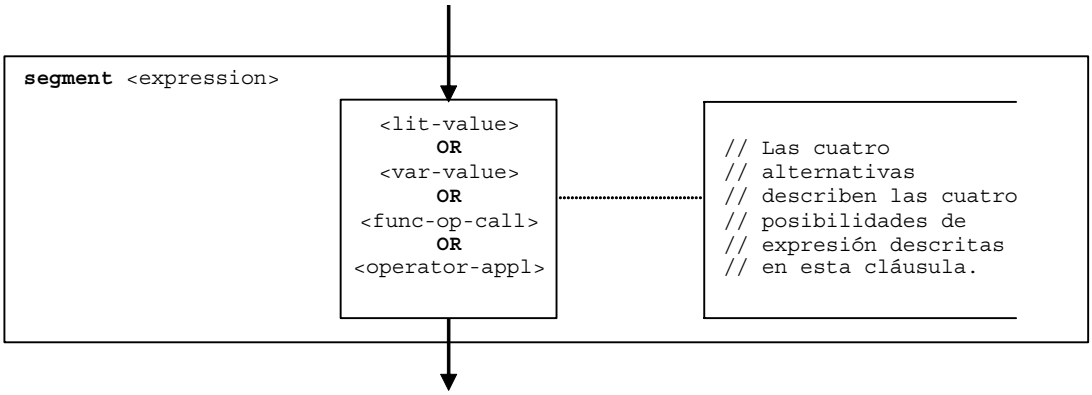


Figura 70/Z.143 – Segmento de diagrama de flujo <expression>

9.18.1 Segmento de diagrama de flujo <lit-value>

El segmento de diagrama de flujo `<lit-value>` de la figura 71 pone un valor literal en la pila de valores de una entidad.

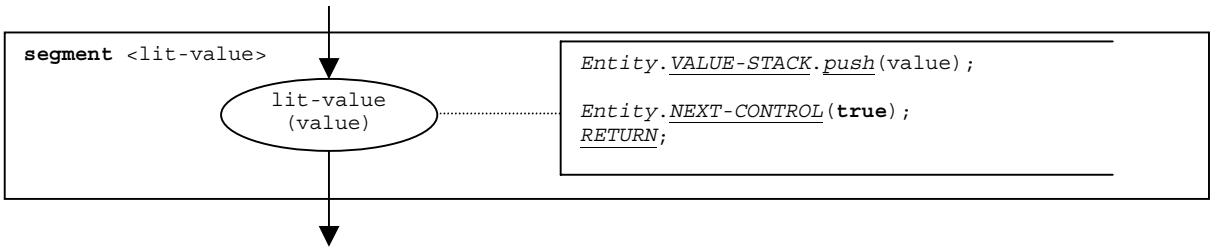


Figura 71/Z.143 – Segmento de diagrama de flujo <lit-value>

9.18.2 Segmento de diagrama de flujo <var-value>

El segmento de diagrama de flujo `<var-value>` de la figura 72 pone el valor de una variable en la pila de valores de una entidad.

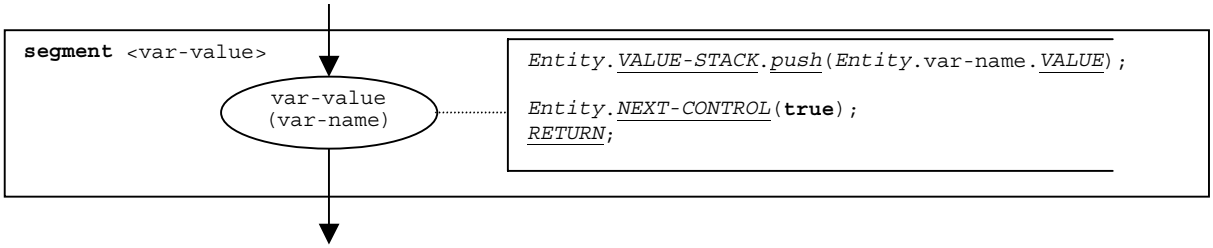


Figura 72/Z.143 – Segmento de diagrama de flujo <var-value>

9.18.3 Segmento de diagrama de flujo <func-op-call>

El segmento de diagrama de flujo `<func-op-call>` de la figura 73 se refiere a solicitudes de funciones o de operaciones que devuelven un valor que se pone en la pila de valores de la entidad. Todas estas solicitudes se consideran expresiones.

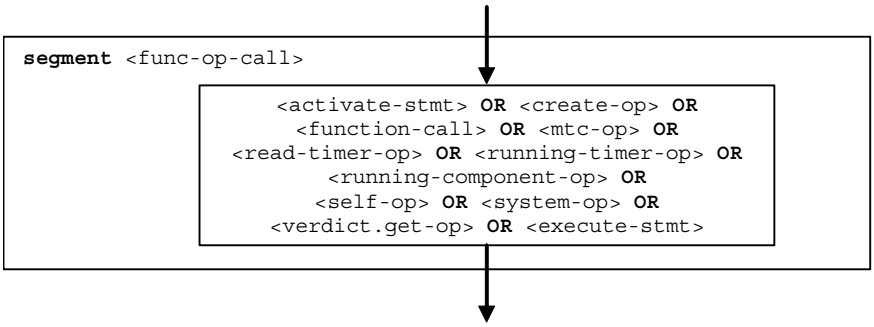


Figura 73/Z.143 – Segmento de diagrama de flujo <func-op-call>

9.18.4 Segmento de diagrama de flujo <operator-appl>

La representación de diagrama de flujo de la figura 74 corresponde directamente a la suposición de que se utiliza notación polaca inversa para evaluar las expresiones con operadores. Se calculan los operandos del operador y se introducen en la pila de evaluación. Para aplicar el operador, primero se extraen los operandos de la pila de evaluación y luego se aplica el operador. Finalmente se introduce en la pila de evaluación el resultado de aplicar el operador. Tanto la extracción de operandos como la introducción del resultado forman parte de la instrucción de aplicación del operador, `Entity.APPLY-OPERATOR(operator)` de la figura 74, es decir, la semántica operacional no los modela.

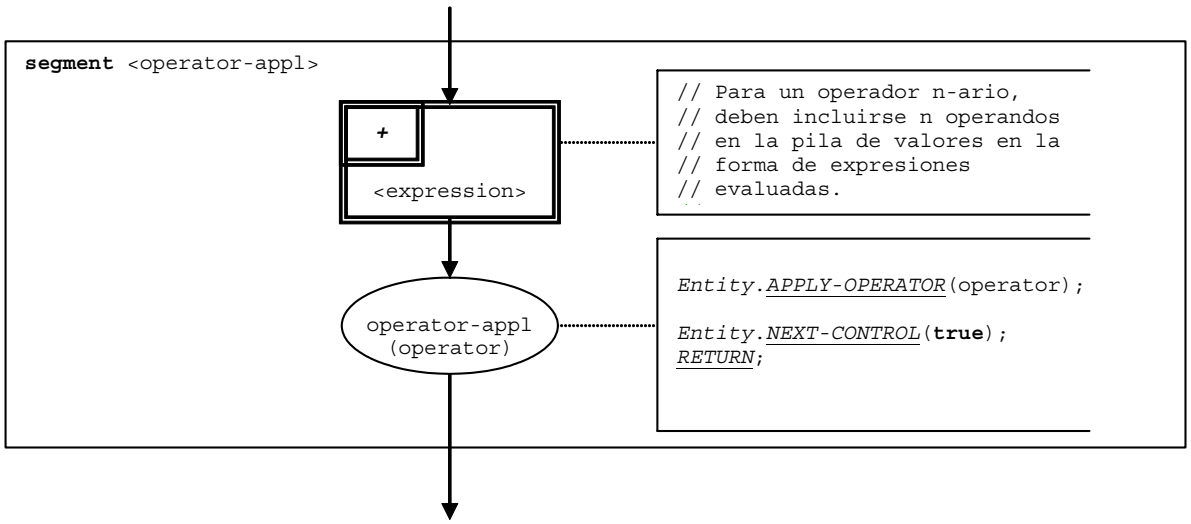


Figura 74/Z.143 – Segmento de diagrama de flujo <operator-appl>

9.18b Segmento de diagrama de flujo <dynamic-error>

Si ocurre un error dinámico, el sistema de pruebas invoca el segmento de diagrama de flujo `<dynamic-error>` (véase la figura 74-b). Se liberan todos los recursos adjudicados al caso de prueba y se asigna el veredicto de **error** al caso de prueba. Se otorga el control a la instrucción en la parte de control posterior a la instrucción de ejecución en que ocurrió el error.

Si el caso de prueba no finaliza en el periodo de tiempo especificado (véase 9.17.2), el control del módulo invoca el segmento de diagrama de flujo `<dynamic-error>`.<

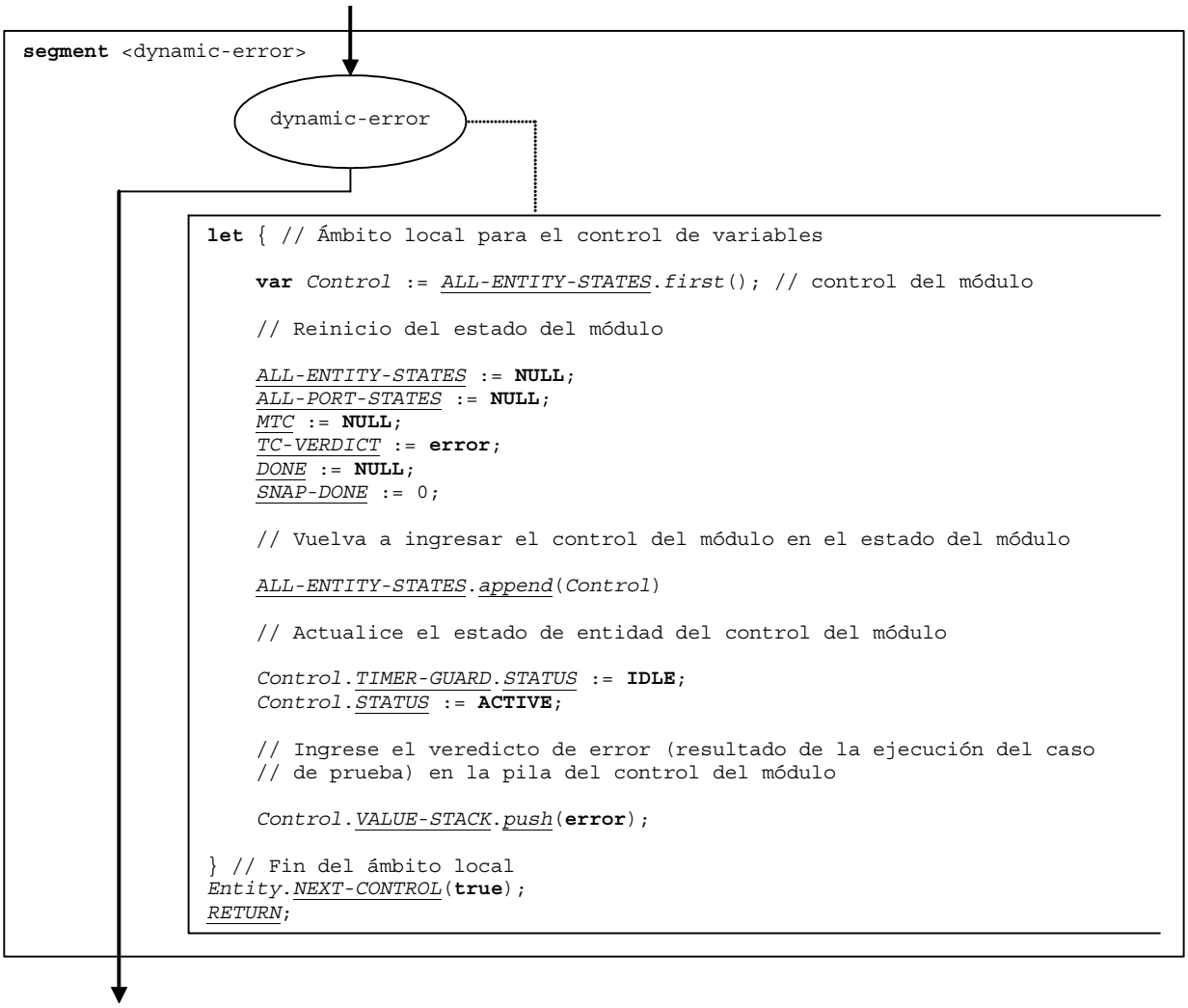


Figura 74-b/Z.143 – Segmento de diagrama de flujo <dynamic-error>

9.19 Segmento de diagrama de flujo <finalize-component-init>

El segmento de diagrama de flujo `<finalize-component-init>` forma parte del diagrama de flujo que representa el comportamiento de una definición de tipo de componente. En la figura 75 se define su ejecución.

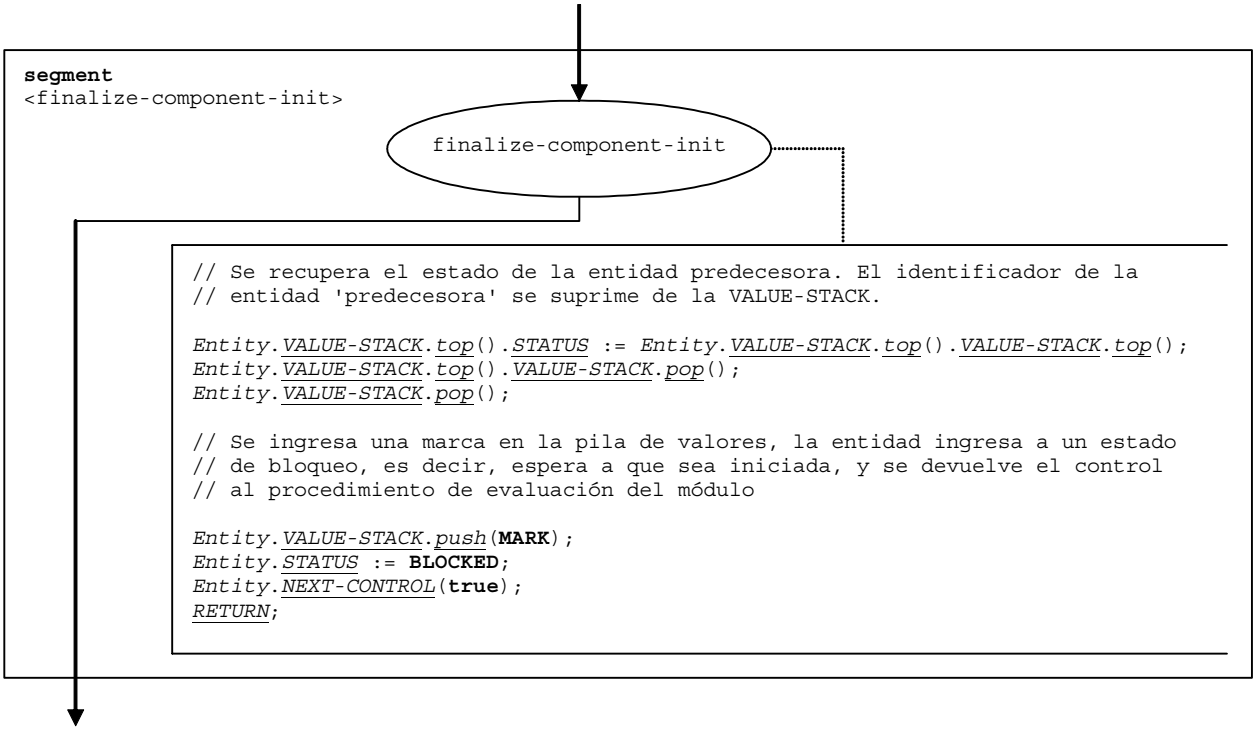


Figura 75/Z.143 – Segmento de diagrama de flujo <finalize-component-init>

9.20 Segmento de diagrama de flujo <init-component-scope>

El segmento de diagrama de flujo `<init-component-scope>` forma parte del diagrama de flujo que representa el comportamiento de una definición de tipo de componente. En la figura 76 se define su ejecución.

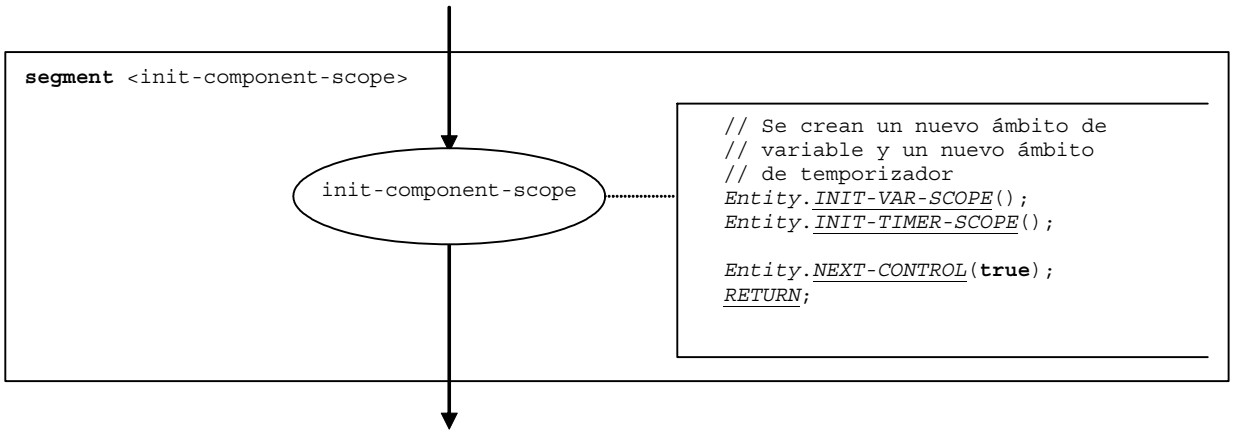


Figura 76/Z.143 – Segmento de diagrama de flujo <init-component-scope>

9.21 Segmento de diagrama de flujo <parameter-handling>

Se utiliza el segmento de diagrama de flujo `<parameter-handling>` al inicio de diagramas de flujo que representan casos de prueba, altstep y funciones. El segmento inicializa un nuevo ámbito y crea variables y temporizadores para manejar los parámetros. El segmento de diagrama de flujo `<parameter-handling>` supone que el registro de solicitud del caso de prueba, altstep o función se encuentra en la cima de la pila de valores.

En la figura 77 se muestra la ejecución del segmento de diagrama de flujo <parameter-handling>.

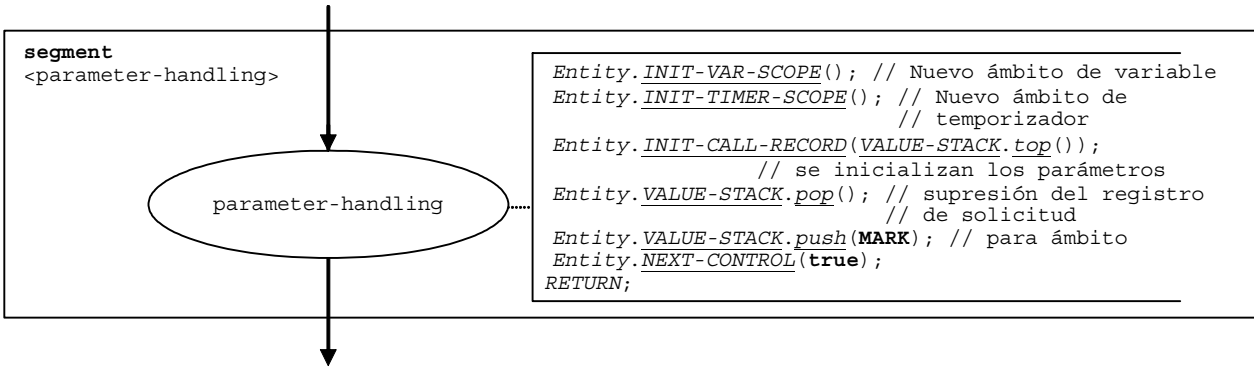


Figura 77/Z.143 – Segmento de diagrama de flujo <parameter-handling>

9.22 Segmento de diagrama de flujo <statement-block> (Bloque de instrucciones)

La siguiente es la estructura sintáctica de los bloques de instrucciones:

{ <statement$_1$>; ... ; <statement$_n$> **}**

Los bloques de instrucciones son unidades de ámbito. Al ingresar a una unidad de ámbito, deben inicializarse nuevos ámbitos para variables, temporizadores y pila de valores. Al salir de una unidad de ámbito, deben destruirse todas las variables, temporizadores y valores de pila de dicho ámbito.

NOTA 1 – El bloque de instrucciones no es un concepto 'oficial' de TTCN-3. Los bloques de instrucciones ocurren únicamente en el cuerpo de las funciones, altstep, casos de prueba y control del módulo, y dentro de instrucciones compuestas, como por ejemplo, **alt**, **if-else** y **do-while**.

NOTA 2 – Las operaciones de recepción y las solicitudes de altstep no pueden aparecer en bloques de instrucciones, se incluyen en instrucciones **alt** o en operaciones **call**.

NOTA 3 – La semántica operacional también maneja las operaciones y las declaraciones de la misma forma que a las instrucciones, es decir, las admite en los bloques de instrucciones.

NOTA 4 – Se considera que algunas funciones útiles de TTCN-3, como **system** o **self**, son expresiones que no son de utilidad como instrucciones independientes en bloques de instrucciones. En la figura 78 no figuran sus representaciones de diagrama de flujo.

El segmento de diagrama de flujo <statement-block> de la figura 78 define la ejecución de los bloques de instrucciones.

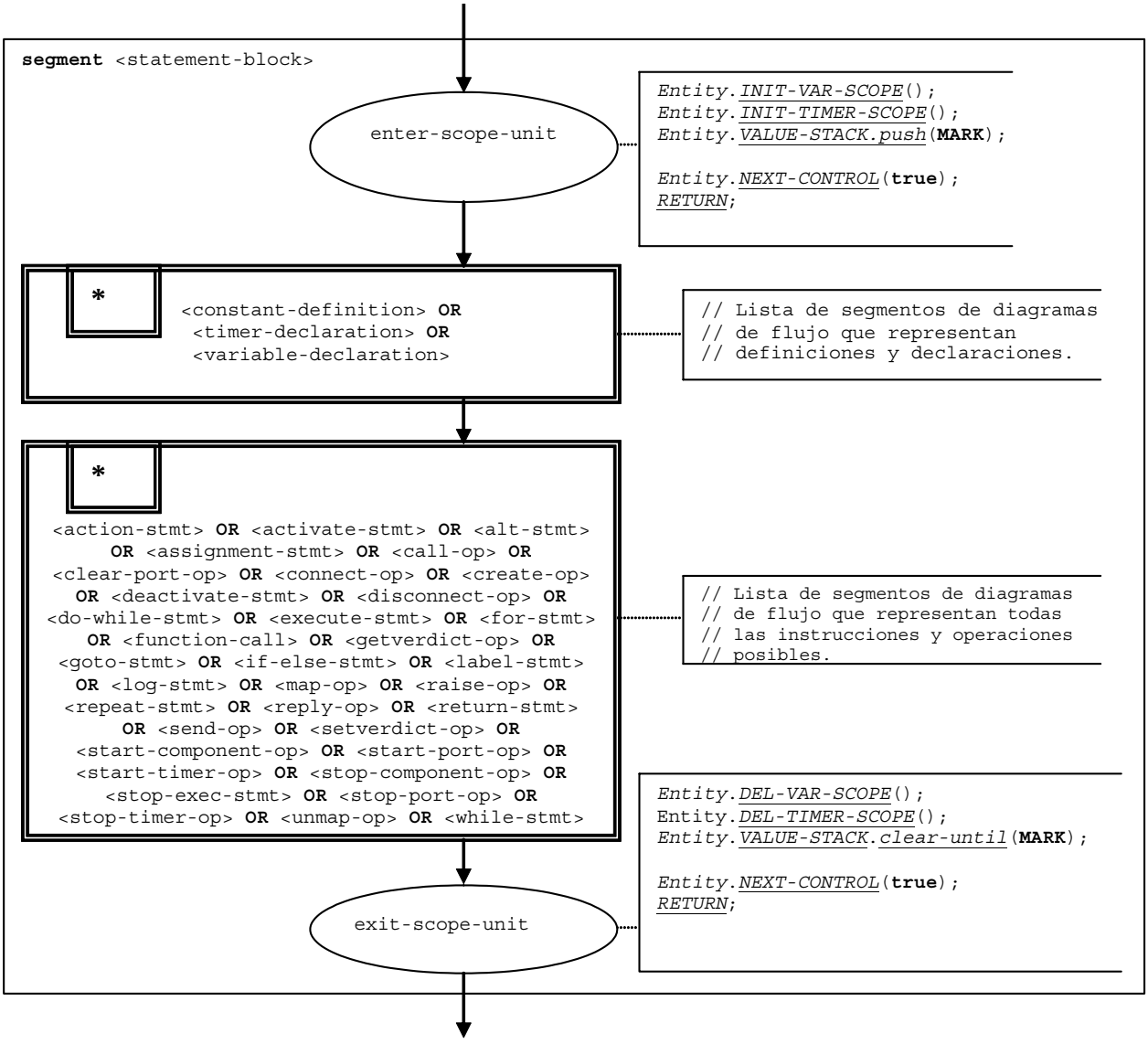


Figura 78/Z.143 – Segmento de diagrama de flujo <statement-block>

9.23 Instrucción for

La siguiente es la estructura sintáctica de la **instrucción for**:

```
for (<assignment>|<variable-declaration>, <boolean expression>, <assignment>) <statement-block>
```

Se considera que la inicialización de la variable de índice y el correspondiente manejo de la variable de índice son asignaciones de la variable de índice. También se permite que la variable de índice se declare e inicialice directamente en la **instrucción for**. <boolean-expression> describe el criterio de finalización del bucle especificado por la **instrucción for** mientras que <statement-block> describe el cuerpo del bucle.

El segmento de diagrama de flujo <for-stmt> que se muestra en la figura 79 define la ejecución de la **instrucción for**. La asignación <assignment> inicial o la declaración alternativa de variables con la asignación <var-declaration-init> (véase 9.57.1) describe la inicialización de la variable de índice. La **instrucción for** es una unidad de ámbito para una variable de índice recientemente declarada, lo que se modela mediante los nodos enter-var-scope y exit-var-scope.

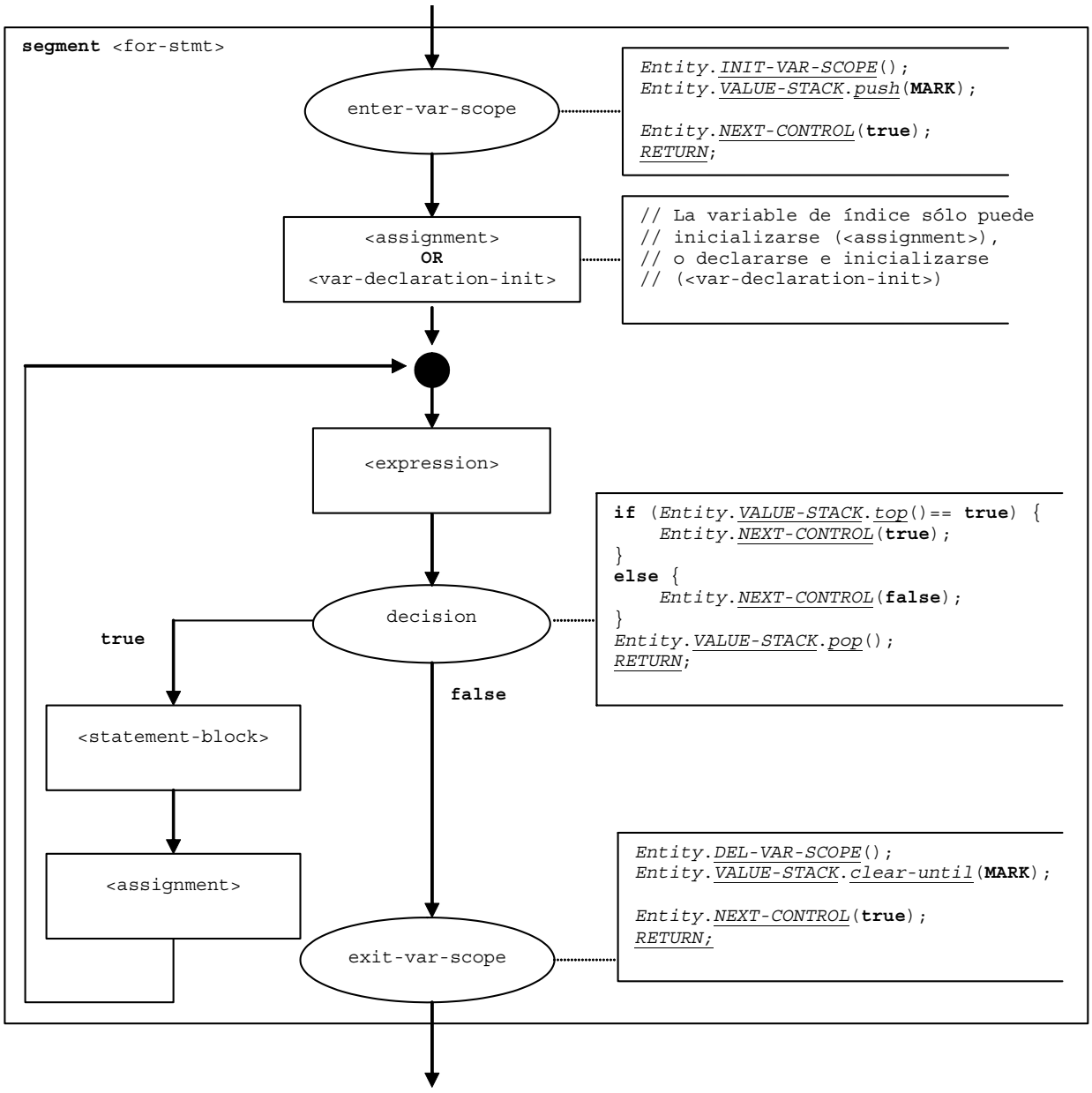


Figura 79/Z.143 – Segmento de diagrama de flujo <for-stmt>

9.24 Solicitudes de funciones

La siguiente es la estructura sintáctica de las solicitudes de funciones:

<function-name>([<act-par-desc$_1$>, ... , <act-par-desc$_n$>])

<function-name> indica el nombre de la función y <act-par-descr$_1$>, … , <act-par-descr$_n$> es la descripción de los valores de parámetro efectivos de la solicitud de función.

NOTA 1 – Las solicitudes de función y las solicitudes de altstep se tratan de una misma forma. Por consiguiente, la solicitud de altstep (véase 9.4) hace referencia a la presente cláusula.

Se supone que para cada <act-par-desc$_1$> se conoce el correspondiente identificador de parámetro <f-par-Id$_1$>, es decir, la estructura sintáctica anterior puede extenderse así:

<function-name>((<f-par-Id$_1$>,<act-par-desc$_1$>), ... , (<f-par-Id$_n$>,<act-par-desc$_n$>))

El segmento de diagrama de flujo <function-call> de la figura 80 define la ejecución de las solicitudes de función. La ejecución consiste en tres pasos. En el primer paso se crea un registro de solicitud para la función <function-name>. En el segundo paso se calculan los valores de los parámetros efectivos y se asignan al campo correspondiente del registro de solicitud. Para el tercer paso se deben distinguir dos casos: la función solicitada es una función definida por el usuario (<user-def-func-call>), es decir, existe una representación de diagrama de flujo

para la función, o la función solicitada es una función predefinida o externa (<predef-ext-func-call>). Si se trata de una solicitud de una función definida por el usuario, se otorga el control a la función solicitada. Si se trata de una función predefinida o externa, se supone que se puede utilizar el registro de solicitud para ejecutar la función en un solo paso. El manejo de los parámetros por referencia y del valor devuelto (debe introducirse en la pila de valores) es responsabilidad de la función solicitada, es decir, no se trata en esta semántica operacional.

NOTA 2 – Si la solicitud de función modela un altstep, sólo se escogerá la rama <user-def-func-call>, ya que existe una representación de diagrama de flujo del altstep.

NOTA 3 – El segmento <function call> también se utiliza para describir el inicio del MTC en las instrucciones **execute**. En dicho caso, se construye un registro de solicitud para el caso de prueba y se escogerá únicamente la rama <user-def-func-call>.

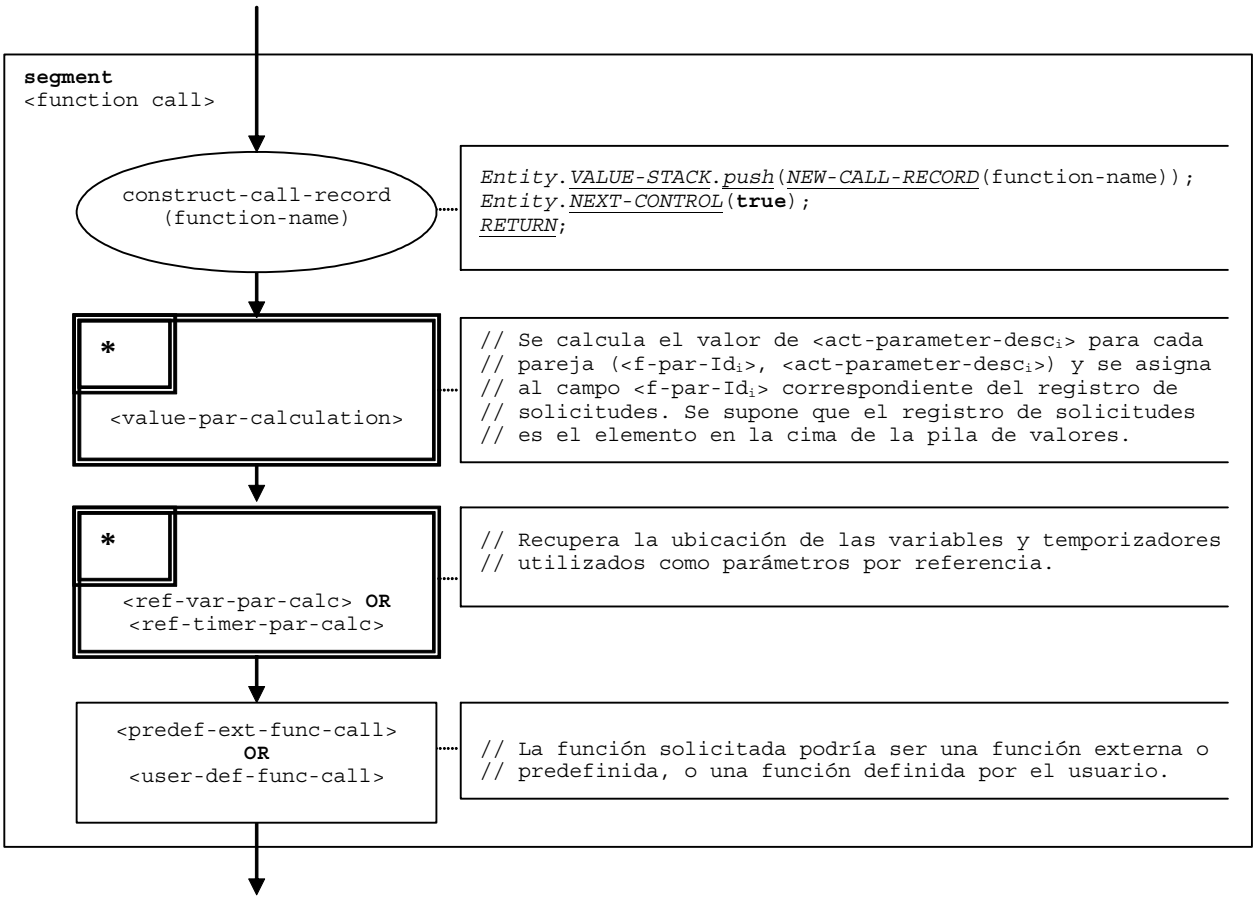


Figura 80/Z.143 – Segmento de diagrama de flujo <function-call>

9.24.1 Segmento de diagrama de flujo <value-par-calculation>

El segmento de diagrama de flujo <value-par-calculation> se utiliza para calcular los valores de los parámetros efectivos y asignarlos a los campos correspondientes de los registros de solicitud para funciones, altstep y casos de prueba.

Se supone que el registro de solicitudes es el elemento en la cima de la pila de valores y que se debe manejar la pareja:

(<f-par-Id$_i$>, <act-parameter-desc$_i$>)

El parámetro <act-parameter-desc$_i$>, que debe evaluarse, y el parámetro <f-par-Id$_i$> son el identificador de un parámetro formal que posee un campo correspondiente en el registro de solicitudes en la pila de valores.

En la figura 81se muestra la ejecución del segmento de diagrama de flujo <value-par-calculation>.

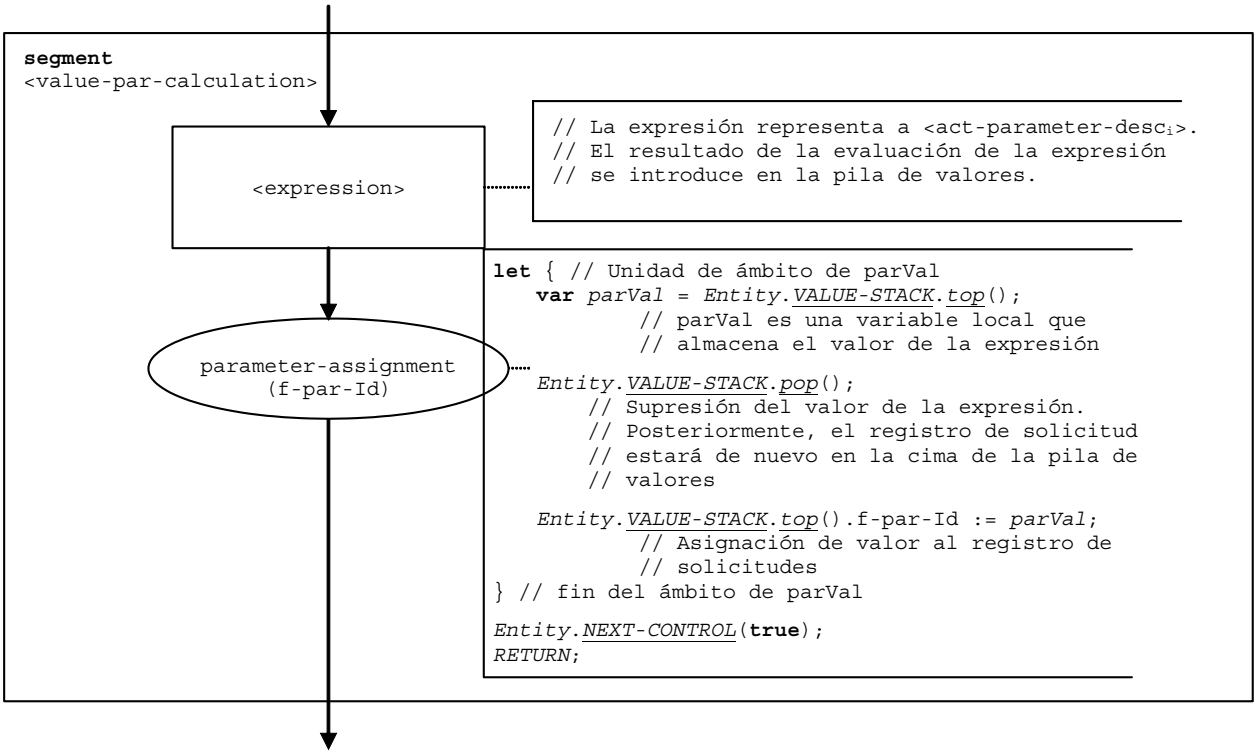


Figura 81/Z.143 – Segmento de diagrama de flujo <value-par-calculation>

9.24.2 Segmento de diagrama de flujo <ref-par-var-calc>

Se utiliza el segmento de diagrama de flujo <ref-par-var-calc> para recuperar las ubicaciones de las variables utilizadas como parámetros por referencia efectivos y asignarlas a los campos correspondientes en los registros de solicitud de funciones, altstep y casos de prueba.

Se supone que el registro de solicitudes es el elemento en la cima de la pila de valores y que se debe manejar la pareja:

(<f-par-Id$_i$>, <act-par$_i$>)

<act-par$_i$> es el parámetro efectivo cuya ubicación se debe recuperar y <f-par-Id$_i$> es el identificador de un parámetro formal que posee un campo correspondiente en el registro de solicitudes en la pila de valores.

En la figura 82 se muestra la ejecución del segmento de diagrama de flujo <ref-par-var-calc>.

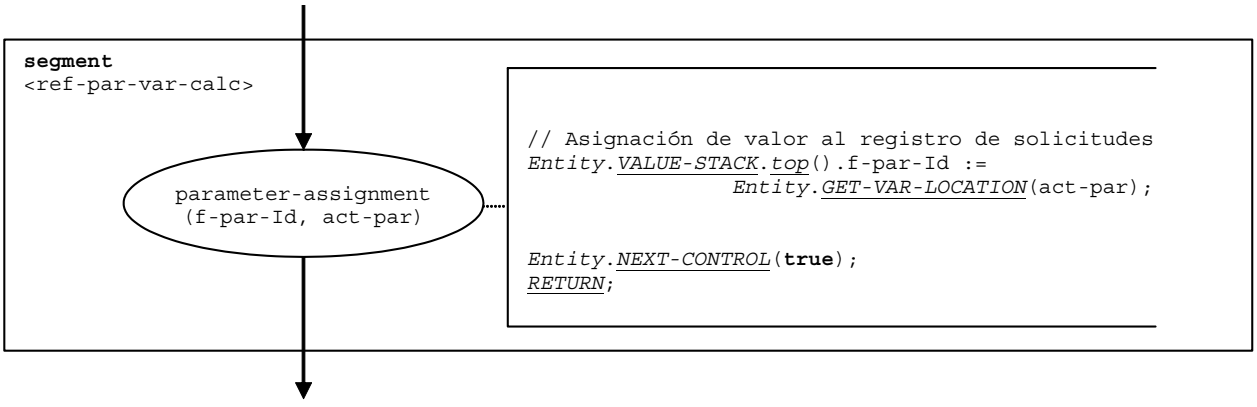


Figura 82/Z.143 – Segmento de diagrama de flujo <ref-par-var-calc>

9.24.3 Segmento de diagrama de flujo <ref-par-timer-calc>

Se utiliza el segmento de diagrama de flujo <ref-par-timer-calc> para recuperar la ubicación de los temporizadores utilizados como parámetros por referencia efectivos y asignarlas a los campos correspondientes de los registros de solicitud para funciones, altstep y casos de prueba.

Se supone que el registro de solicitudes es el elemento superior de la pila de valores y que se debe manejar la pareja:

(<f-par-Id$_i$>, <act-par$_i$>)

<act-par$_i$> es el parámetro efectivo cuya ubicación se debe recuperar y <f-par-Id$_i$> es el identificador de un parámetro formal que posee un campo correspondiente en el registro de solicitudes en la pila de valores.

En la figura 83 se muestra la ejecución del segmento de diagrama de flujo <ref-par-timer-calc>.

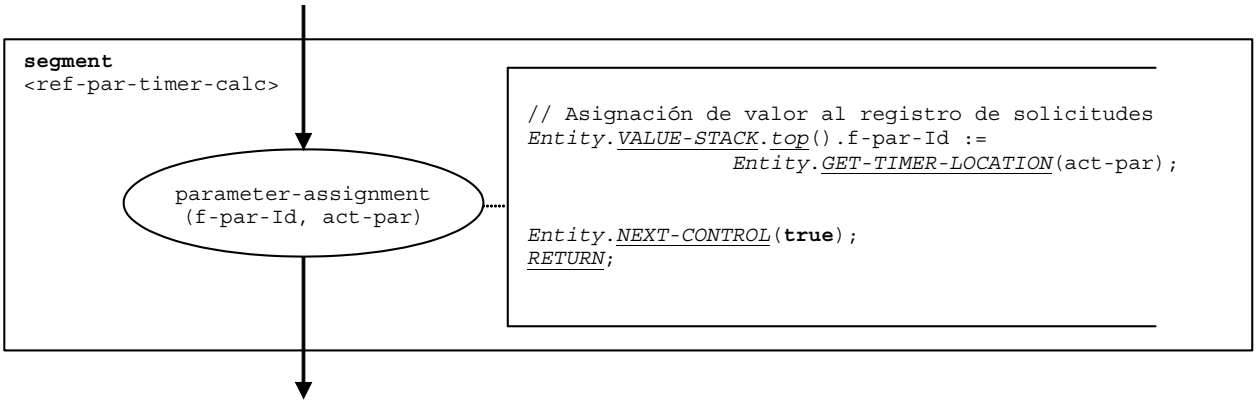


Figura 83/Z.143 – Segmento de diagrama de flujo <ref-par-timer-calc>

9.24.4 Segmento de diagrama de flujo <user-def-func-call>

El segmento de diagrama de flujo <user-def-func-call> (figura 84) describe la transferencia del control a una función solicitada definida por el usuario.

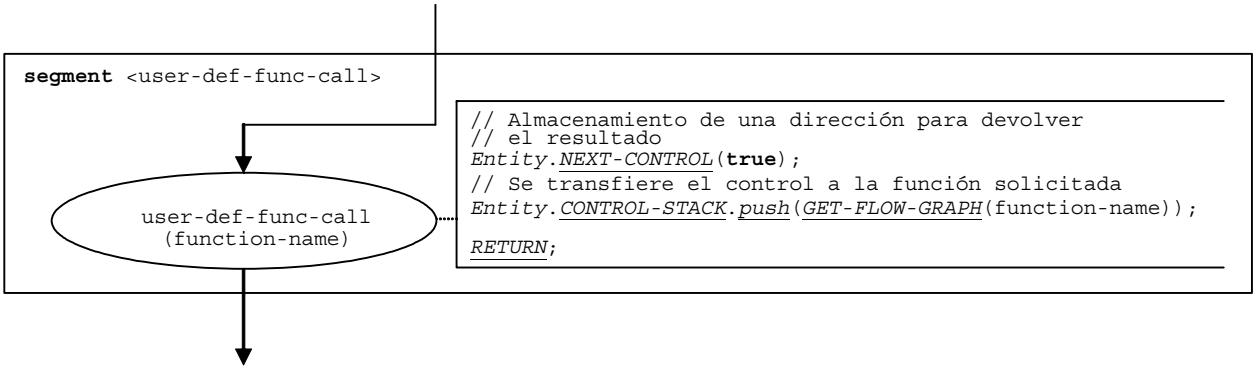


Figura 84/Z.143 – Segmento de diagrama de flujo <user-def-func-call>

9.24.5 Segmento de diagrama de flujo <predef-ext-func-call>

El segmento de diagrama de flujo <predef-ext-func-call> (figura 85) describe la solicitud de una función predefinida o externa.

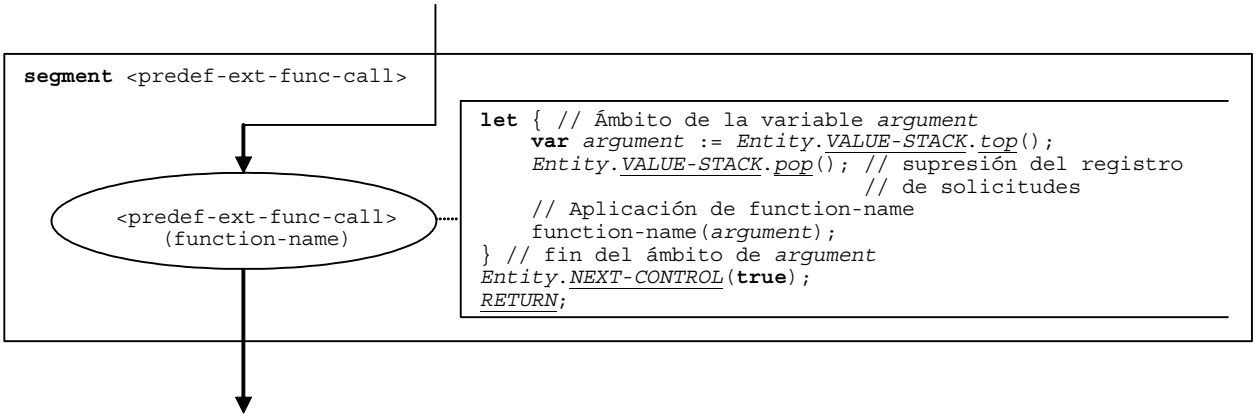


Figura 85/Z.143 – Segmento de diagrama de flujo <predef-ext-func-call>

9.25 Operación getcall

La siguiente es la estructura sintáctica de las operaciones **getcall**:

```
<portId>.getcall (<matchingSpec>) [from <component expression>] -> [<assignmentPart>]
```

Salvo por la palabra clave **getcall**, esta estructura sintáctica es idéntica a la estructura sintáctica de las operaciones **receive**. Por consiguiente, la semántica operacional trata la operación **getcall** de la misma forma que la operación **receive**. Esto también se aprecia en el segmento de diagrama de flujo <getcall-op> (véase la figura 86), que define la ejecución de las operaciones **getcall**. La figura hace referencia a los segmentos de diagrama de flujo relativos a la operación **receive** (véase 9.37).

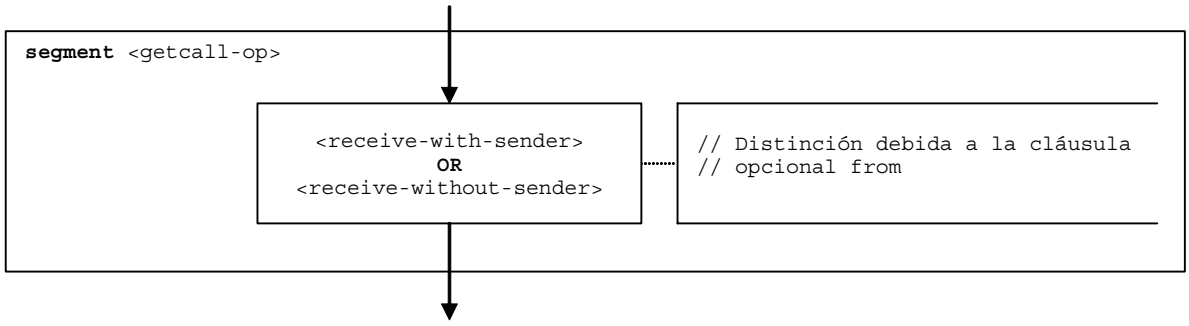


Figura 86/Z.143 – Segmento de diagrama de flujo <getcall-op>

9.26 Operación getreply

La siguiente es la estructura sintáctica de las operaciones **getreply**:

```
<portId>.getreply (<matchingSpec>) [from <component-expression>] [-> <assignmentPart>]
```

Salvo por la palabra clave **getreply**, esta estructura sintáctica es idéntica a la estructura sintáctica de las operaciones **receive**. Por consiguiente, la semántica operacional trata la operación **getreply** de la misma forma que la operación **receive**. Esto también se aprecia en el segmento de diagrama de flujo <getreply-op> (véase la figura 87), que define la ejecución de las operaciones **getreply**. La figura hace referencia a los segmentos de diagrama de flujo relativos a la operación **receive** (véase 9.37).

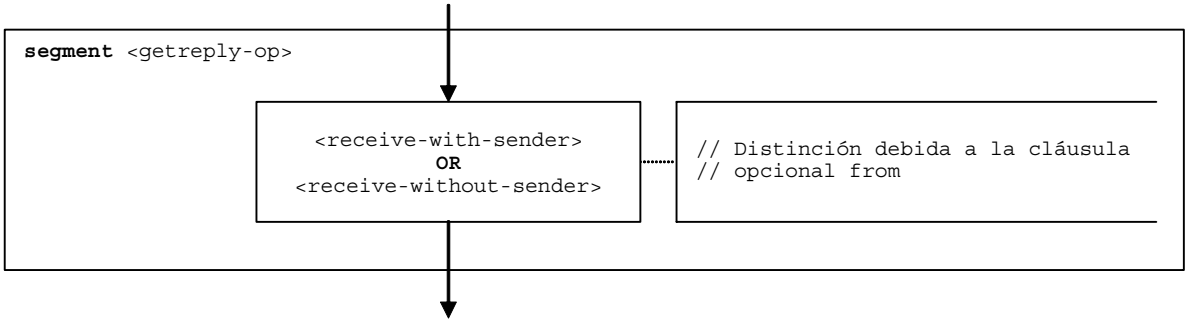


Figura 87/Z.143 – Segmento de diagrama de flujo <getreply-op>

9.27 Operación getverdict

La siguiente es la estructura sintáctica de las operaciones **getverdict**:

```
getverdict
```

El segmento de diagrama de flujo <getverdict-op> de la figura 88 define la ejecución de las operaciones **getverdict**.

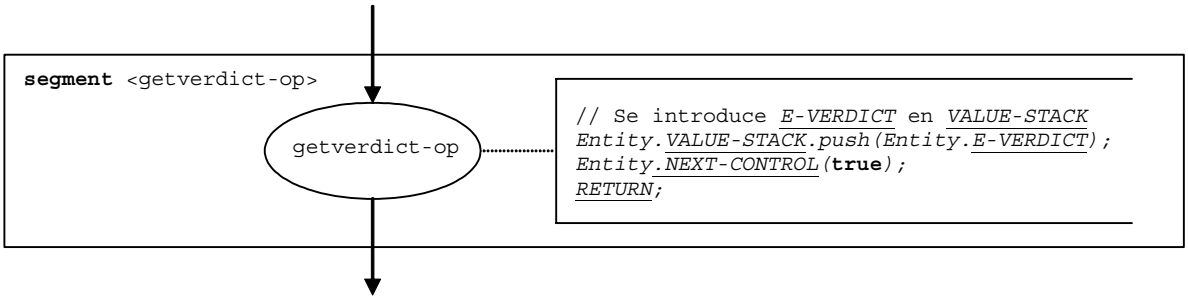


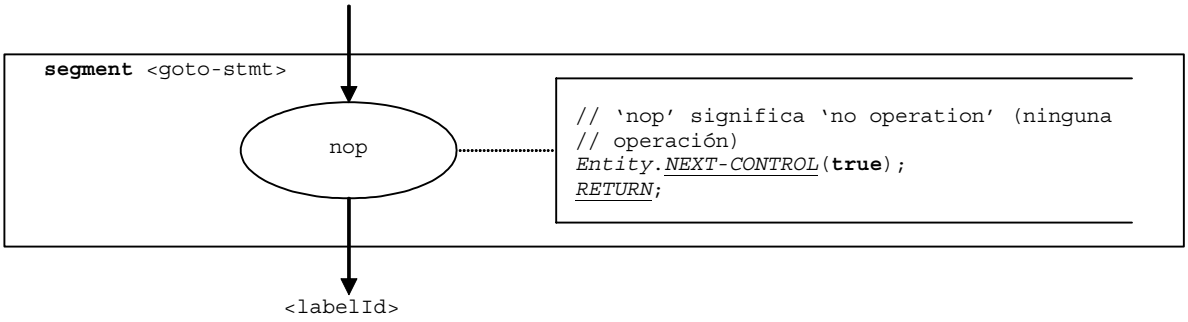
Figura 88/Z.143 – Segmento de diagrama de flujo <getverdict-op>

9.28 Instrucción goto

La siguiente es la estructura sintáctica de las instrucciones **goto**:

```
goto <labelId>
```

El segmento de diagrama de flujo <goto-stmt> de la figura 89 define la ejecución de las instrucciones **goto**.



NOTA – El parámetro <labelId> de la instrucción **goto** indica que se transfiere el control al sitio en que se define la etiqueta <*labelId*> (véase también 9.30).

Figura 89/Z.143 – Segmento de diagrama de flujo <goto-stmt>

9.29 Instrucción if-else

La siguiente es la estructura sintáctica de la instrucción **`if-else`**:

```
if (<boolean-expression>) <statement-block1>
    [else <statement-block2>]
```

La parte else de la instrucción **`if-else`** es opcional.

El segmento de diagrama de flujo `<if-else-stmt>` de la figura 90 define la ejecución de las instrucciones **`if-else`**.

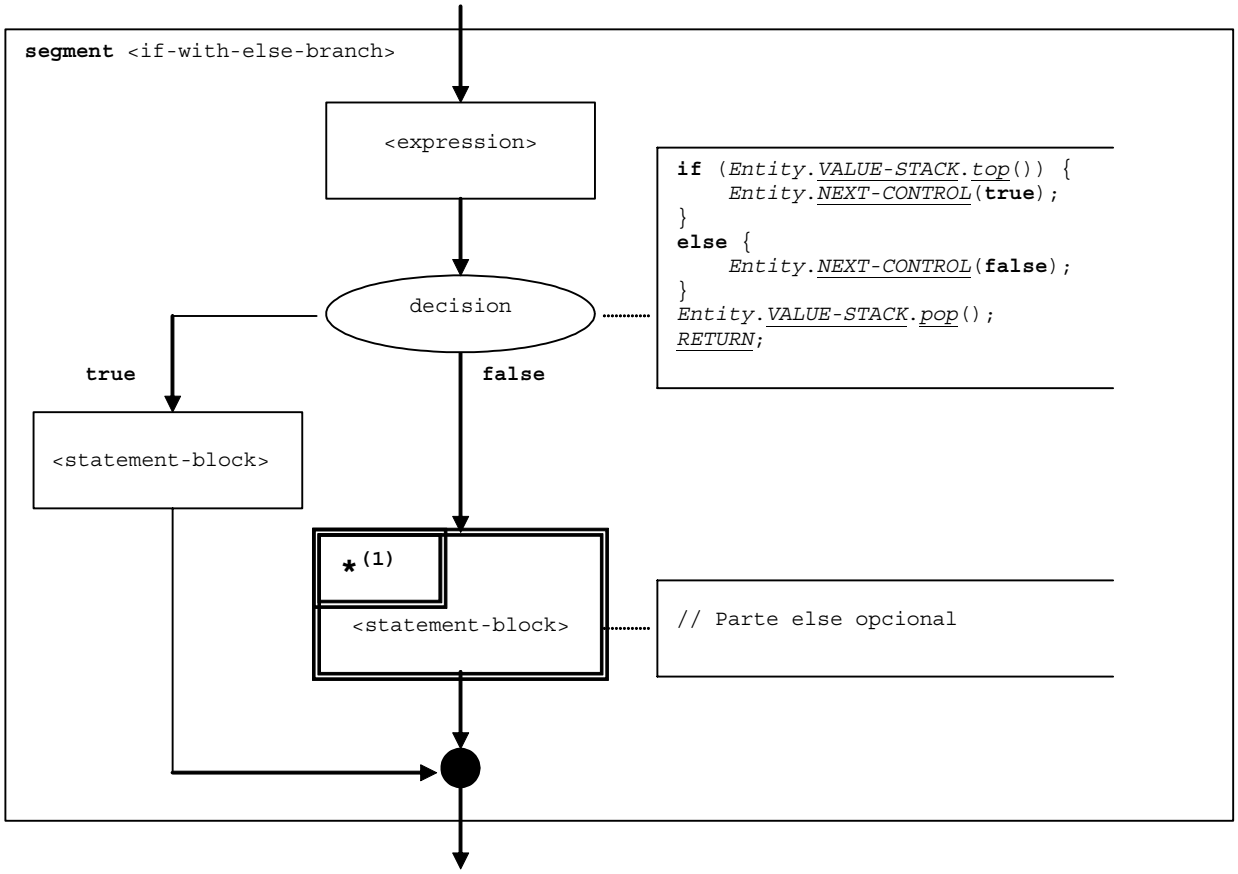


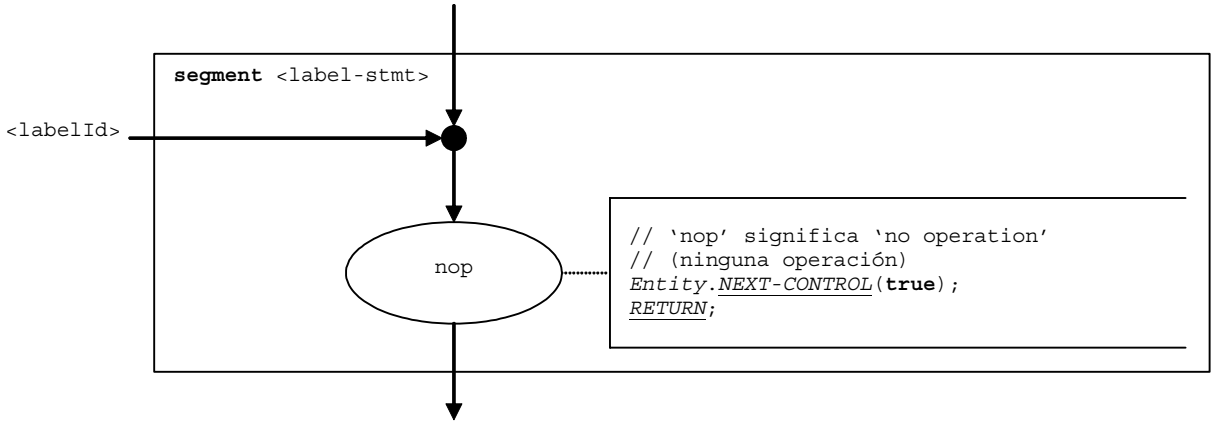
Figura 90/Z.143 – Segmento de diagrama de flujo <if-else-stmt>

9.30 Instrucción label (etiqueta)

La siguiente es la estructura sintáctica de la instrucción **`label`**:

```
label <labelId>
```

El segmento de diagrama de flujo <label-stmt> de la figura 91 define la ejecución de las instrucciones **label**.



NOTA – El parámetro <labelId> de la instrucción label indica la posibilidad de usar una etiqueta como destino de un salto efectuado mediante una instrucción **goto** (véase también 9.28).

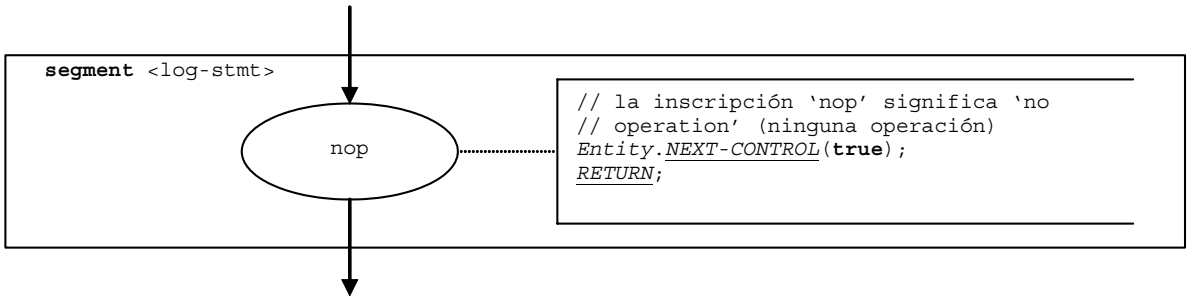
Figura 91/Z.143 – Segmento de diagrama de flujo <label-stmt>

9.31 Instrucción log

La siguiente es la estructura sintáctica de las instrucciones **log**:

```
log (<informal-description>)
```

El segmento de diagrama de flujo <log-stmt> de la figura 92 define la ejecución de las instrucciones **log**.



NOTA – El parámetro <informal description> de la instrucción **log** no es relevante para la semántica operacional y por lo tanto no se representa en el segmento de diagrama de flujo.

Figura 92/Z.143 – Segmento de diagrama de flujo <log-stmt>

9.32 Operación map

La siguiente es la estructura sintáctica de las operaciones **map**:

```
map(<component-expression>:<portId1>, system:<portId2>)
```

Se considera que <portId1> y <portId2> son los identificadores correspondientes del puerto del componente de prueba y de la interfaz del sistema de pruebas. Los componentes a que pertenece <portId1> se referencian por medio de la referencia de componente <component-expression>. La referencia puede almacenarse en variables o puede devolverse en una función, es decir, se trata de una expresión que da como resultado una referencia de componente. Se utiliza la pila de valores para almacenar la referencia de componente.

NOTA – Para la operación **map** es indiferente si la instrucción **system**:<portId> aparece como primer o segundo parámetro. En aras de simplicidad, se supone que siempre es el segundo parámetro.

El segmento de diagrama de flujo <map-op> de la figura 93 define la ejecución de las operaciones **map**.

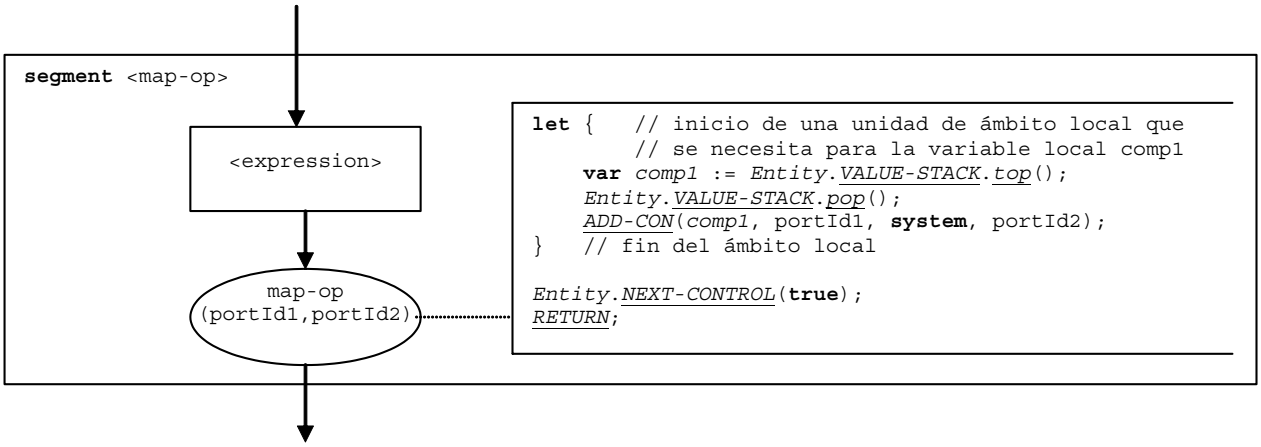


Figura 93/Z.143 – Segmento de diagrama de flujo <map-op>

9.33 Operación mtc

La siguiente es la estructura sintáctica de las operaciones **mtc**:

```
mtc
```

El segmento de diagrama de flujo <mtc-op> de la figura 94 define la ejecución de las operaciones **mtc**.

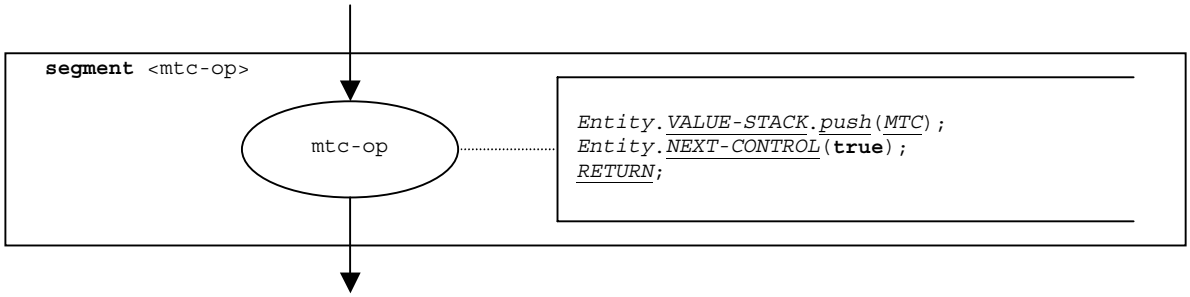


Figura 94/Z.143 – Segmento de diagrama de flujo <mtc-op>

9.34 Declaración de puertos

La siguiente es la estructura sintáctica de la declaración de **puertos**:

```
<portType> <portName>
```

Las definiciones de tipo de componente pueden contener declaraciones de puertos. Como resultado de la declaración de un puerto, se crea un puerto nuevo cuando se crea un nuevo componente del tipo correspondiente. El segmento de diagrama de flujo <port-declaration> de la figura 95 define la ejecución de las declaraciones de puertos.

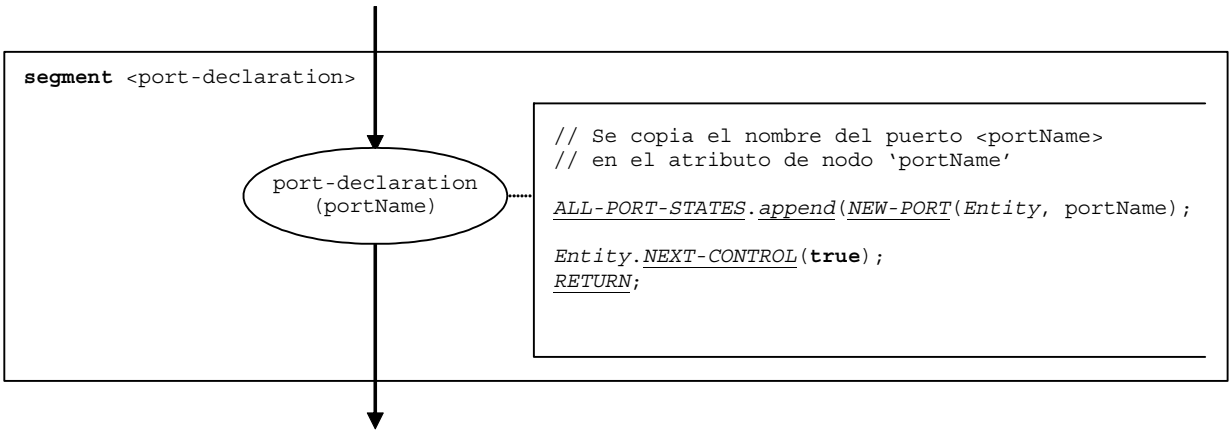


Figura 95/Z.143 – Segmento de diagrama de flujo <port-declaration>

9.35 Operación raise

La siguiente es la estructura sintáctica de las operaciones **raise**:

```
<portId>.raise (<exceptSpec>) [to <component-expression>]
```

El parámetro opcional <component-expression> de la cláusula to se refiere a la entidad receptora. Podría tomar la forma de un valor variable o del valor devuelto por una función.

El segmento de diagrama de flujo <raise-op> de la figura 96 define la ejecución de las operaciones **raise**.

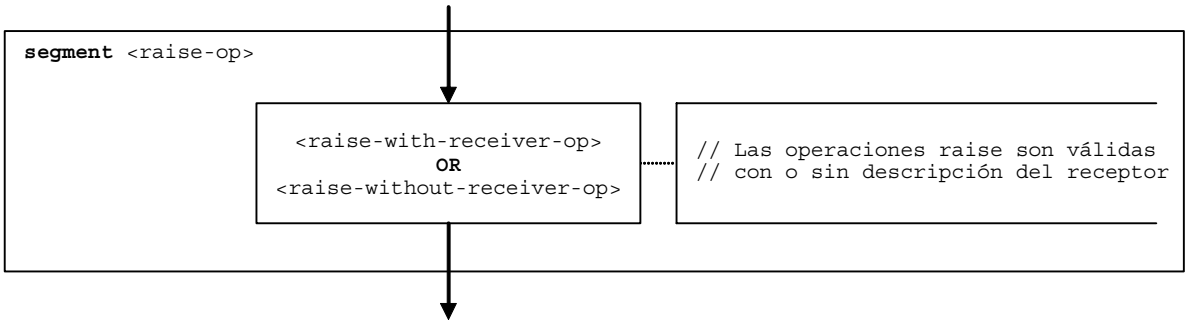


Figura 96/Z.143 – Segmento de diagrama de flujo <raise-op>

9.35.1 Segmento de diagrama de flujo <raise-with-receiver-op>

El segmento de diagrama de flujo <raise-with-receiver-op> de la figura 97 define la ejecución de las operaciones **raise**, especificando el receptor en la forma de una expresión.

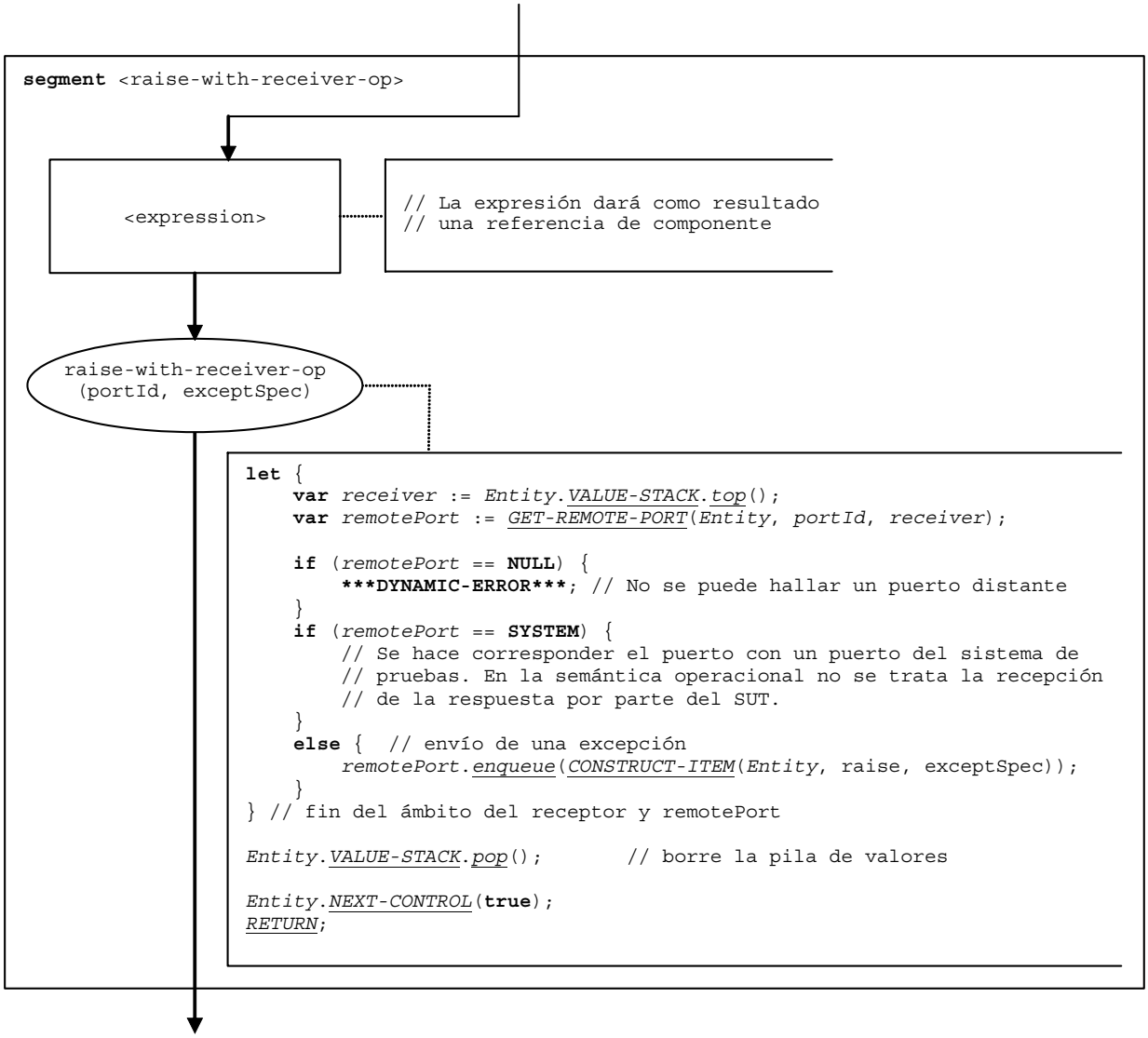


Figura 97/Z.143 – Segmento de diagrama de flujo <raise-with-receiver-op>

9.35.2 Segmento de diagrama de flujo <raise-without-receiver-op>

El segmento de diagrama de flujo <raise-without-receiver-op> de la figura 98 define la ejecución de las operaciones raise sin cláusula **to**.

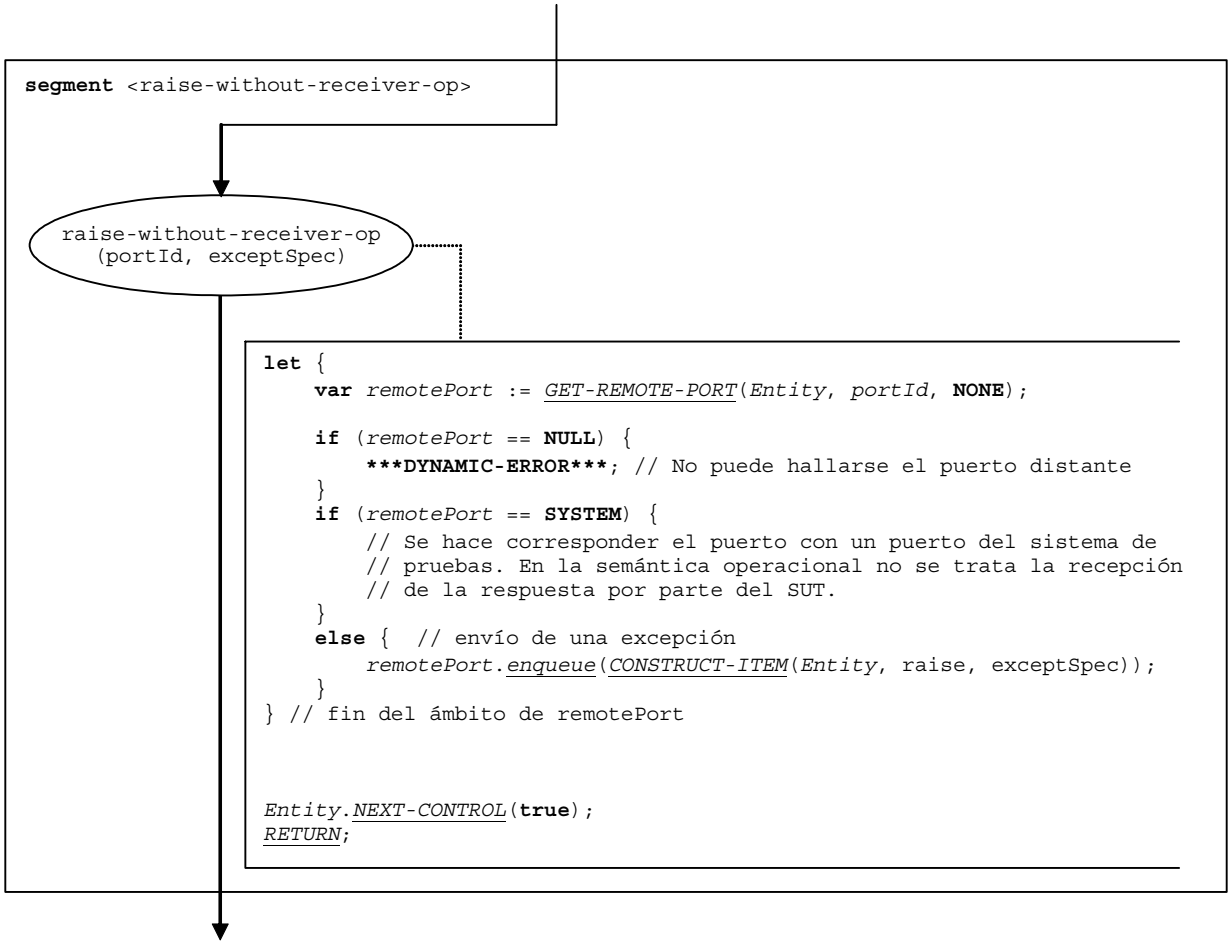


Figura 98/Z.143 – Segmento de diagrama de flujo <raise-without-receiver-op>

9.36 Operación de temporizador read

La siguiente es la estructura sintáctica de las operaciones de temporizador **read**:

```
<timerId>.read
```

El segmento de diagrama de flujo <read-timer-op> de la figura 99 define la ejecución de las operaciones de temporizador **read**.

El uso de la operación de temporizador **read** es diferente si se utiliza para el control de acceso booleano de las instrucciones **alt** o de las operaciones **call** bloqueantes que si se le da otro uso. Si se le utiliza para el control de acceso booleano, el resultado de la operación de temporizador **read** depende del estado puntual, es decir, de los valores de *SNAP-STATUS* y de *SNAP-VALUE* de las vinculación de temporizador. En los demás casos, el valor de *STATUS*, *ACT-DURATION* y *TIME-LEFT* de las vinculaciones de temporizador determinan el resultado de la operación.

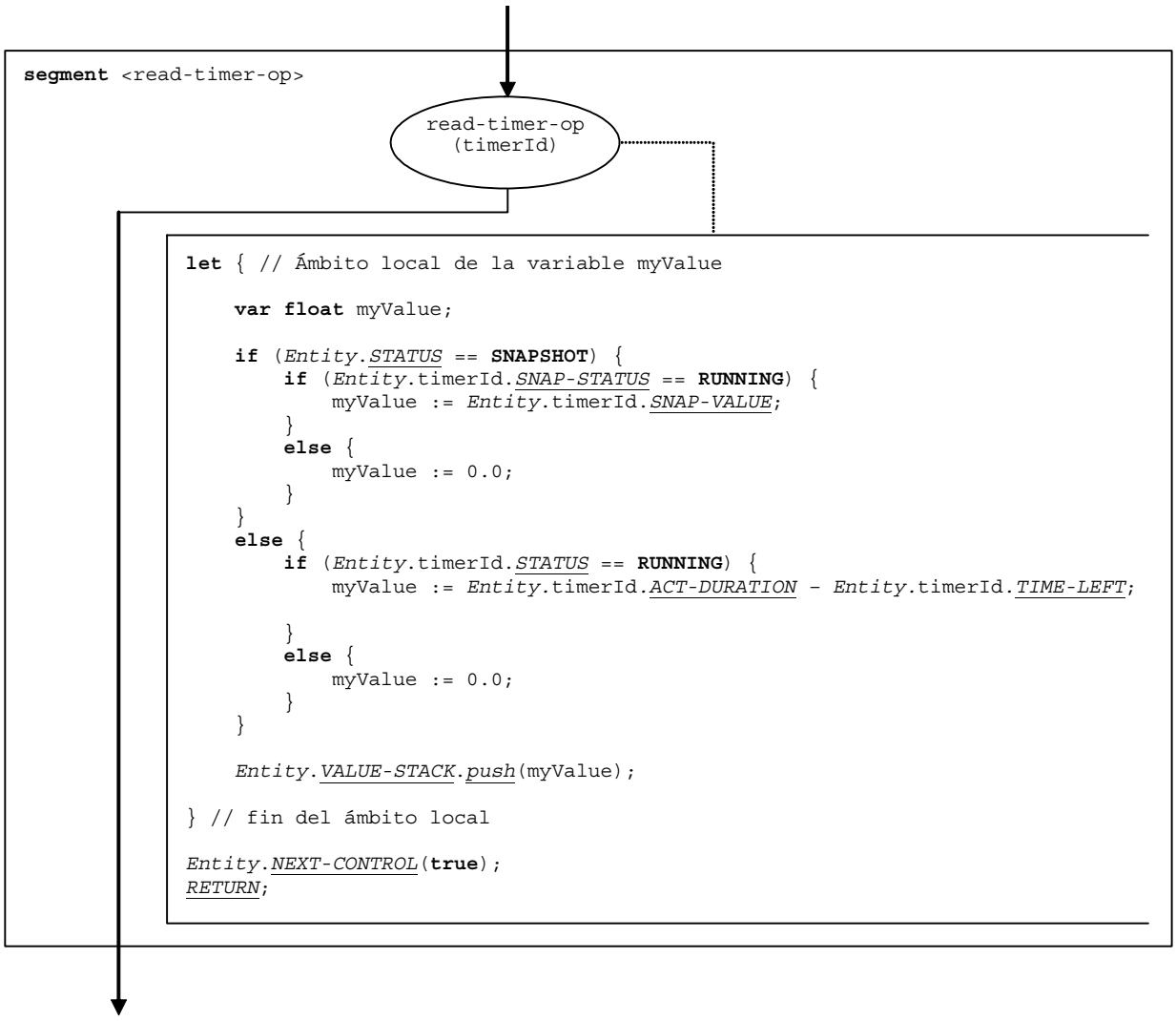


Figura 99/Z.143 – Segmento de diagrama de flujo <read-timer-op>

9.37 Operación receive

La siguiente es la estructura sintáctica de las operaciones **receive**:

```
<portId>.receive (<matchingSpec>) [from <component-expression>] [-> <assignmentPart>]
```

El parámetro opcional <component-expression> de la cláusula **from** se refiere a la entidad emisora. Puede ser un valor variable o el valor devuelto por una función es decir, se supone que es una expresión. <assignmentPart> indica la asignación de la información recibida siempre y cuando la información recibida corresponda con la especificación para correspondencia <matchingSpec> y con la cláusula opcional **from**.

El segmento de diagrama de flujo <receive-op> de la figura 100 define la ejecución de las operaciones **receive**.

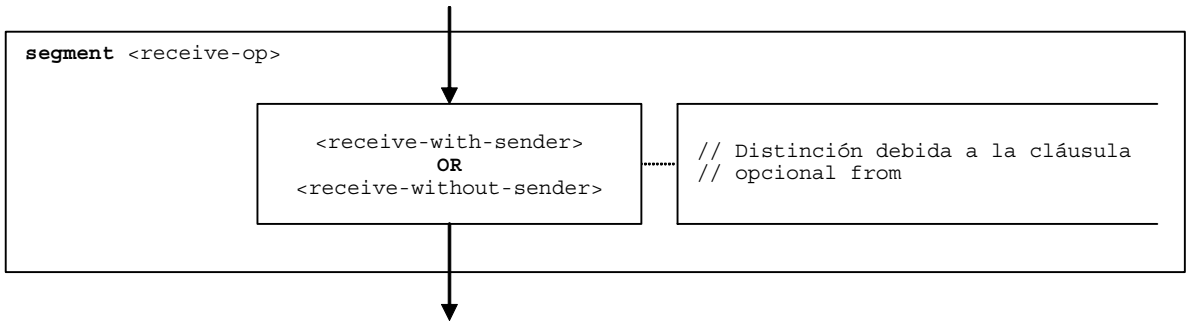


Figura 100/Z.143 – Segmento de diagrama de flujo <receive-op>

9.37.1 Segmento de diagrama de flujo <receive-with-sender>

El segmento de diagrama de flujo `<receive-with-sender>` de la figura 101 define la ejecución de las operaciones **`receive`** en las que se especifica el emisor en la forma de una expresión.

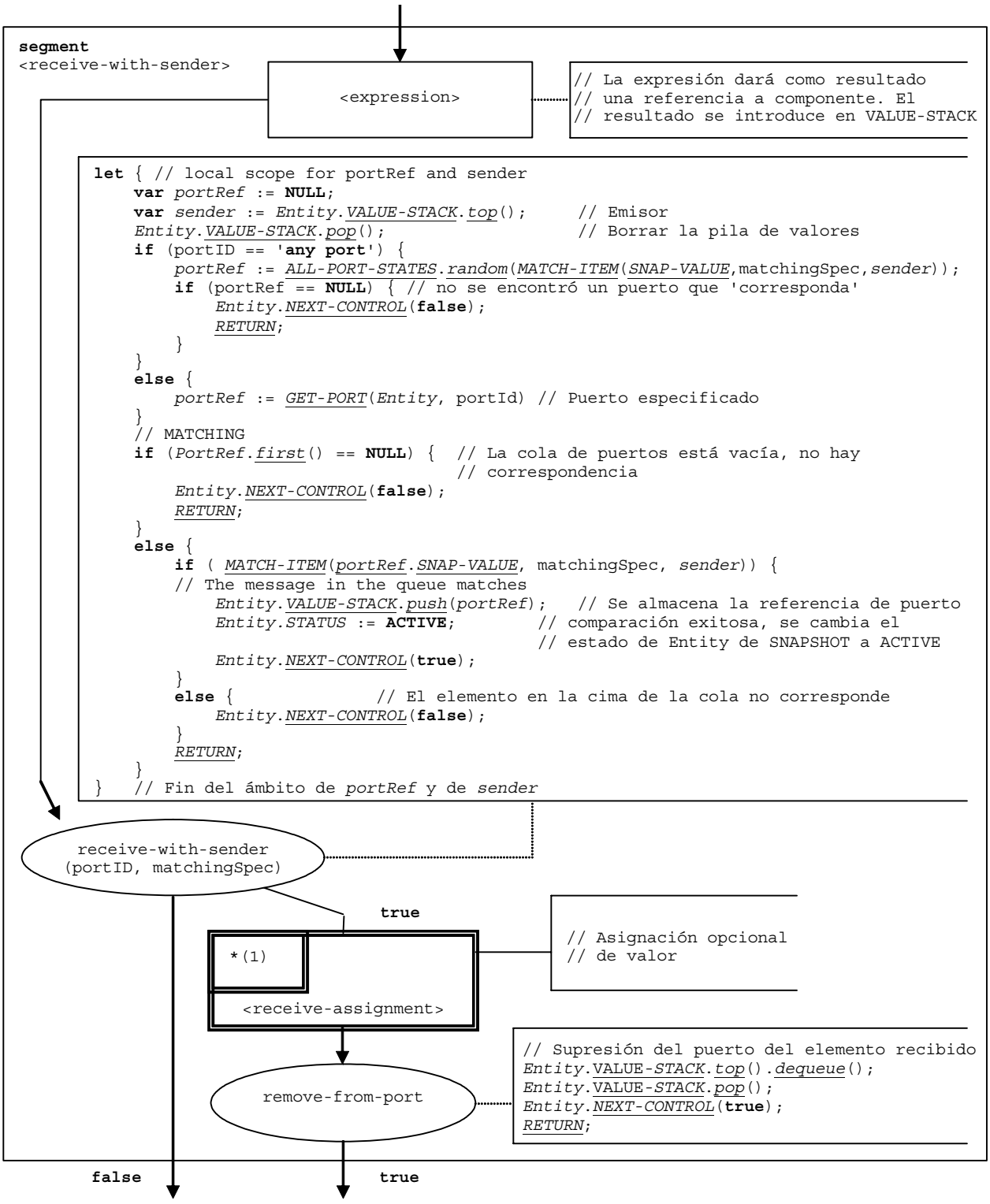


Figura 101/Z.143 – Segmento de diagrama de flujo <receive-with-sender>

9.37.2 Segmento de diagrama de flujo <receive-without-sender>

El segmento de diagrama de flujo <receive-without-sender> de la figura 102 define la ejecución de las operaciones **receive** sin cláusula **from**.

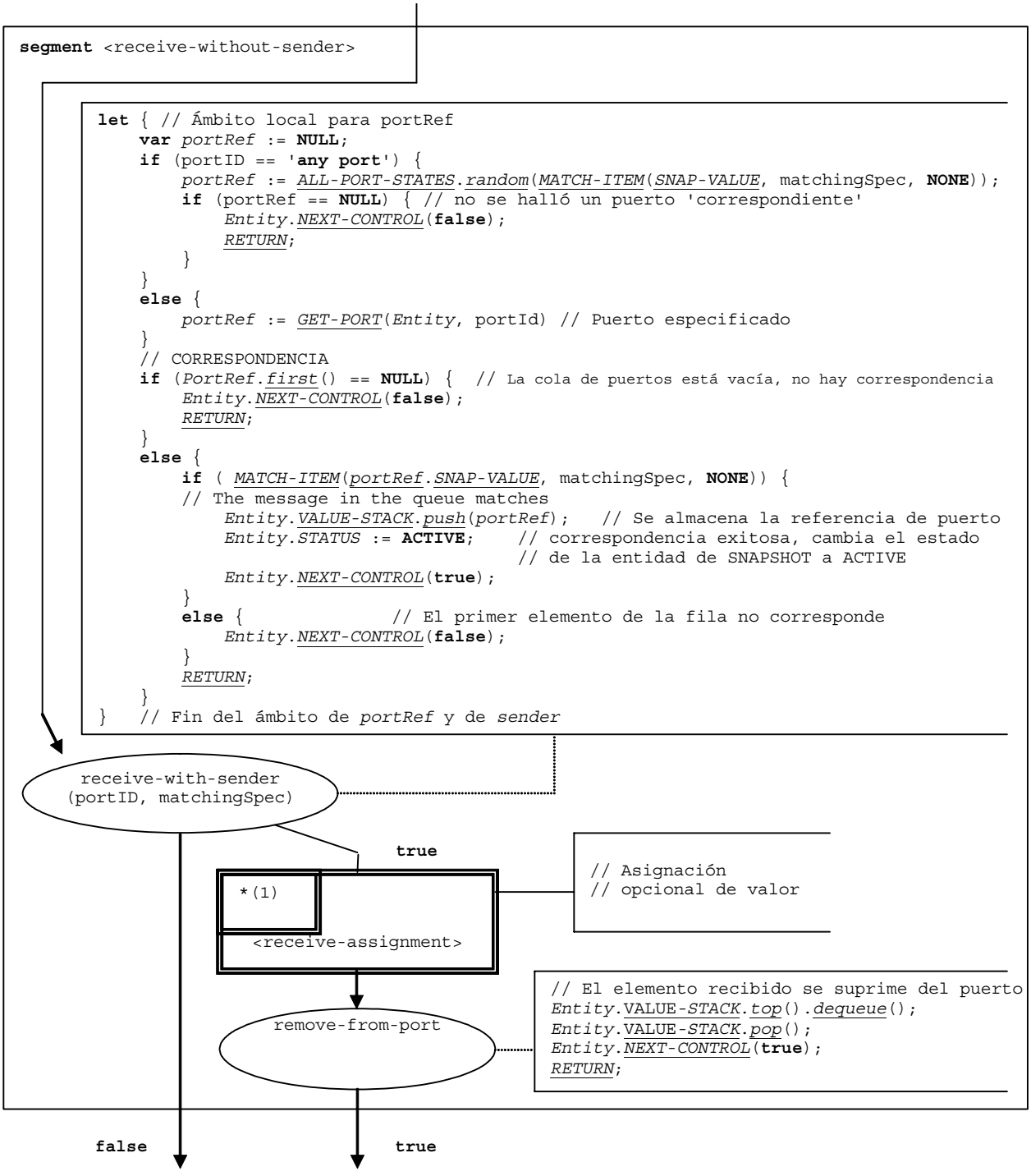


Figura 102/Z.143 – Segmento de diagrama de flujo <receive-without-sender>

9.37.3 Segmento de diagrama de flujo <receive-assignment>

El segmento de diagrama de flujo <receive-assignment> de la figura 103 define la recuperación de información a partir de los mensajes recibidos y su asignación a variables.

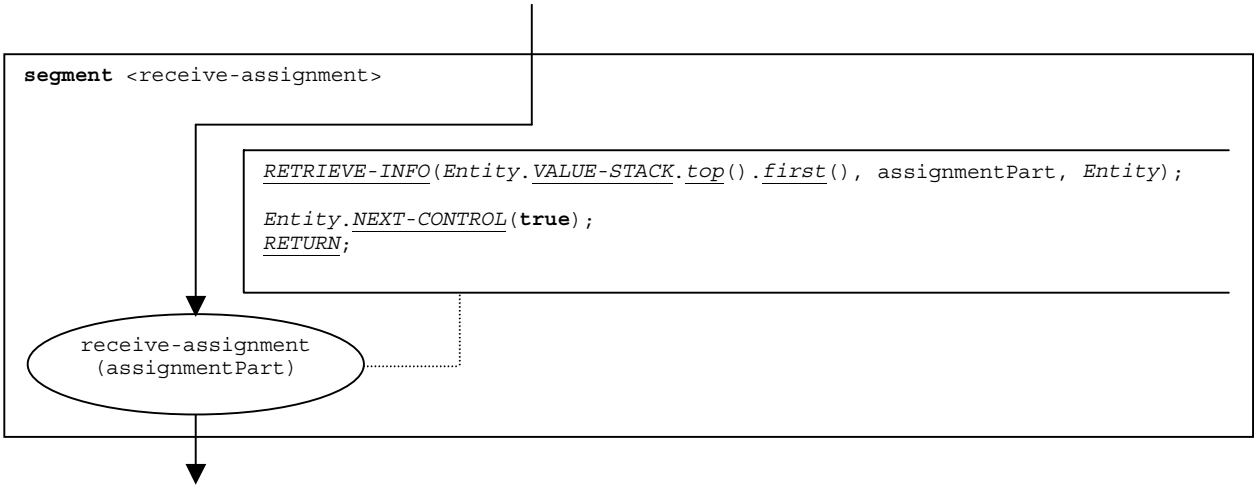


Figura 103/Z.143 – Segmento de diagrama de flujo <receive-assignment>

9.38 Instrucción repeat

La siguiente es la estructura sintáctica de la instrucción **repeat**:

```
repeat
```

La instrucción **repeat** es básicamente una instrucción **return** sin valor devuelto, que también cambia el estado de la entidad a **REPEAT**. El estado **REPEAT** obligará a que se vuelva a evaluar la instrucción **alt** en la que ejecutó la instrucción repeat. El segmento de diagrama de flujo <repeat-stmt> de la figura 104 define la ejecución de las instrucciones **repeat**.

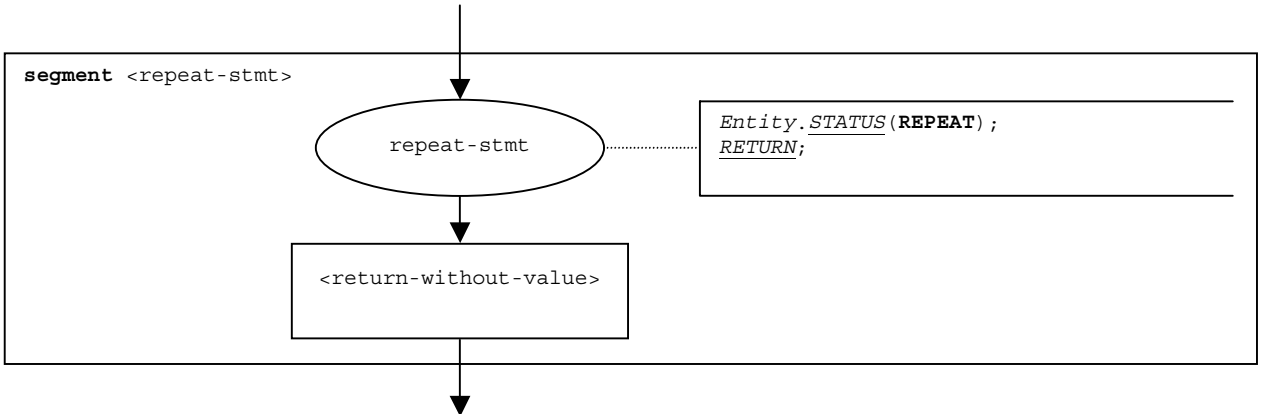


Figura 104/Z.143 – Segmento de diagrama de flujo <repeat-stmt>

9.39 Operación reply

La siguiente es la estructura sintáctica de las operaciones **reply**:

```
<portId>.reply (<replySpec>) [to <component-expression>]
```

El parámetro opcional <component-expression> opcional de la cláusula **to** se refiere a la entidad receptora. Puede ser un valor variable o el valor devuelto por una función.

El segmento de diagrama de flujo <reply-op> de la figura 105 define la ejecución de las operaciones **reply**.

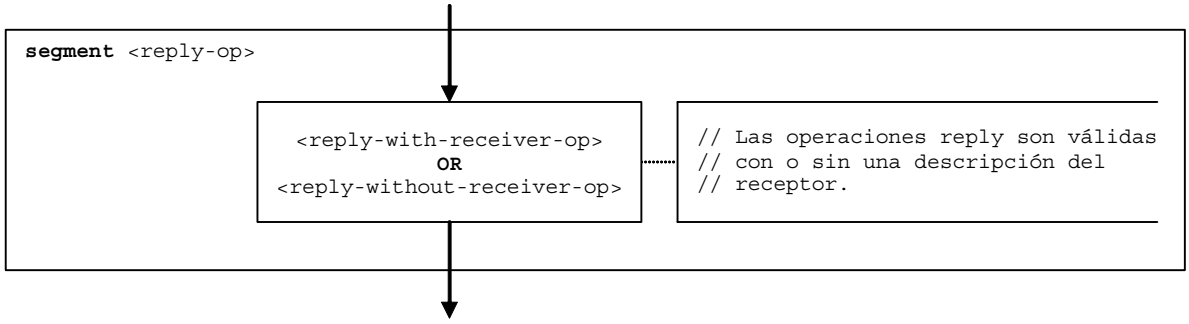


Figura 105/Z.143 – Segmento de diagrama de flujo <reply-op>

9.39.1 Segmento de diagrama de flujo <reply-with-receiver-op>

El segmento de diagrama de flujo <reply-with-receiver-op> de la figura 106 define la ejecución de las operaciones **reply** en la que se especifica el receptor en la forma de una expresión.

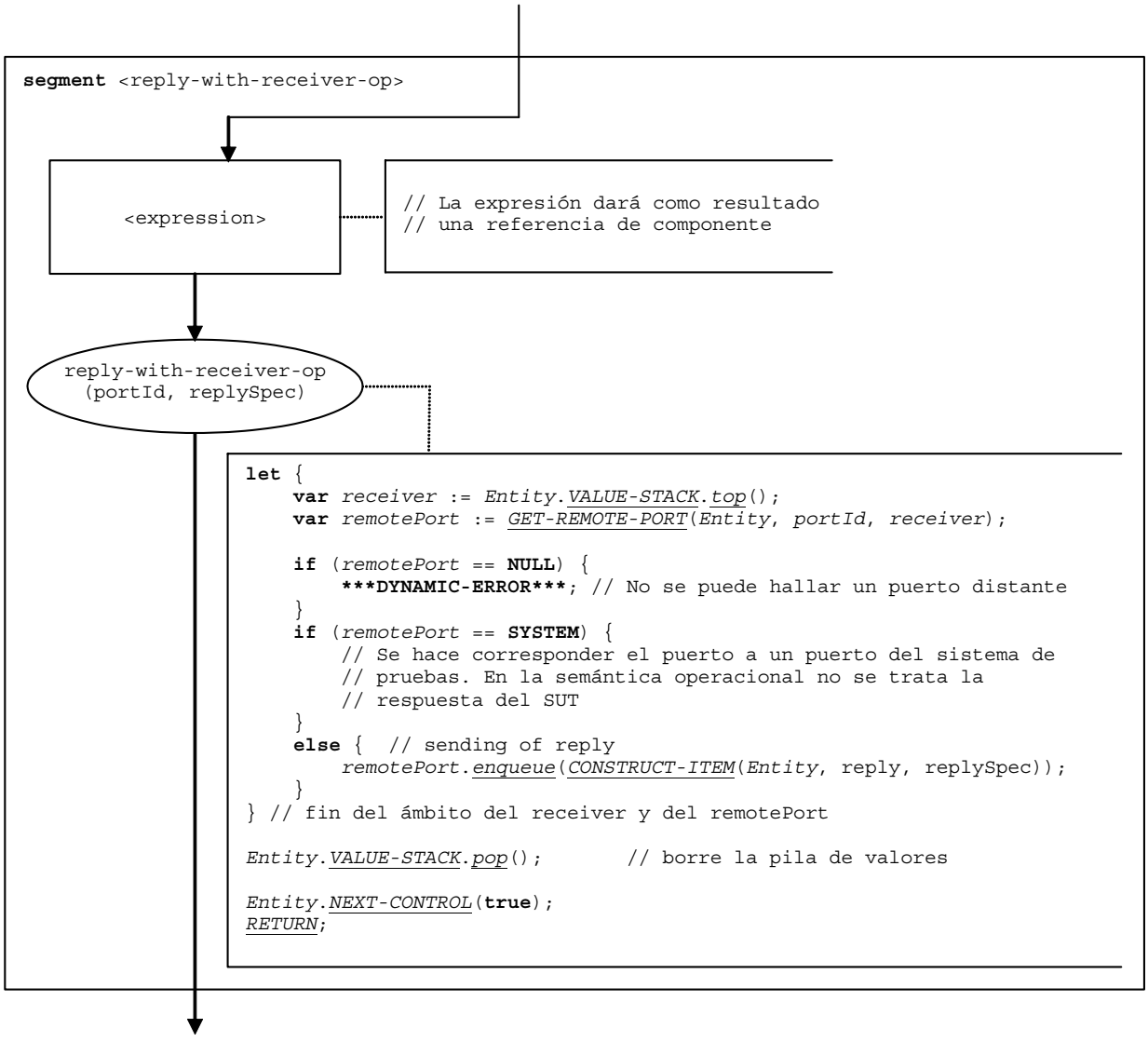


Figura 106/Z.143 – Segmento de diagrama de flujo <reply-with-receiver-op>

9.39.2 Segmento de diagrama de flujo <reply-without-receiver-op>

El segmento de diagrama de flujo `<reply-without-receiver-op>` de la figura 107 define la ejecución de las operaciones reply sin cláusula **`to`**.

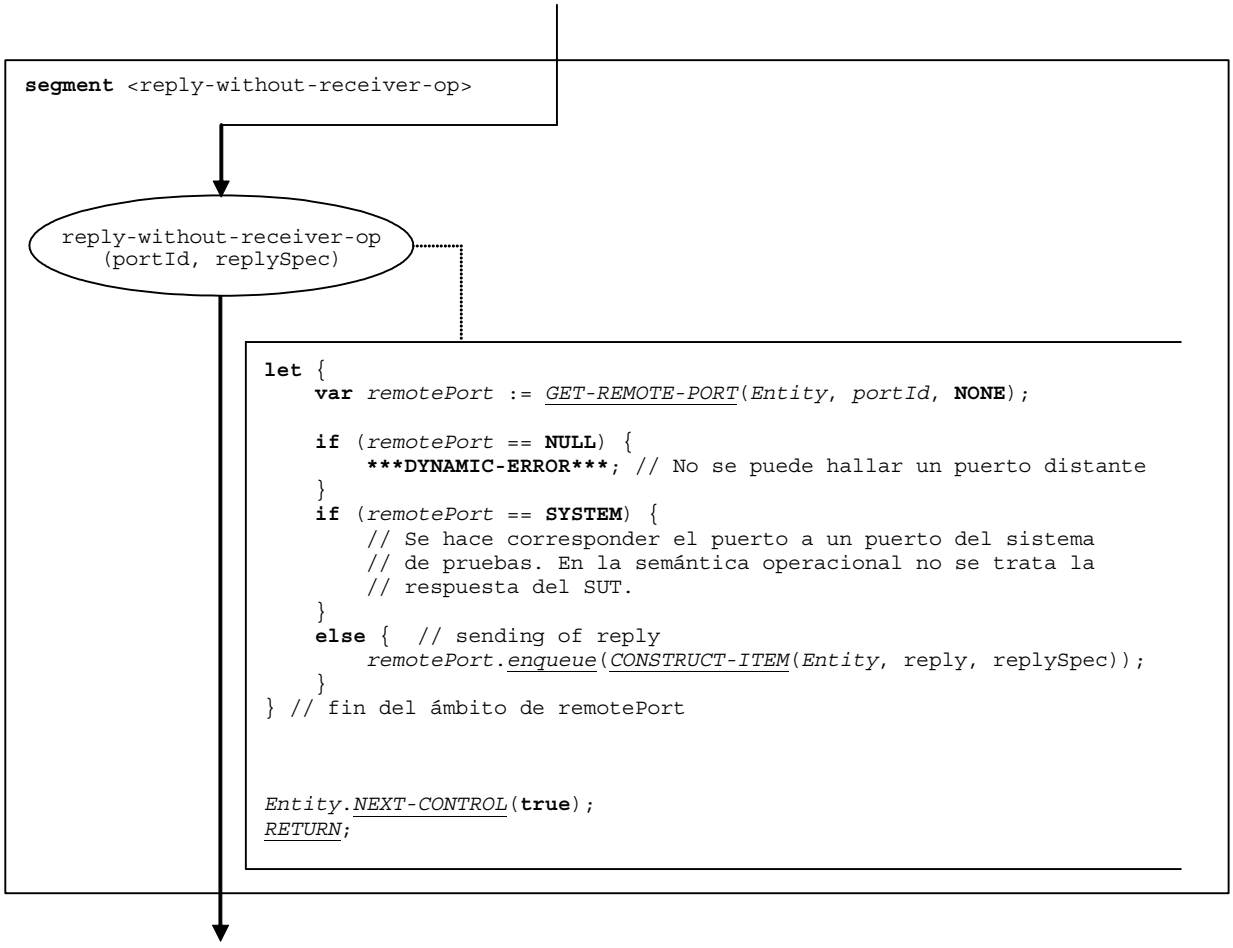


Figura 107/Z.143 – Segmento de diagrama de flujo <reply-without-receiver-op>

9.40 Instrucción return

La siguiente es la estructura sintáctica de las instrucciones **return**:

```
return [<expression>]
```

El parámetro opcional <expression> describe el valor que podría devolver una función. La ejecución de las instrucciones return implica que la unidad de ámbito local deja de ejercer el control, es decir, deben borrarse las variables y temporizadores conocidos únicamente en este ámbito local y debe actualizarse la pila de valores. Las instrucciones **return** tienen el mismo efecto que las operaciones de componente **stop** si se trata de la última instrucción de una descripción de comportamiento.

NOTA – Los casos de prueba y el control del módulo finalizarán siempre con una operación de componente **stop**. Esto se debe a su representación de diagrama de flujo (véase 8.2). Únicamente los otros componentes de prueba pueden finalizar con la instrucción **return**.

El segmento de diagrama de flujo <return-stmt> de la figura 108 define la ejecución de las instrucciones **return**.

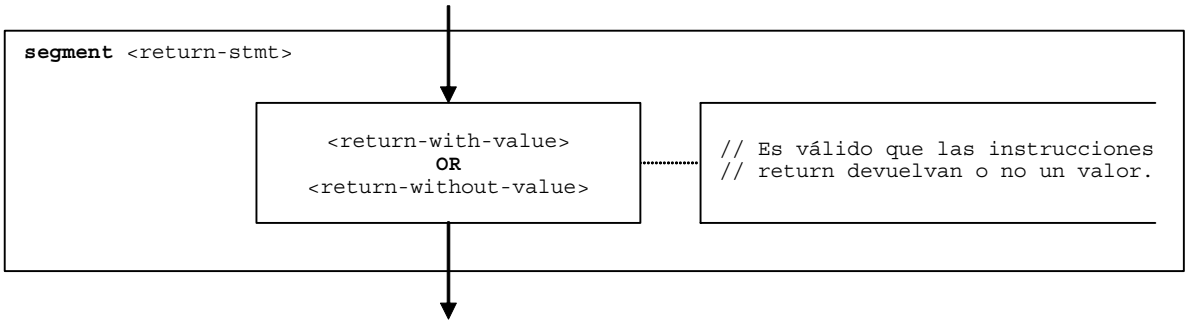


Figura 108/Z.143 – Segmento de diagrama de flujo <return-stmt>

9.40.1 Segmento de diagrama de flujo <return-with-value>

El segmento de diagrama de flujo <return-with-value> de la figura 109 define la ejecución de los **return** que devuelven un valor en forma de expresión.

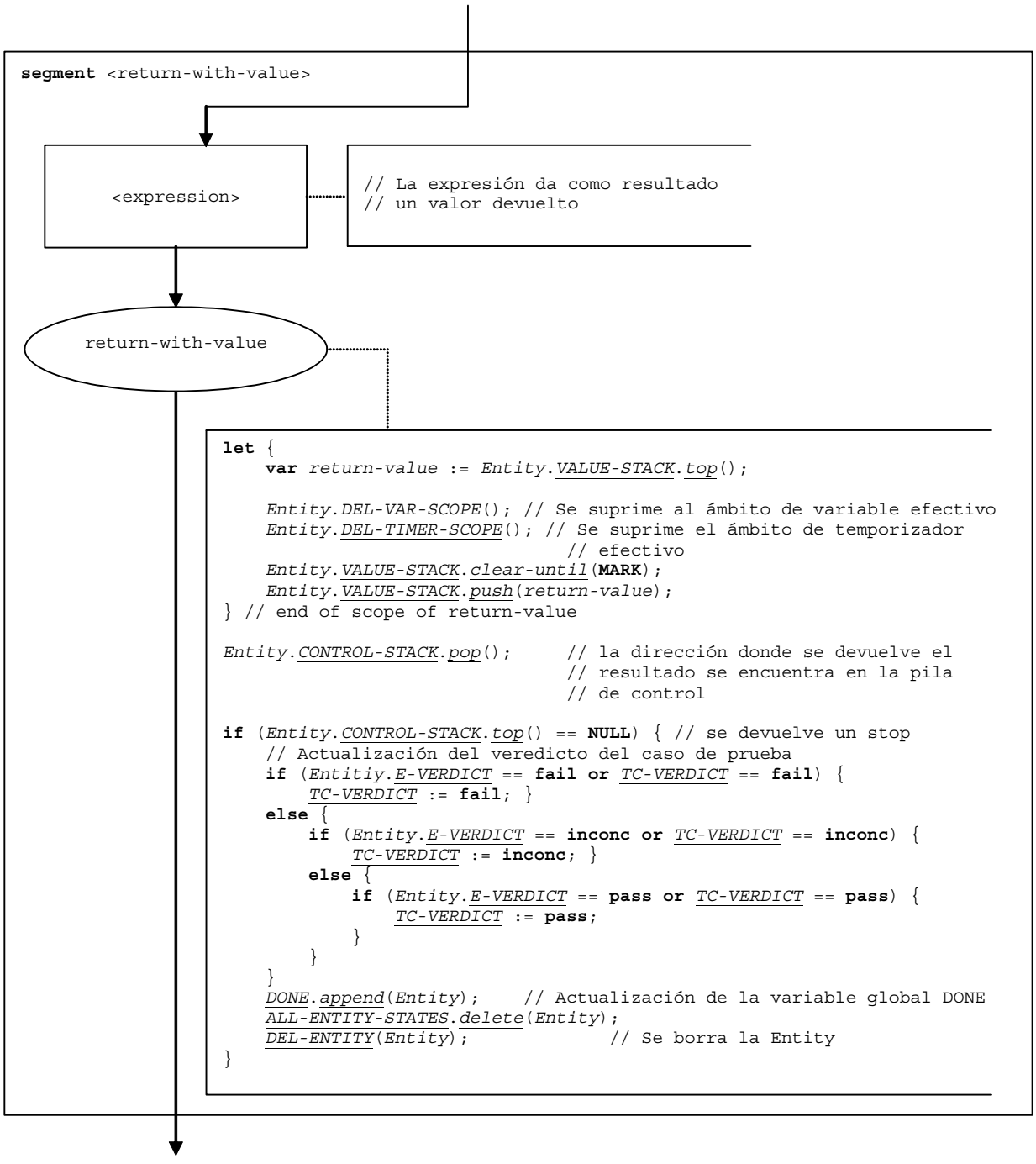


Figura 109/Z.143 – Segmento de diagrama de flujo <return-with-value>

9.40.2 Segmento de diagrama de flujo <return-without-value>

El segmento de diagrama de flujo <return-without-value> de la figura 110 define la ejecución de una instrucción **return** que no devuelve ningún valor.

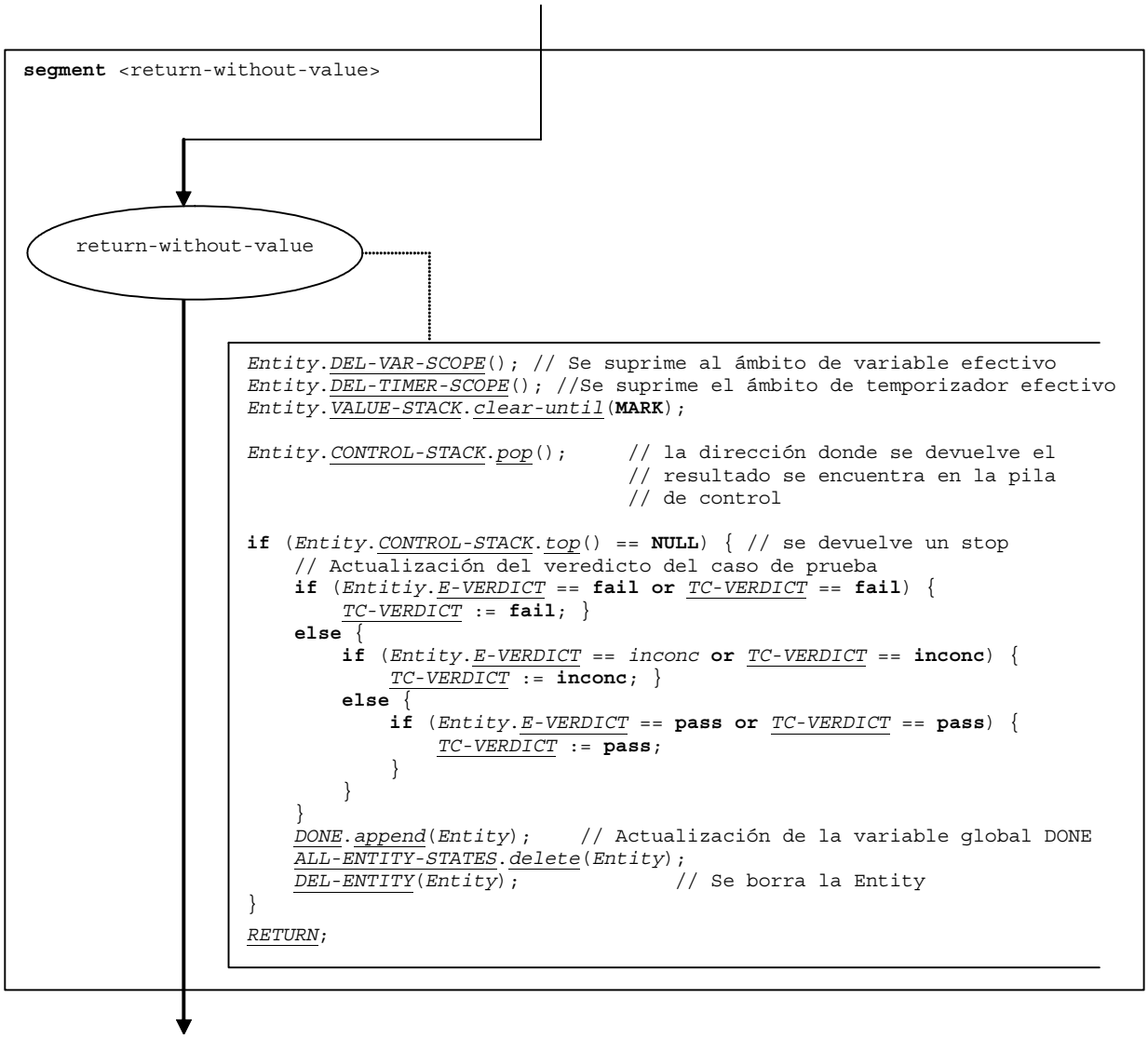


Figura 110/Z.143 – Segmento de diagrama de flujo <return-without-value>

9.41 Operación de componente running

La siguiente es la estructura sintáctica de las operaciones de componente **running**:

```
<component-expression>.running
```

La operación de componente **running** comprueba si cierto componente está en ejecución o si se ha detenido. Se identifica el componente a comprobar mediante una referencia de componente, que puede tomar la forma de una variable o de una función que devuelve un valor, es decir que es una expresión. En aras de simplicidad, se considera que las palabras clave '**all component**' y '**any component**' son expresiones especiales.

El uso de la operación de componente **running** es diferente si se le utiliza para el control de acceso booleano de las instrucciones **alt** o de las operaciones **call** bloqueantes que si se le da otro uso. Si se le utiliza para el control de acceso booleano, el resultado de la operación de componente **running** depende del estado puntual. En los demás casos, evalúa directamente la información de estado.

El resultado de la operación de componente **running** se ingresa en la pila de valores de la entidad que solicitó la operación.

El segmento de diagrama de flujo <running-component-op> de la figura 111 define la ejecución de las operaciones de componente **running**.

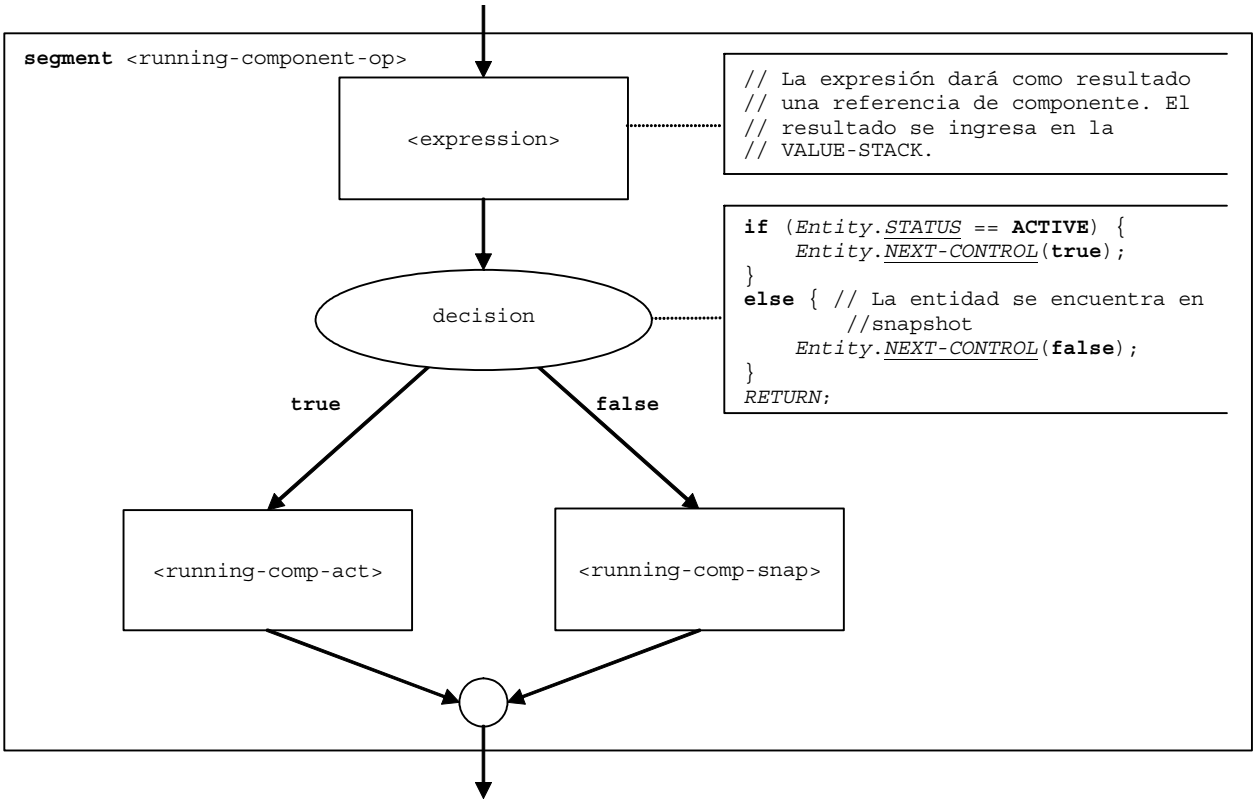


Figura 111/Z.143 – Segmento de diagrama de flujo <running-component-op>

9.41.1 Segmento de diagrama de flujo <running-comp-act>

El segmento de diagrama de flujo <running-comp-act> de la figura 112 describe la ejecución de la operación de componente **running** por fuera de un snapshot, es decir, la entidad está en el estado **ACTIVE**.

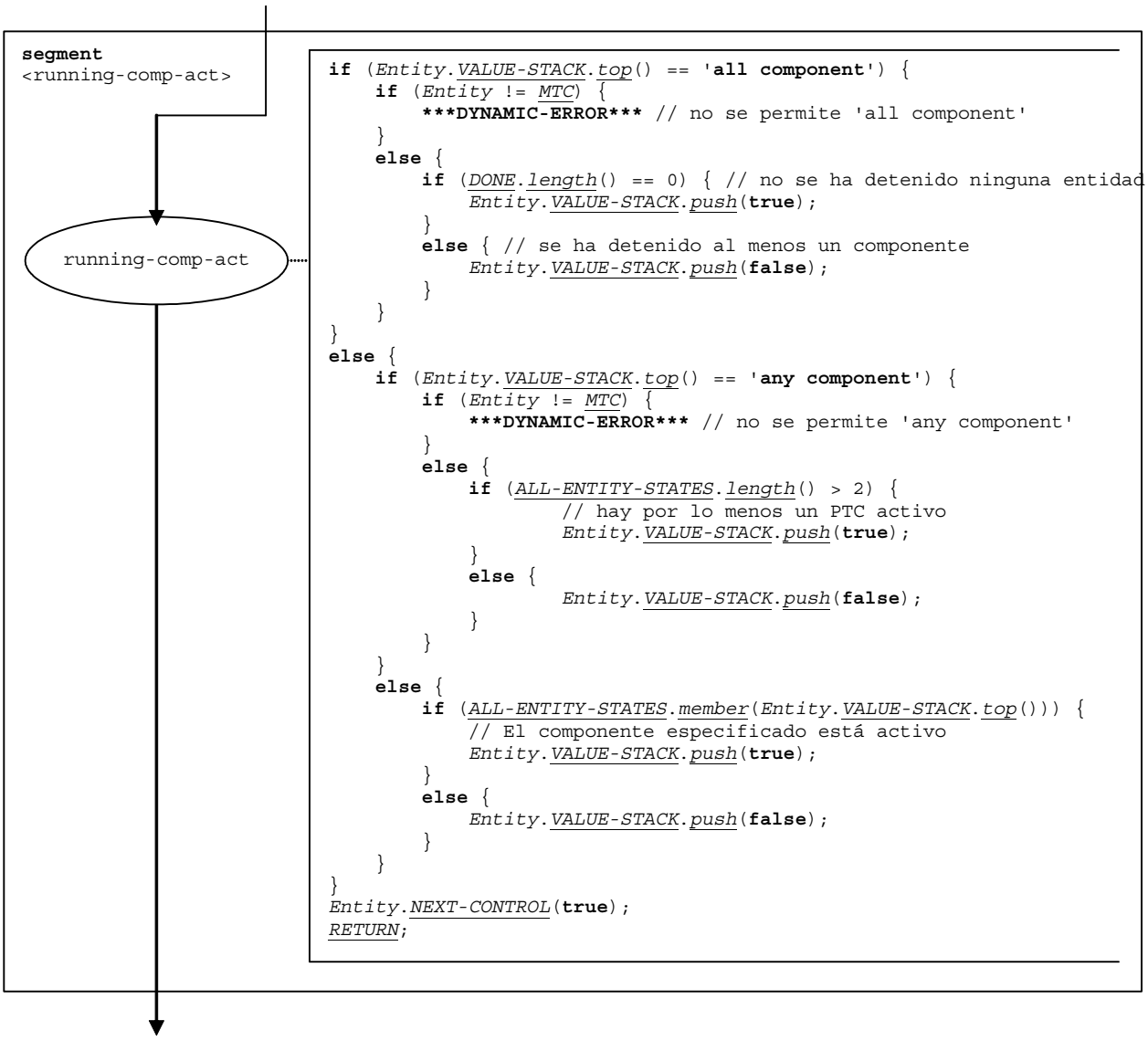


Figura 112/Z.143 – Segmento de diagrama de flujo <running-comp-act>

9.41.2 Segmento de diagrama de flujo <running-comp-snap>

El segmento de diagrama de flujo <running-comp-snap> de la figura 113 describe la ejecución de la operación de componente **running** durante la evaluación de un estado puntual, es decir, la entidad está en el estado **SNAPSHOT**.

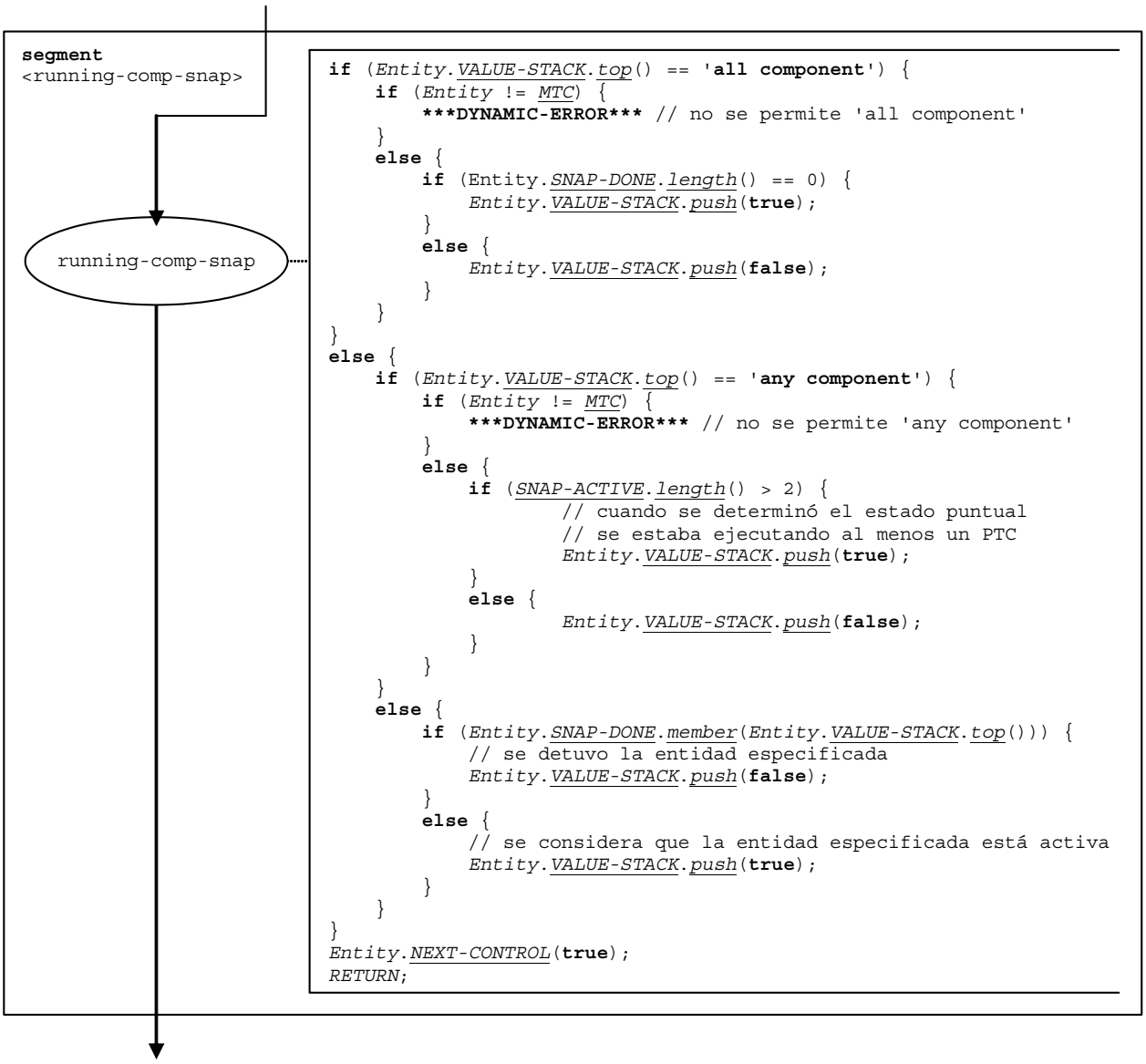


Figura 113/Z.143 – Segmento de diagrama de flujo <running-comp-snap>

9.42 Operación de temporizador running

La siguiente es la estructura sintáctica de las operaciones de temporizador **`running`**:

```
<timerId>.running
```

El segmento de diagrama de flujo <running-timer-op> de la figura 114 define la ejecución de las operaciones de temporizador **`running`**.

El uso de las operaciones de temporizador **`running`** es diferente si se le utiliza para el control de acceso booleano de las instrucciones **`alt`** o de las operaciones **`call`** bloqueantes que si se le da otro uso. Si se le utiliza para el control de acceso booleano, el resultado de las operaciones de temporizador **`running`** depende del estado puntual, es decir, el valor de *SNAP-STATUS* de la vinculación de temporizador. En los demás casos, el valor de *STATUS* de la vinculación de temporizador determina el resultado de la operación.

Se considera que la palabra clave **`any`** es un valor especial de `timerId`.

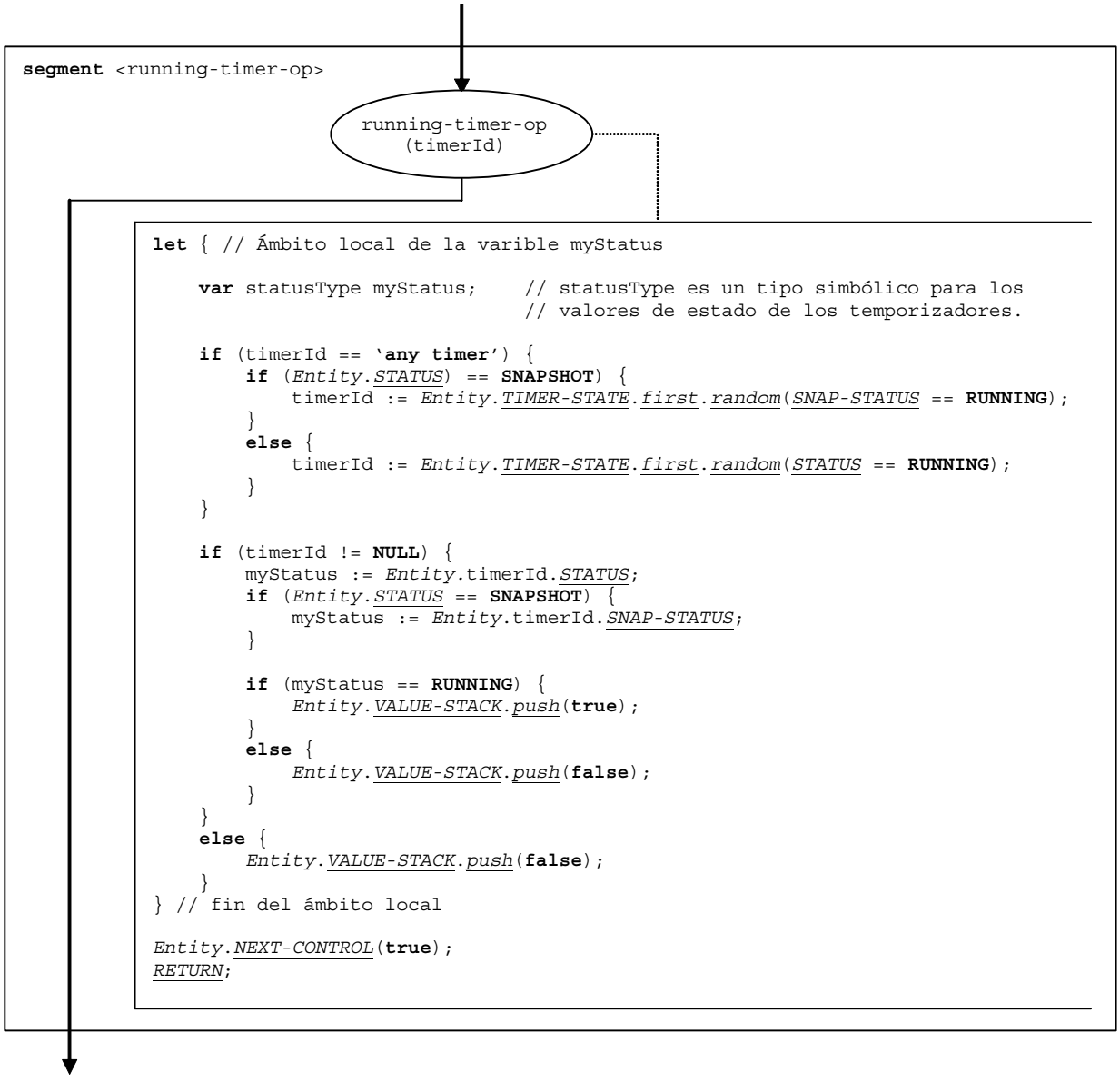


Figura 114/Z.143 – Segmento de diagrama de flujo <running-timer-op>

9.43 Operación self

La siguiente es la estructura sintáctica de las operaciones **self**:

```
self
```

El segmento de diagrama de flujo <self-op> de la figura 115 define la ejecución de las operaciones **self**.

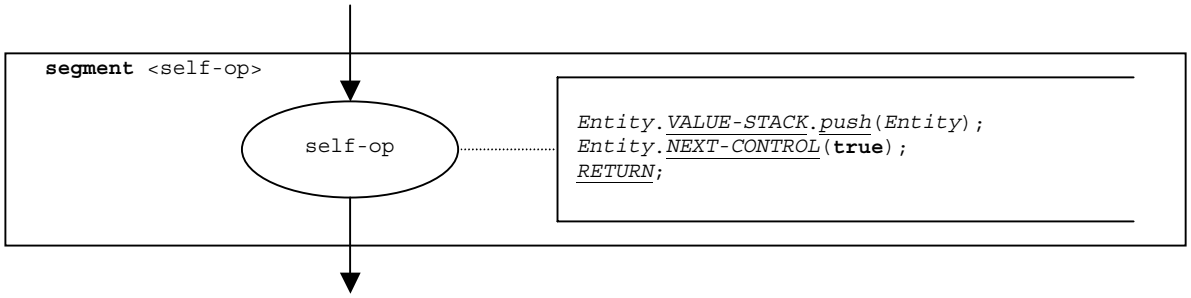


Figura 115/Z.143 – Segmento de diagrama de flujo <self-op>

9.44 Operación send

La siguiente es la estructura sintáctica de las operaciones **send**:

```
<portId>.send (<send-spec>) [to <component-expression>]
```

El parámetro opcional <component-expression> de la cláusula **to** se refiere a la entidad receptora. Puede ser un valor variable o el valor devuelto por una función.

El segmento de diagrama de flujo <send-op> de la figura 116 define la ejecución de las operaciones **send**.

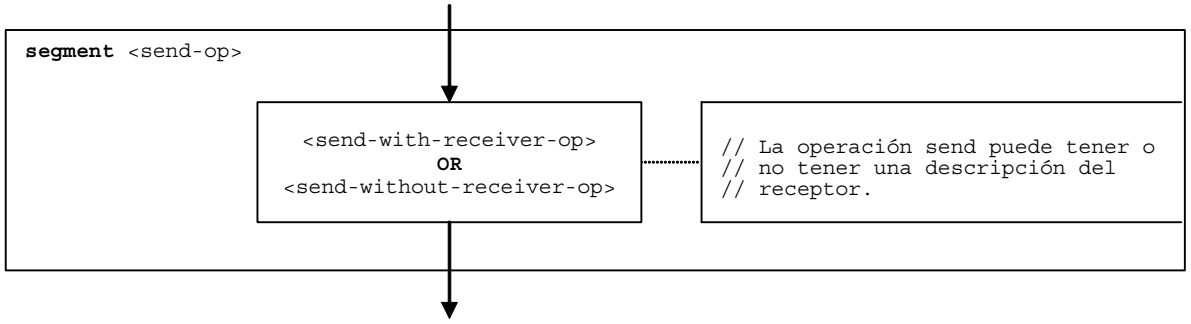


Figura 116/Z.143 – Segmento de diagrama de flujo <send-op>

9.44.1 Segmento de diagrama de flujo <send-with-receiver-op>

El segmento de diagrama de flujo <send-with-receiver-op> de la figura 117 define la ejecución de las operaciones **send**, en las que se especifica el receptor en la forma de una expresión.

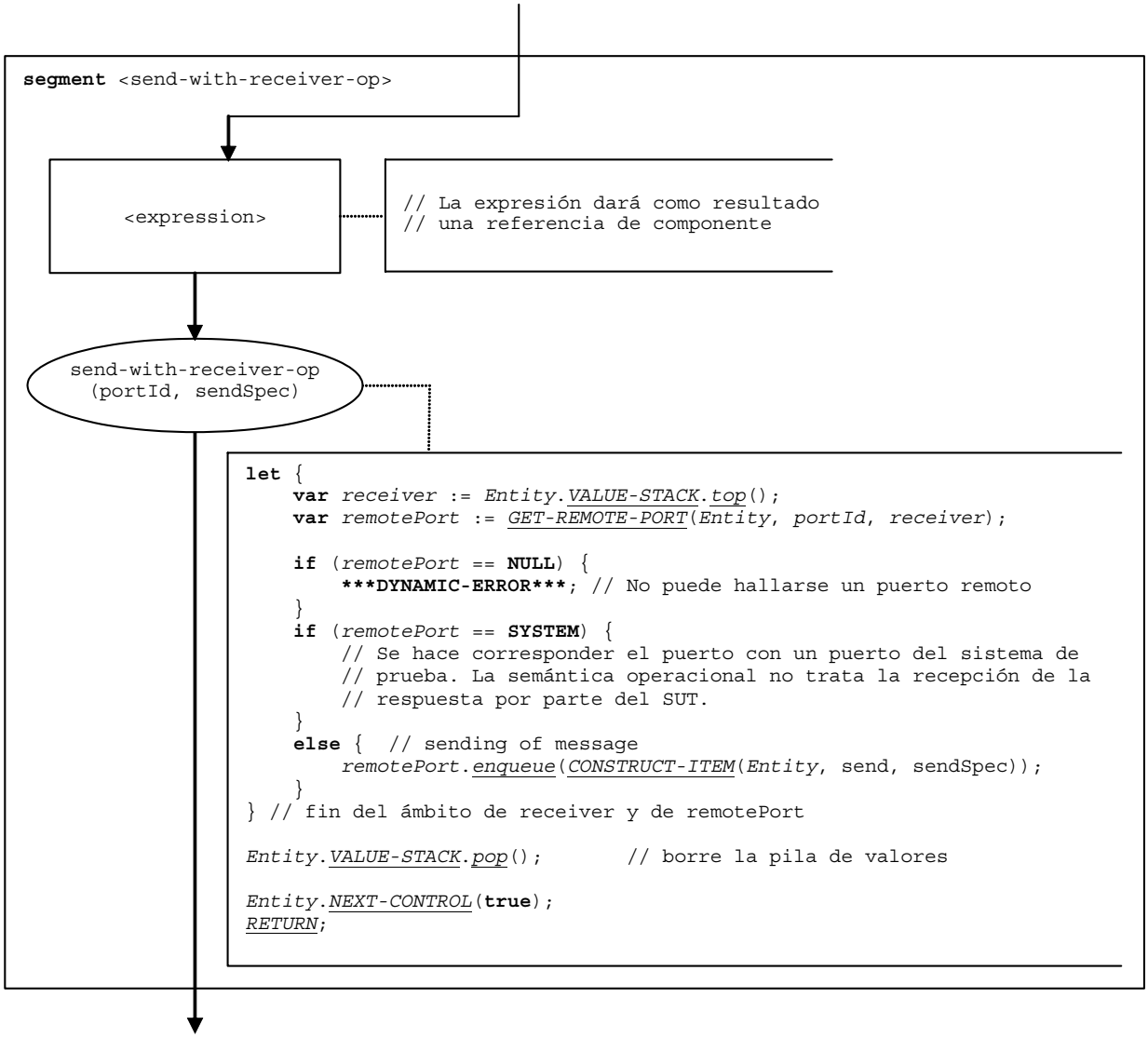


Figura 117/Z.143 – Segmento de diagrama de flujo <send-with-receiver-op>

9.44.2 Segmento de diagrama de flujo <send-without-receiver-op>

El segmento de diagrama de flujo <send-without-receiver-op> de la figura 118 define la ejecución de las operaciones **send** sin cláusula **to**.

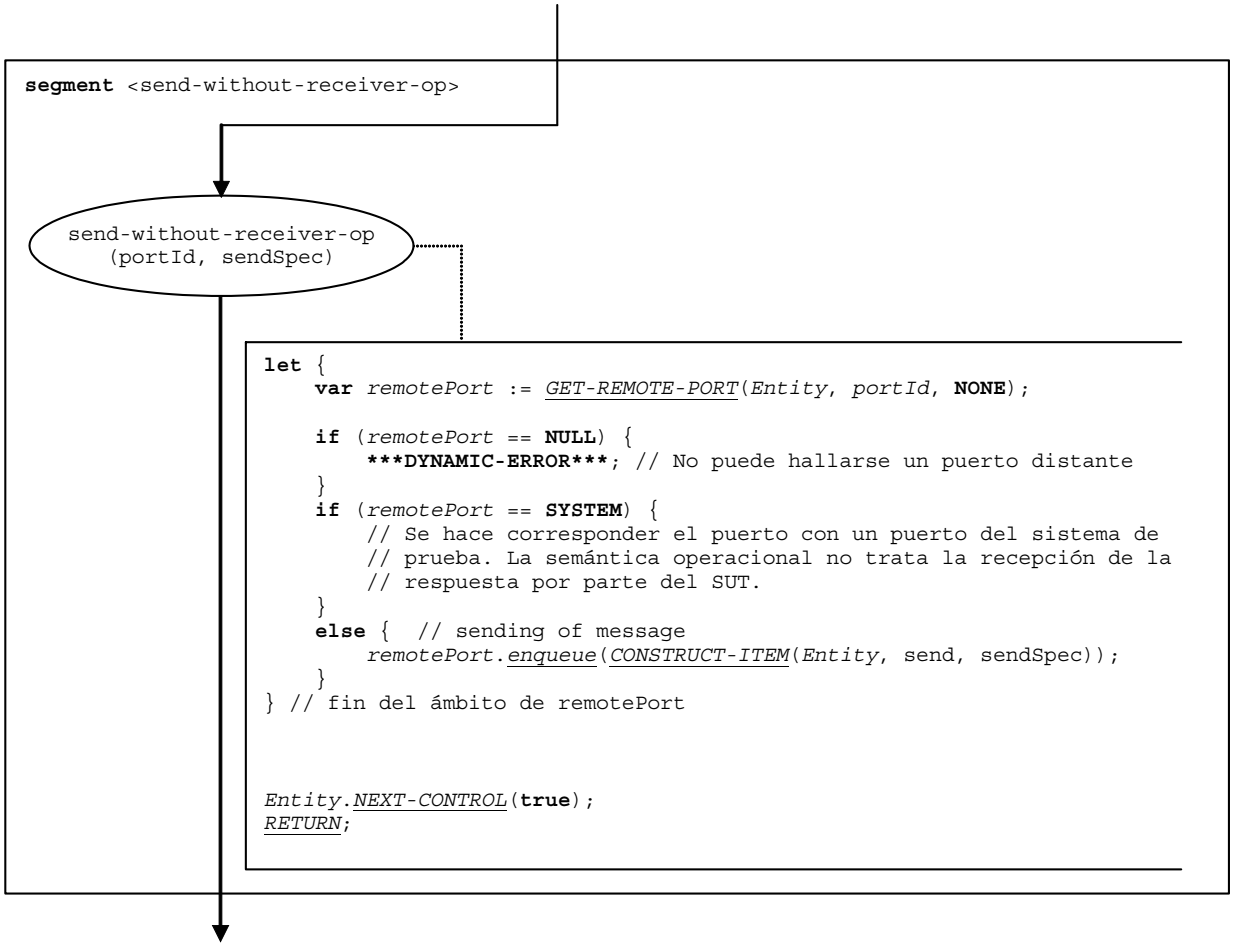


Figura 118/Z.143 – Segmento de diagrama de flujo <send-without-receiver-op>

9.45 Operación setverdict

La siguiente es la estructura sintáctica de las operaciones **setverdict**:

setverdict(<verdicttype-expression>)

El parámetro <verdicttype-expression> de la operación **setverdict** es una expresión que dará como resultado un valor de tipo **verdicttype**, es decir, **none**, **pass**, **inconc** o **fail**. La expresión se evalúa antes de que se aplique la operación **setverdict**.

El segmento de diagrama de flujo <setverdict-op> de la figura 119 define la ejecución de las operaciones **setverdict**.

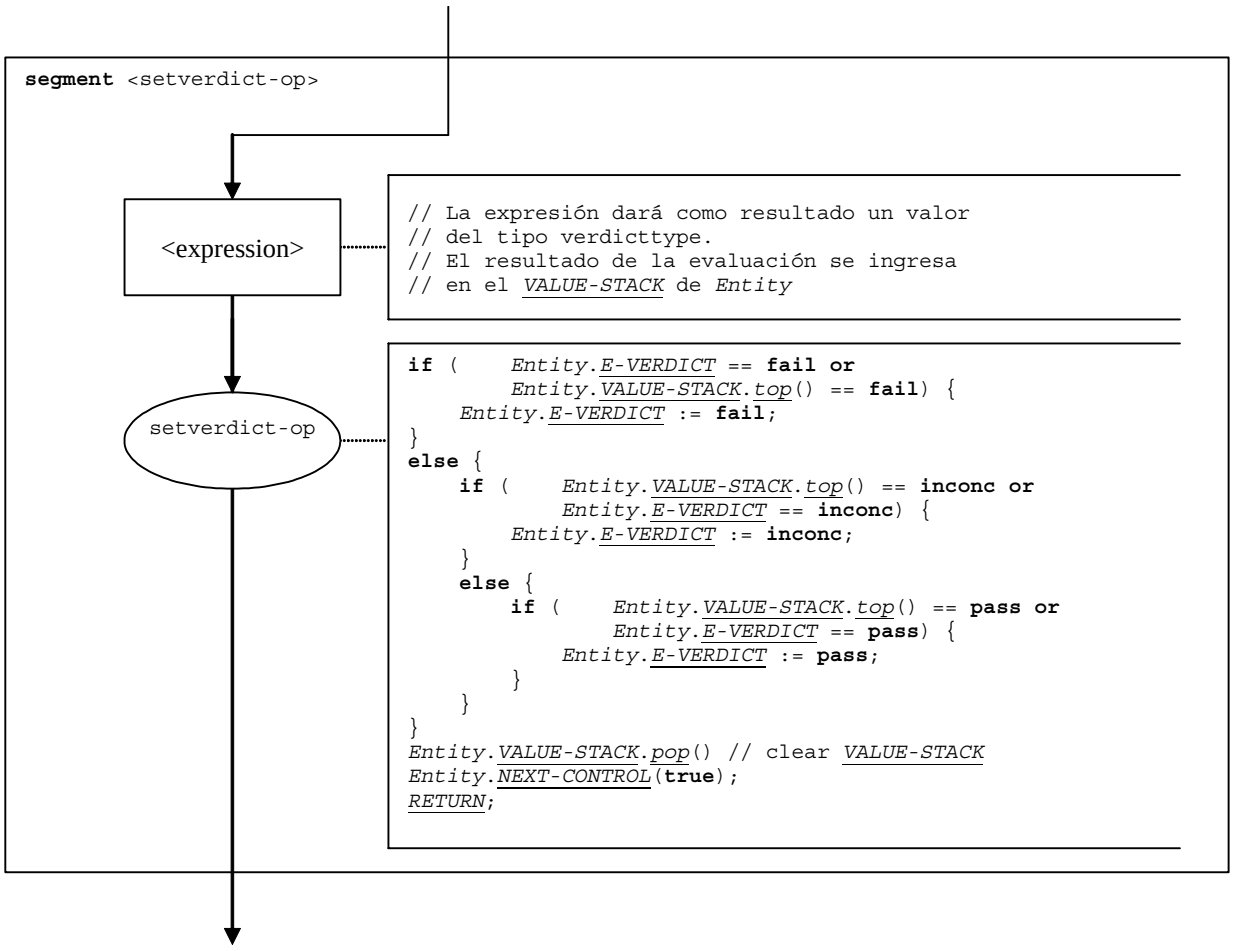


Figura 119/Z.143 – Segmento de diagrama de flujo <setverdict-op>

9.46 Operación de componente start

La siguiente es la estructura sintáctica de las operaciones de componente **start**:

<component-expression>.**start**(<function-name>(<act-par-desc$_1$>,..., <act-par-desc$_n$>))

Las operaciones de componente **start** inician un componente recientemente creado. El componente que se iniciará se identifica mediante una referencia a componente. La referencia puede estar almacenada en una variable o puede ser devuelta por una función, es decir que es una expresión que da como resultado una referencia de componente.

<function-name> indica el nombre de la función que define el comportamiento del nuevo componente, mientras que <act-par-descr$_1$>, ..., <act-par-descr$_n$> son una descripción de los valores efectivos de los parámetros de <function-name>. En las funciones a las que se hace referencia en las operaciones de componente **start** sólo se admiten parámetros por valor. La descripción de los parámetros efectivos son expresiones que deben evaluarse antes de que se pueda ejecutar la solicitud. El tratamiento de parámetros por valor efectivos o formales es similar al tratamiento que estos reciben en solicitudes de funciones (véase 9.24).

El segmento de diagrama de flujo <start-component-op> de la figura 120 define la ejecución de las operaciones de componente **start**. Las operaciones de componente start se efectúan en cuatro pasos. En el primer paso se crea un registro de solicitudes. En el segundo paso se calculan los valores de parámetro efectivos. En el tercer paso se recupera la referencia del componente que ha de iniciarse, y en el cuarto paso se otorga el control y el registro de solicitudes al nuevo componente.

NOTA – El segmento de diagrama de flujo de la figura 120 incluye el manejo de los parámetros por referencia (<ref-var-par-calc>). Los parámetros por referencia son necesarios para explicar los parámetros por referencia de los casos de prueba. La semántica operacional supone que el MTC maneja estos parámetros.

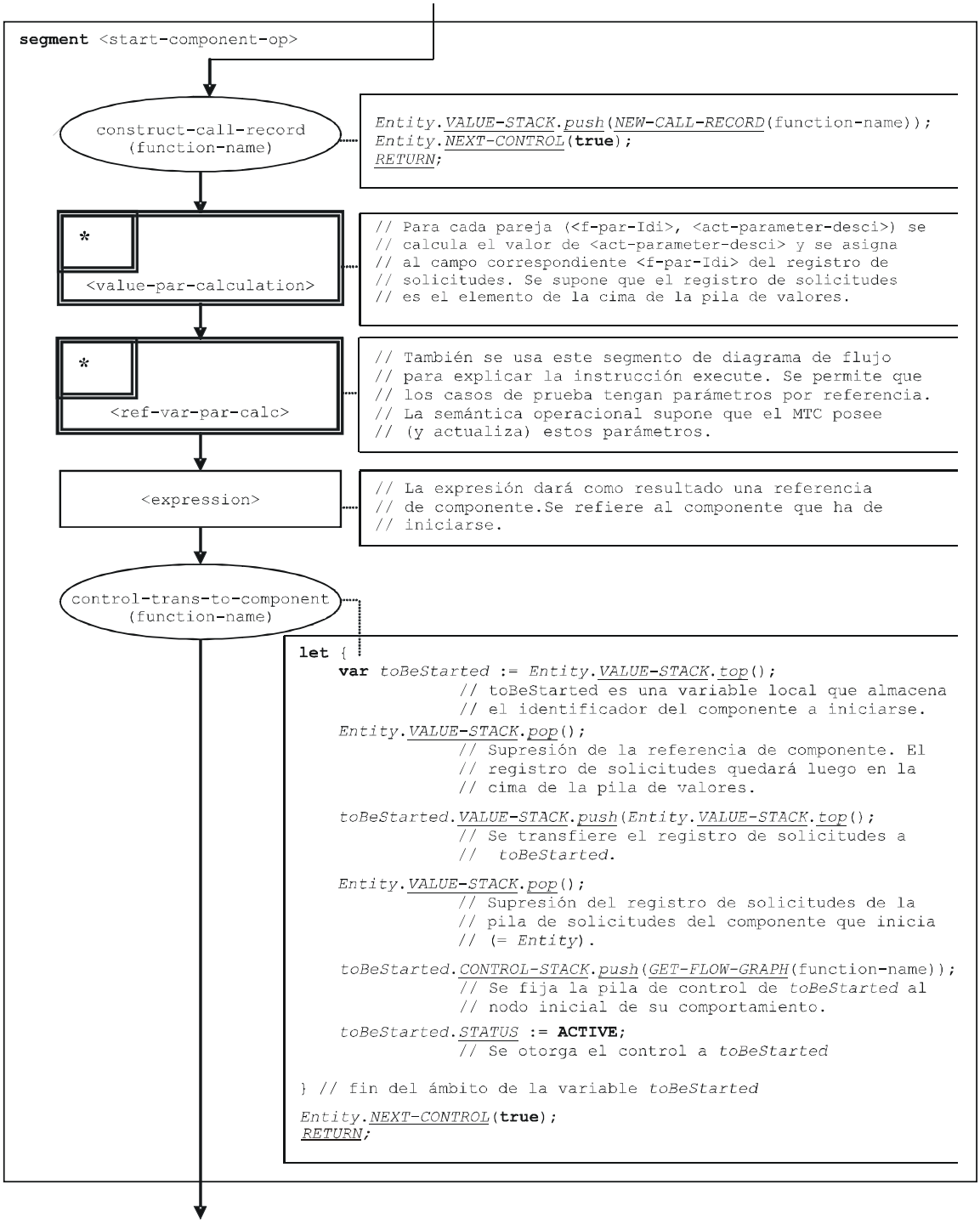


Figura 120/Z.143 – Segmento de diagrama de flujo <start-component-op>

9.47 Operación de puerto start

La siguiente es la estructura sintáctica de las operaciones de puerto **start**:

```
<portId>.start
```

El segmento de diagrama de flujo <start-port-op> de la figura 121 define la ejecución de las operaciones de puerto **start**.

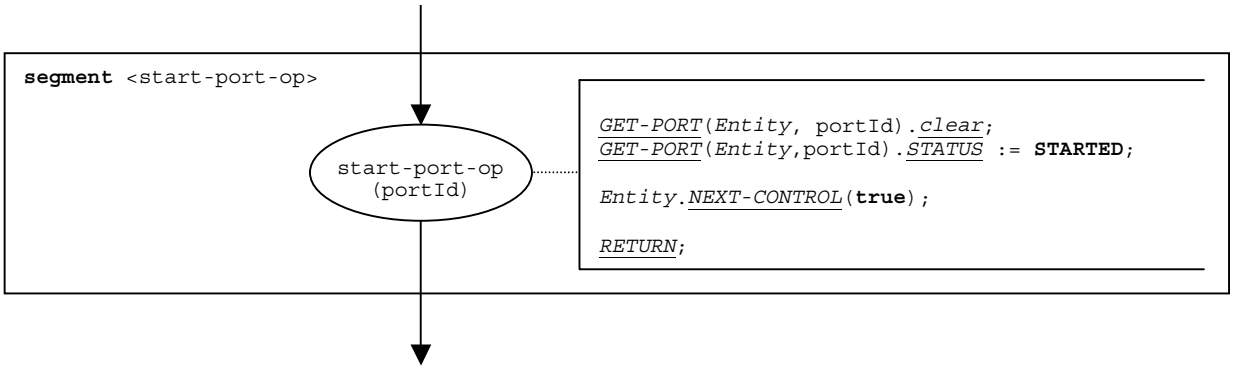


Figura 121/Z.143 – Segmento de diagrama de flujo <start-port-op>

9.48 Operación de temporizador start

La siguiente es la estructura sintáctica de las operaciones de temporizador **start**:

```
<timerId>.start [(<float-expression>)]
```

El parámetro opcional <float-expression> de la operación de temporizador **start** indica la duración efectiva del temporizador. De no proporcionarse, la operación **start** utilizará la duración por defecto. Se deberá evaluar la expresión antes de que se aplique la operación **start**. Se ingresará el resultado de la evaluación en la *VALUE-STACK* de *Entity*

El segmento de diagrama de flujo <start-timer-op> de la figura 122 define la ejecución de las operaciones de temporizador **start**.

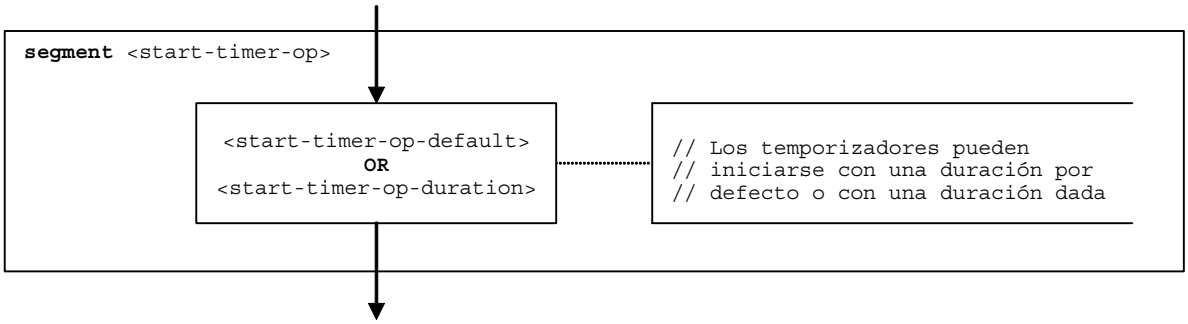


Figura 122/Z.143 – Segmento de diagrama de flujo <start-timer-op>

9.48.1 Segmento de diagrama de flujo <start-timer-op-default>

El segmento de diagrama de flujo <start-timer-op-default> de la figura 123 define la ejecución de las operaciones de temporizador **start** con un valor por defecto.

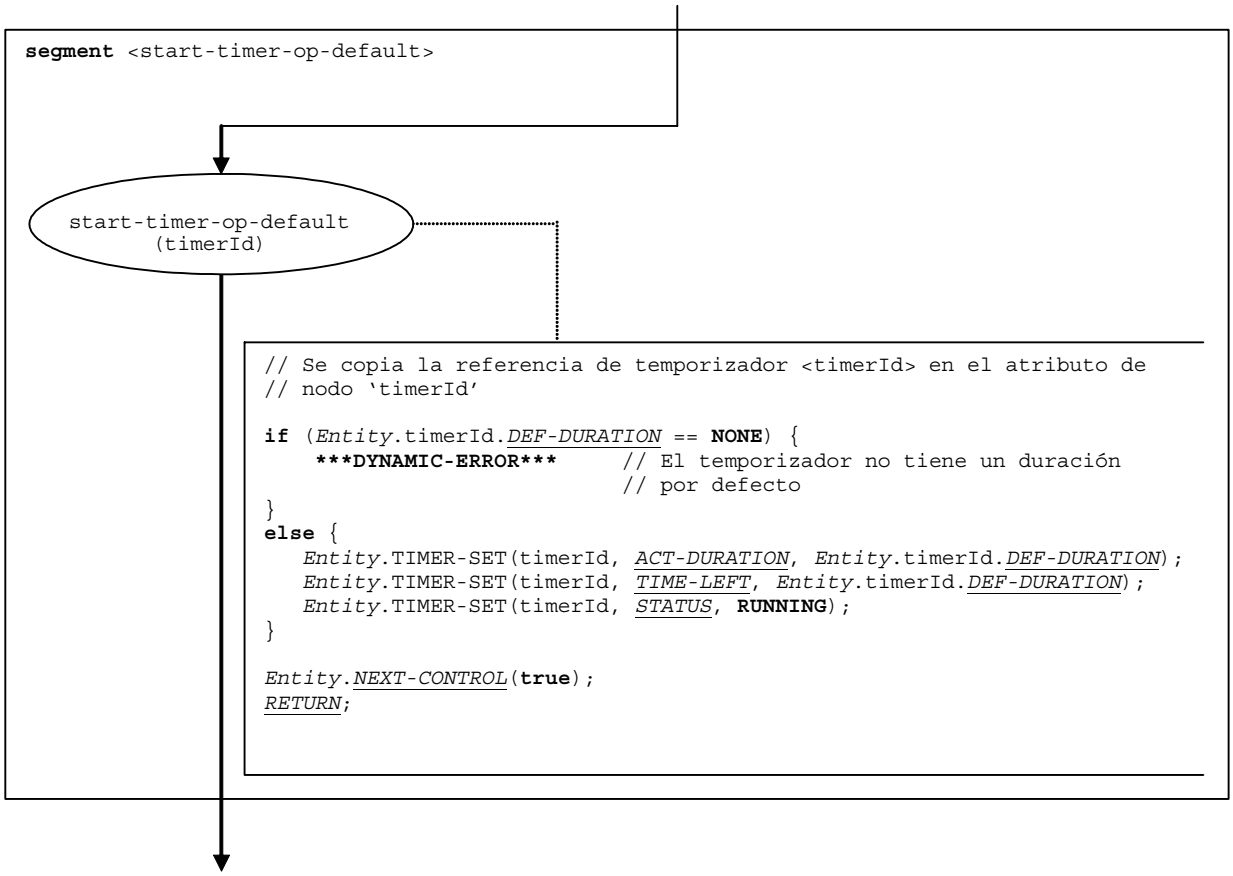


Figura 123/Z.143 – Segmento de diagrama de flujo <start-timer-op-default>

9.48.2 Segmento de diagrama de flujo <start-timer-op-duration>

El segmento de diagrama de flujo <start-timer-op-duration> de la figura 124 define la ejecución de las operaciones de temporizador **start** con una duración determinada.

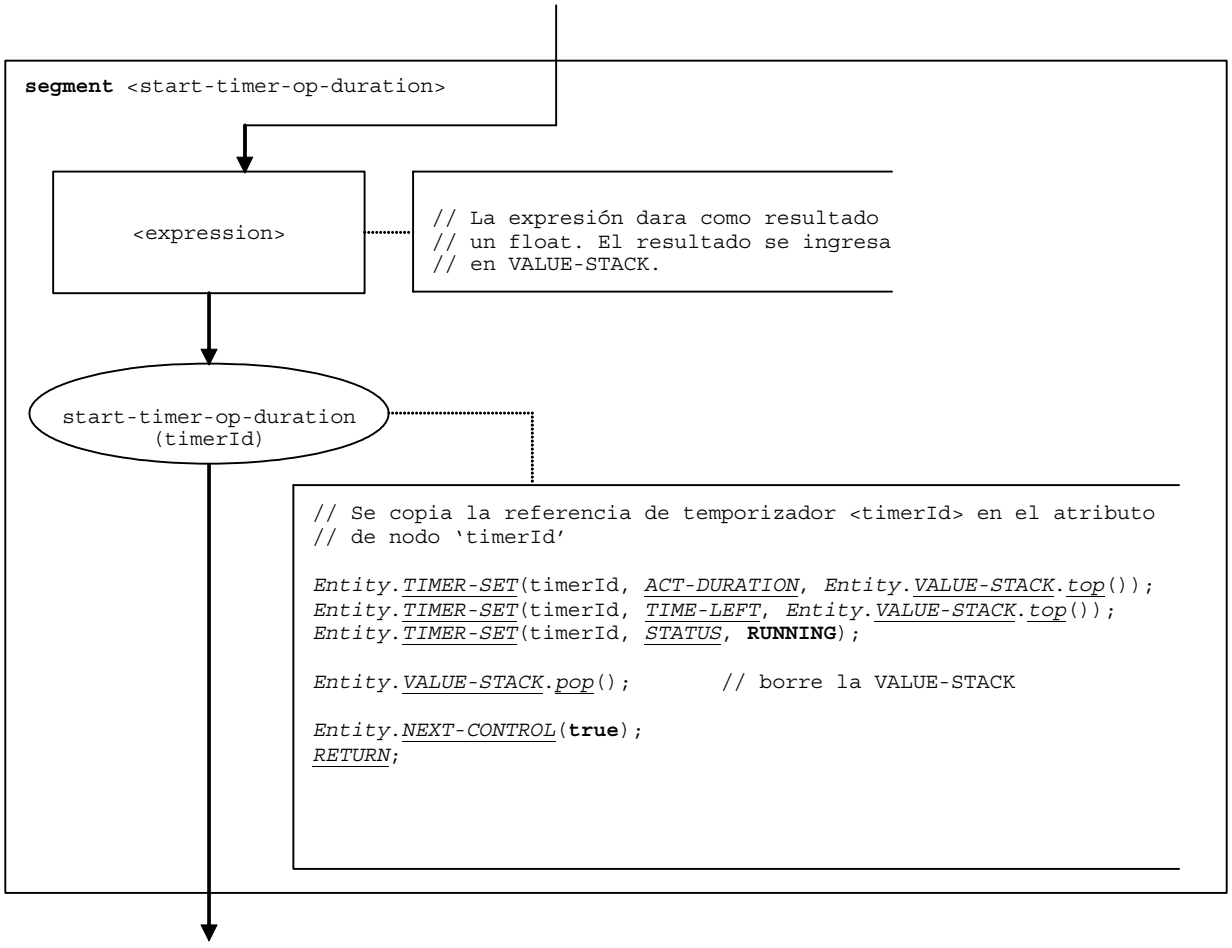


Figura 124/Z.143 – Segmento de diagrama de flujo <start-timer-op-duration>

9.49 Operación de componente stop

La siguiente es la estructura sintáctica de la instrucción de componente **stop**:

```
<component-expression>.stop
```

Las operaciones de componente **stop** detienen el componente especificado. Si el MTC es detenido (por ejemplo. **mtc.stop**) o si se detiene a sí mismo (por ejemplo **self.stop**), se detendrán todos los componentes de prueba. El MTC puede detener todos los componentes de prueba paralelos utilizando la palabra clave **all**, es decir, **all component.stop**.

Se identifica el componente que ha de detenerse mediante una referencia de componente presentada como una expresión, es decir, un valor o una función que devuelve un valor. En aras de simplicidad, se considera que la palabra clave '**all component**' es un valor especial de <component-expression>. Se evalúan las operaciones **mtc** y **self** de conformidad con las cláusulas 9.33 y 9.43.

El segmento de diagrama de flujo <stop-component-op> de la figura 125 define la ejecución de las operaciones de componente **stop**.

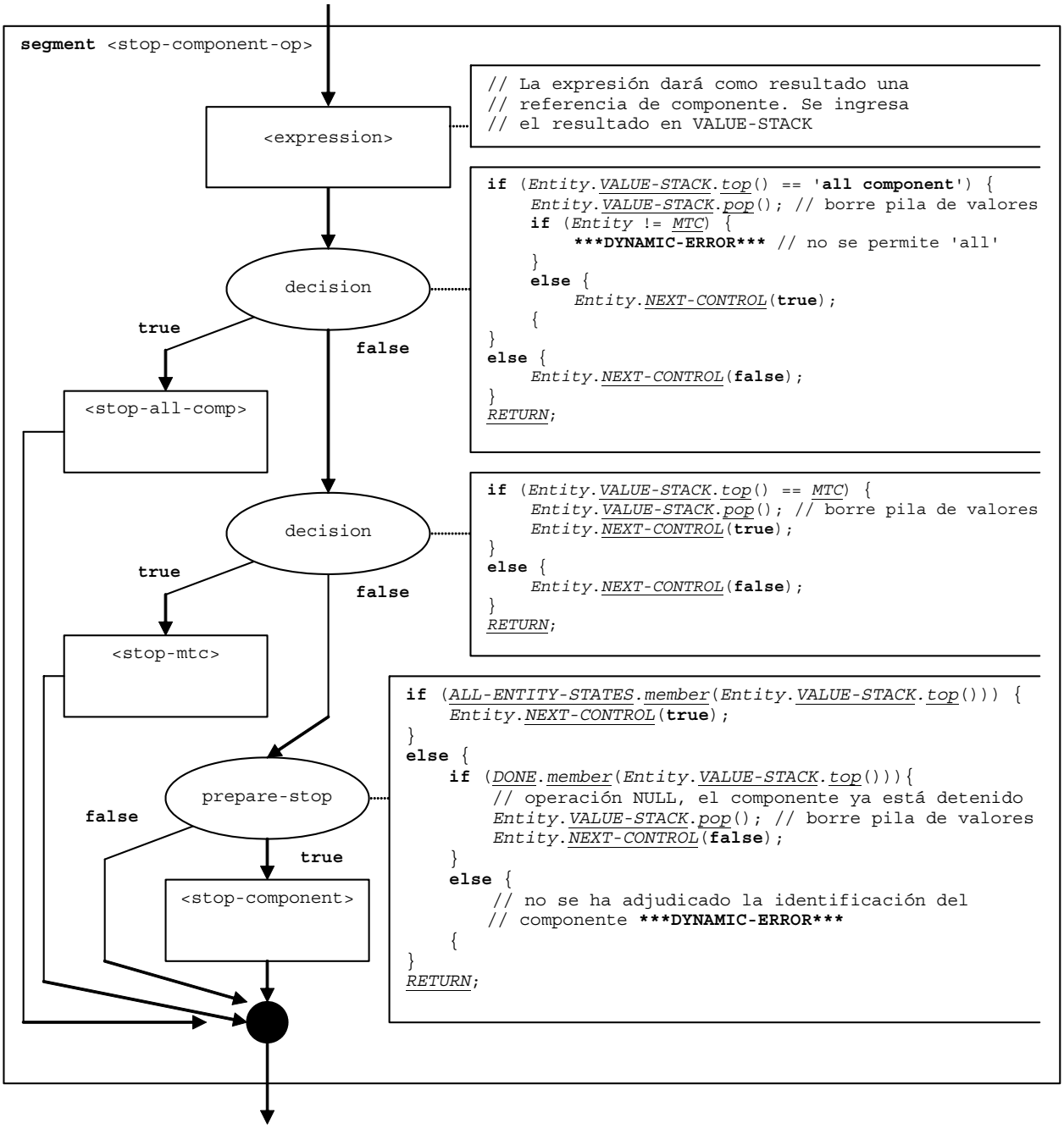


Figura 125/Z.143 – Segmento de diagrama de flujo <stop-component-op>

9.49.1 Segmento de diagrama de flujo <stop-mtc>

El segmento de diagrama de flujo <stop-mtc> de la figura 126 describe la detención de un MTC de un caso de prueba. Como resultado, finaliza el caso de prueba, es decir, el veredicto final se calcula y se ingresa en la pila de valores del control del módulo, se liberan todos los recursos, se vacía la lista DONE del estado del módulo y se finalizan todos los componentes de prueba, incluido el MTC.

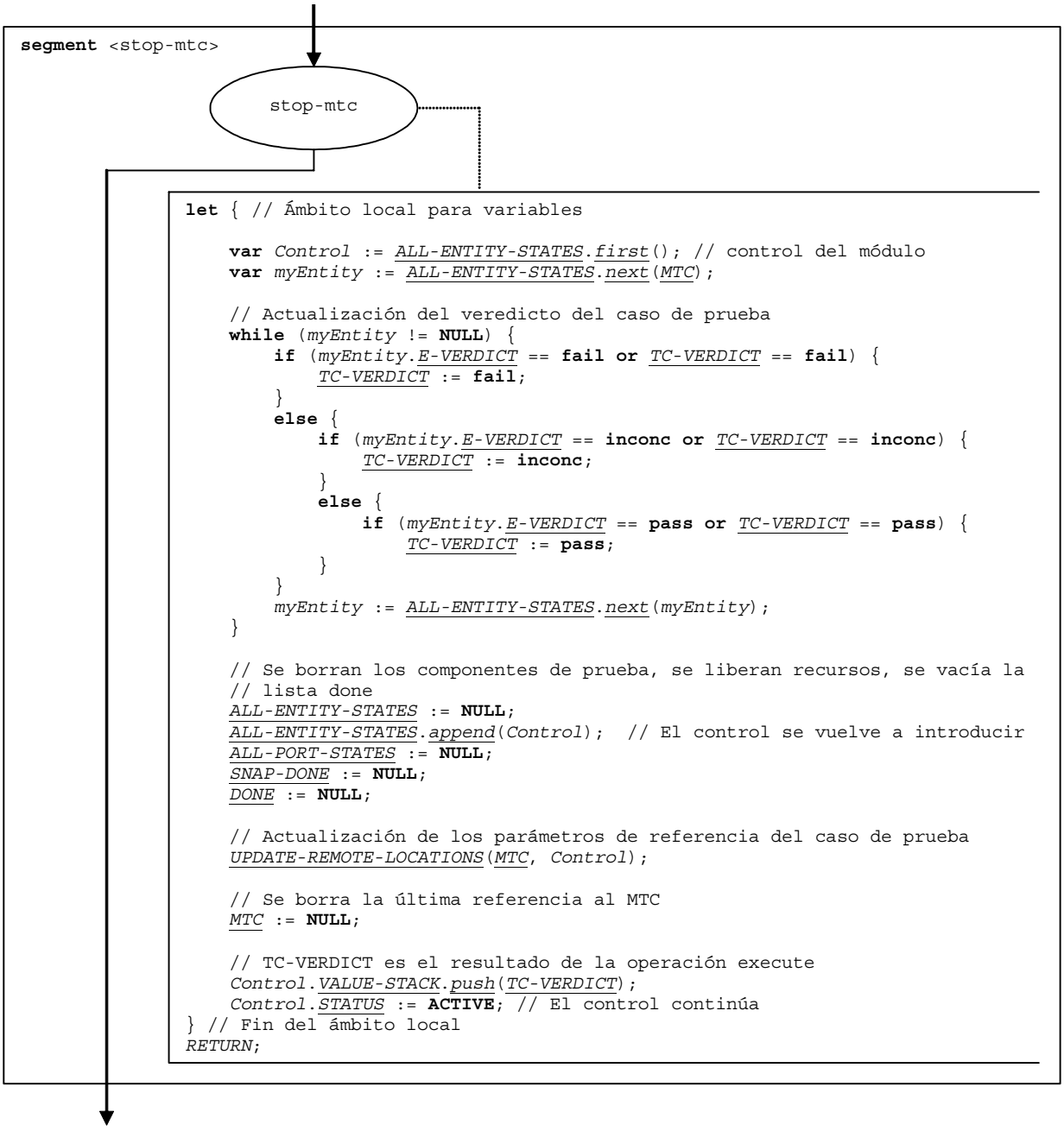


Figura 126/Z.143 – Segmento de diagrama de flujo <stop-mtc-op>

9.49.2 Segmento de diagrama de flujo <stop-component>

El segmento de diagrama de flujo `<stop-component>` de la figura 127 describe la detención de un componente de prueba paralelo. Como resultado, se actualizan el veredicto de caso de prueba *TC-VERDICT* y la lista de componentes de prueba finalizados (*DONE*) y se borra el componente del estado del módulo. El diagrama de flujo `<stop-component>` supone que el identificador del componente que ha de detenerse se encuentra en la cima de la pila de valores del componente que ejecuta el segmento.

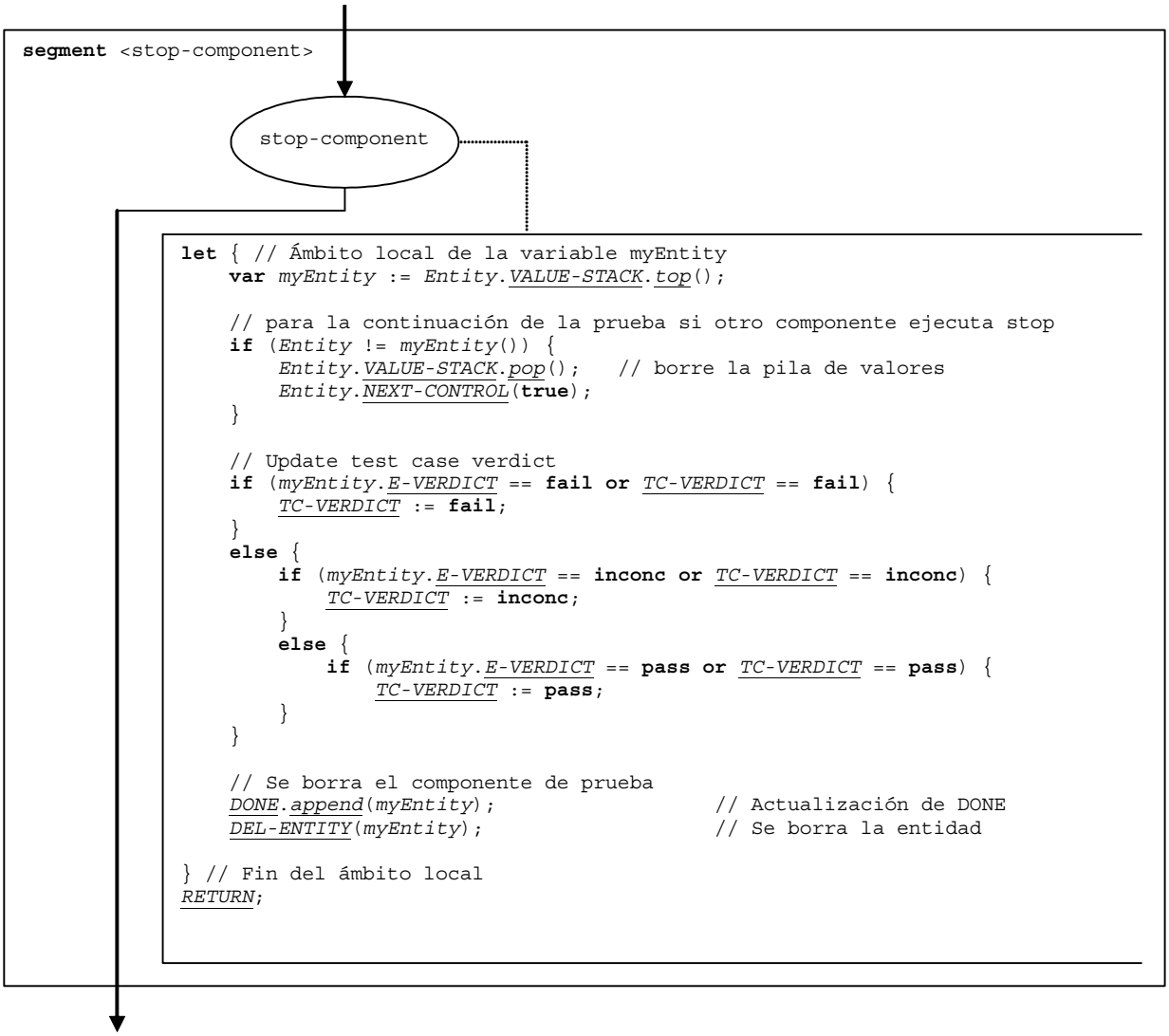


Figura 127/Z.143 – Segmento de diagrama de flujo <stop-component>

9.49.3 Segmento de diagrama de flujo <stop-all-comp>

El segmento de diagrama de flujo <stop-all-comp> de la figura 128 describe la detención de todos los componentes de prueba paralelos de un caso de prueba.

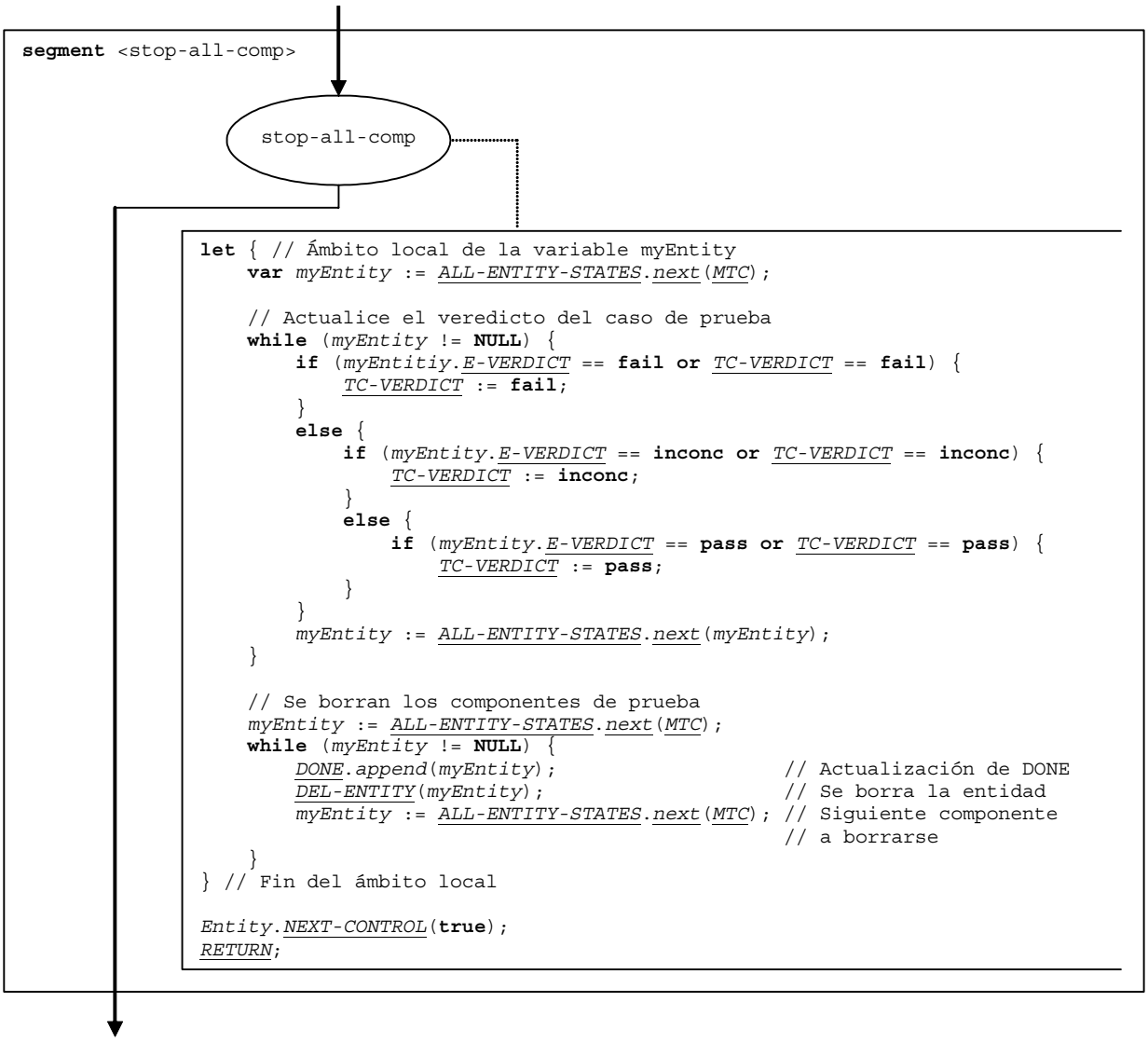


Figura 128/Z.143 – Segmento de diagrama de flujo <stop-all-comp>

9.50 Instrucción de ejecución stop

La siguiente es la estructura sintáctica de la instrucción de ejecución **stop**:

```
stop
```

El efecto de la instrucción de ejecución **stop** depende de la entidad que ejecuta la instrucción de ejecución **stop**:

a) Si es el control del módulo quien ejecuta la instrucción **stop**, finaliza la campaña de pruebas, es decir, todos los componentes de prueba y el control del módulo desaparecen del estado del módulo.

b) Si es el MTC quien ejecuta la instrucción **stop**, todos los componentes de prueba paralelos y el MTC finalizan su ejecución. El veredicto general del caso de prueba se actualiza y se ingresa en la pila de valores del control del módulo. Finalmente, el control se devuelve al control del módulo y finaliza el MTC.

c) Si es uno de los componentes de prueba quien ejecuta la instrucción **stop**, se actualizan el veredicto general del caso de prueba, *TC-VERDICT*, y la lista global *DONE*. Luego, el componente desaparece por completo del módulo.

El segmento de diagrama de flujo <stop-exec-stmt> de la figura 129 describe la ejecución de la instrucción **stop**.

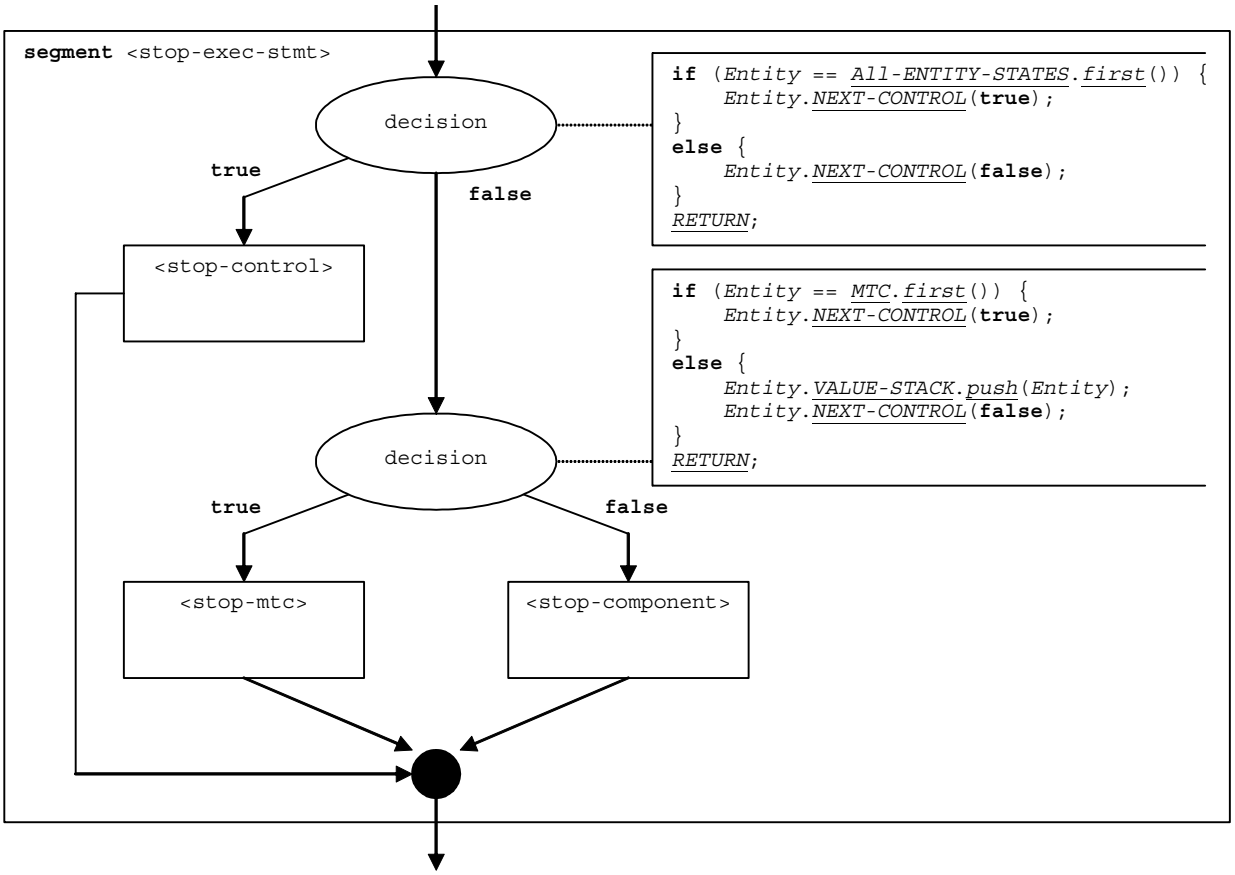


Figura 129/Z.143 – Segmento de diagrama de flujo <stop-exec-stmt>

9.50.1 Segmento de diagrama de flujo <stop-control>

El segmento de diagrama de flujo <stop-control> de la figura 130 describe la detención del control del módulo. Como resultado, *ALL-ENTITY-STATES* se fija a **NULL**, es decir, se cumple con la condición de finalización del procedimiento de evaluación del módulo (véase 8.6).

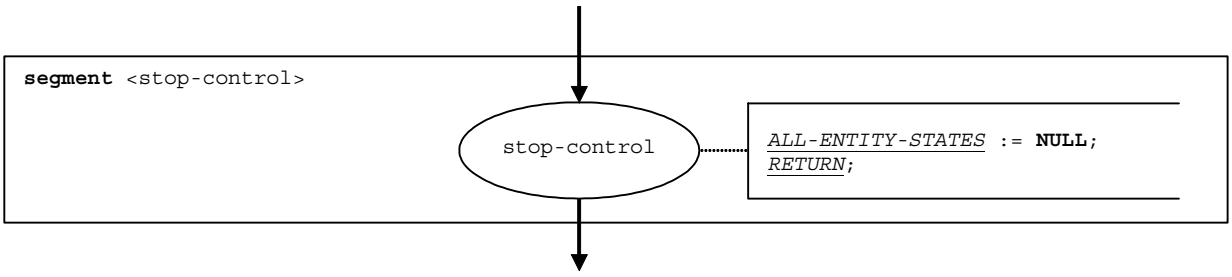


Figura 130/Z.143 – Segmento de diagrama de flujo <stop-control >

9.51 Operación de puerto stop

La siguiente es la estructura sintáctica de las operaciones de puerto **`stop`**:

```
<portId>.stop
```

El segmento de diagrama de flujo <stop-port-op> de la figura 131 define la ejecución de las operaciones de puerto **`stop`**.

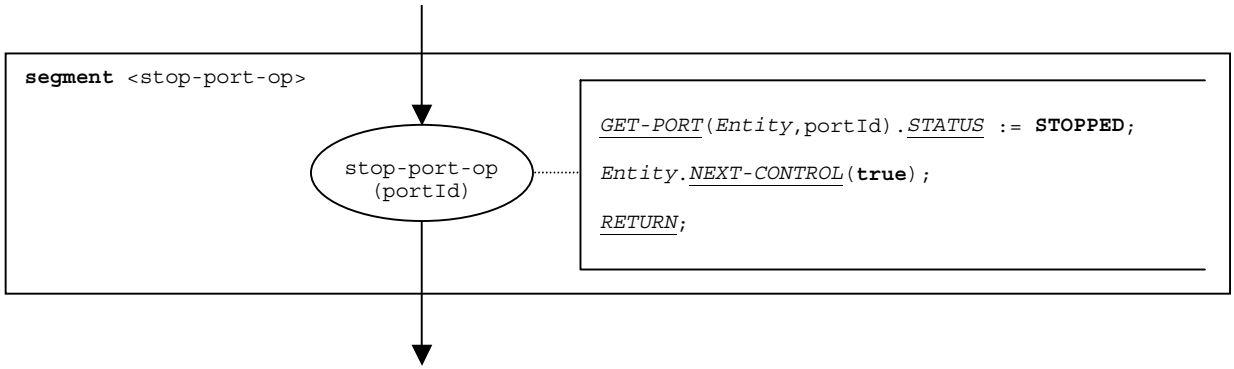


Figura 131/Z.143 – Segmento de diagrama de flujo <stop-port-op>

9.52 Operación de temporizador stop

La siguiente es la estructura sintáctica de las operaciones de temporizador **`stop`**:

```
<timerId>.stop
```

El segmento de diagrama de flujo <stop-timer-op> de la figura 132 define la ejecución de las operaciones de temporizador **`stop`**.

La palabra clave **`all`** se trata como un valor especial de `timerId`.

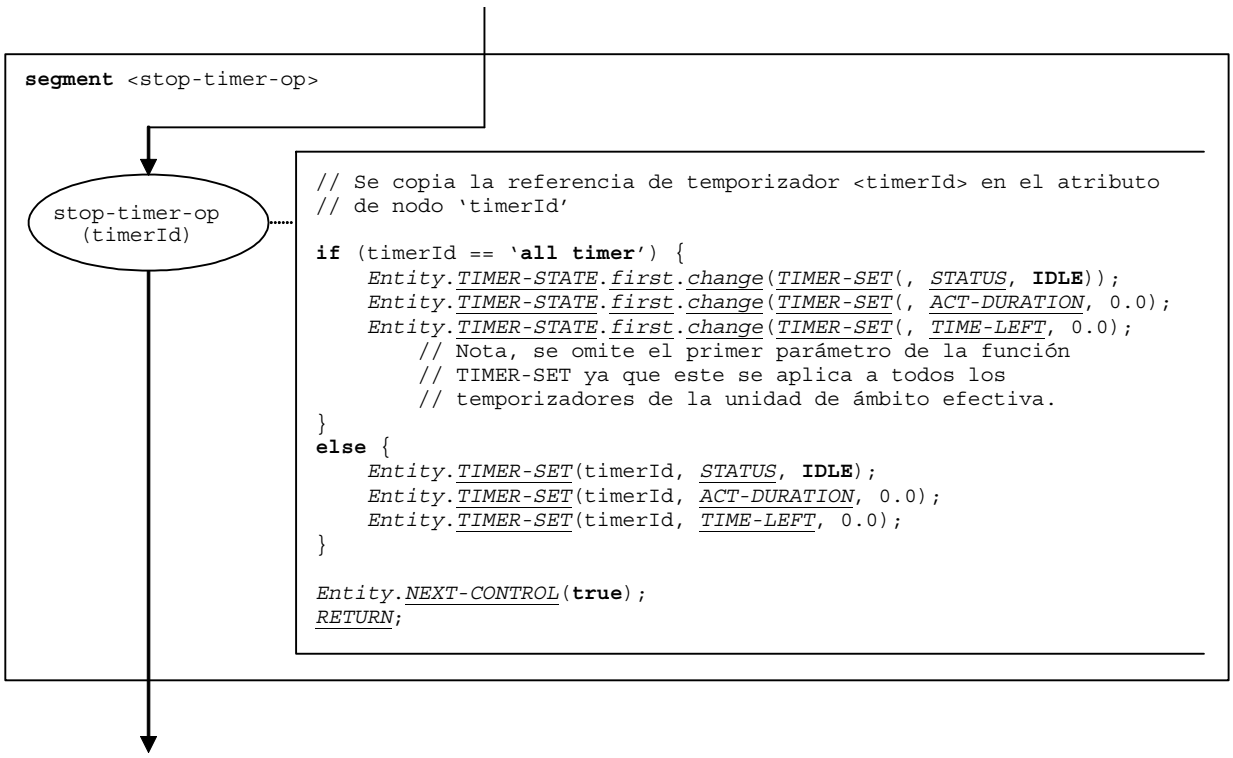


Figura 132/Z.143 – Segmento de diagrama de flujo <stop-timer-op>

9.53 Operación system

La siguiente es la estructura sintáctica de las operaciones **system**:

```
system
```

El segmento de diagrama de flujo <system-op> de la figura 133 define la ejecución de las operaciones **system**.

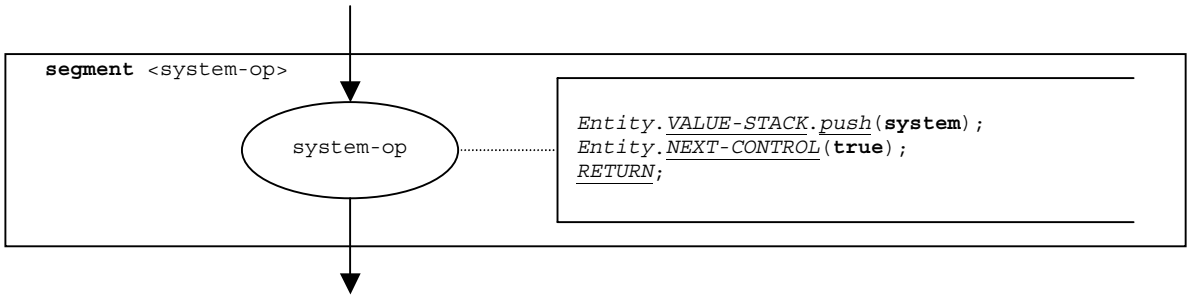


Figura 133/Z.143 – Segmento de diagrama de flujo <system-op>

9.54 Declaración timer

La siguiente es la estructura sintáctica de la declaración **timer**:

```
timer <timerId> [:= <float-expression>]
```

Como resultado de la declaración timer, se crea una nueva vinculación de temporizador. Es opcional declarar una duración por defecto. Se considera que el valor por defecto es una expresión que da como resultado un valor del tipo **float**.

El segmento de diagrama de flujo <timer-declaration> de la figura 134 define la ejecución de las declaraciones timer.

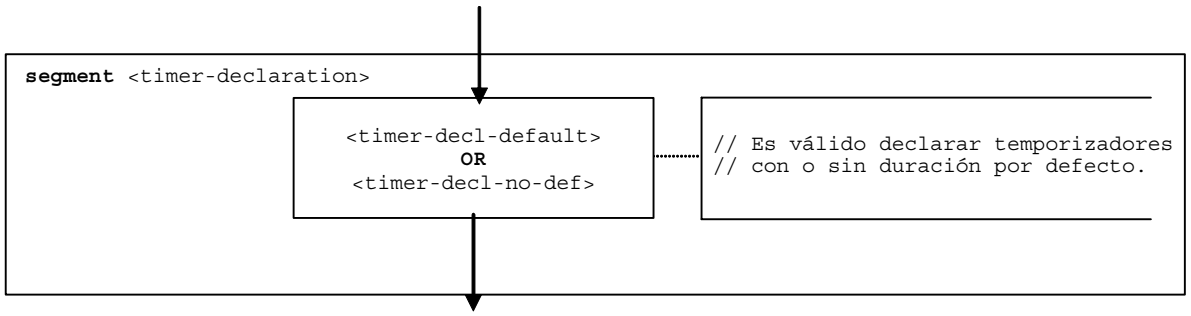


Figura 134/Z.143 – Segmento de diagrama de flujo <timer-declaration>

9.54.1 Segmento de diagrama de flujo <timer-decl-default>

El segmento de diagrama de flujo <timer-decl-default> de la figura 135 define la ejecución de las declaraciones timer en que se proporciona una duración por defecto en la forma de una expresión.

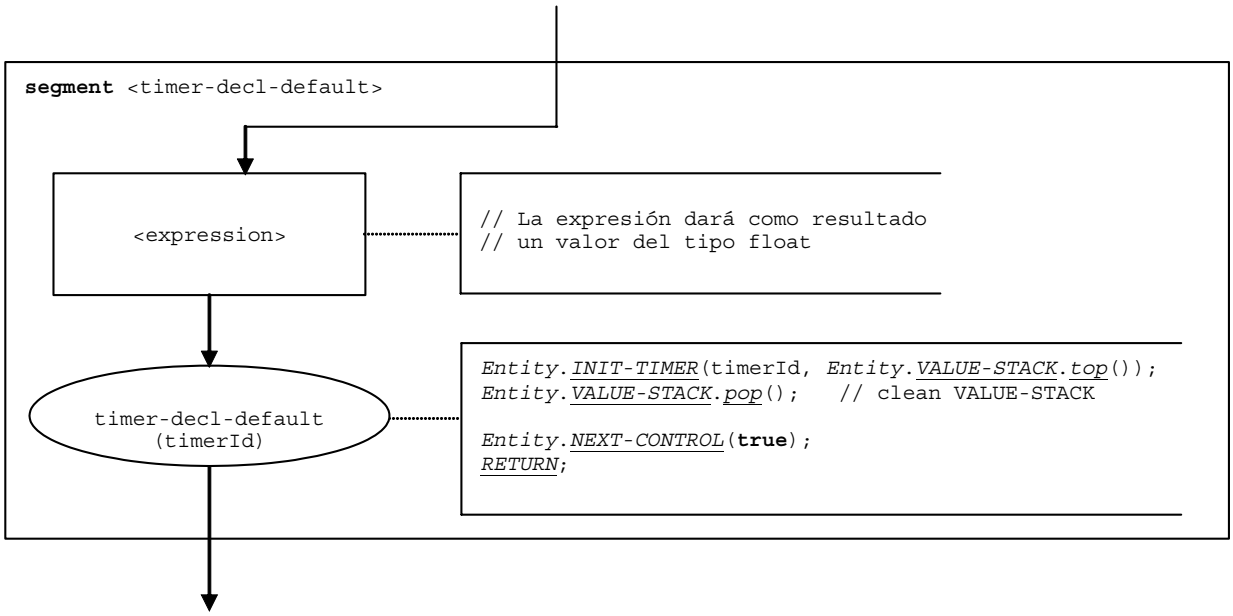


Figura 135/Z.143 – Segmento de diagrama de flujo <timer-decl-default>

9.54.2 Segmento de diagrama de flujo <timer-decl-no-def>

El segmento de diagrama de flujo <timer-decl-no-def> de la figura 136 define la ejecución de las declaraciones timer en que no se proporciona una duración por defecto, es decir, no se está definida la duración por defecto del temporizador.

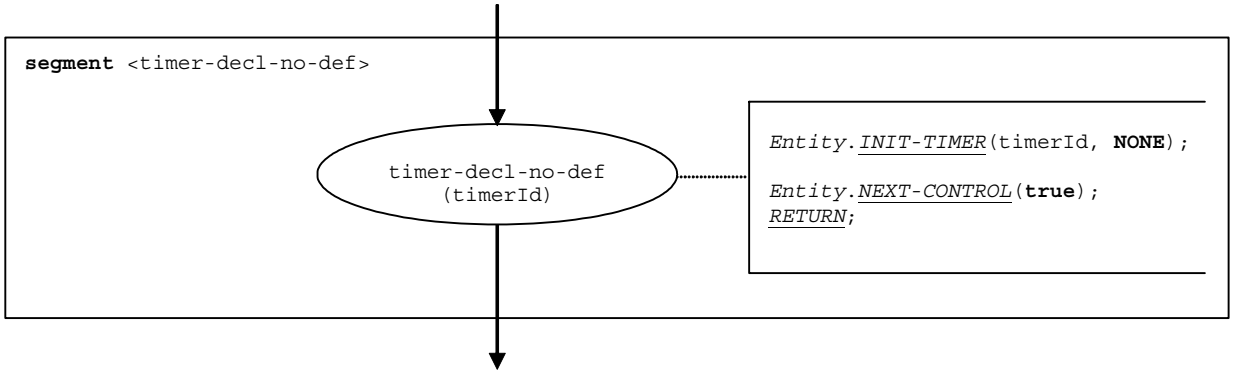


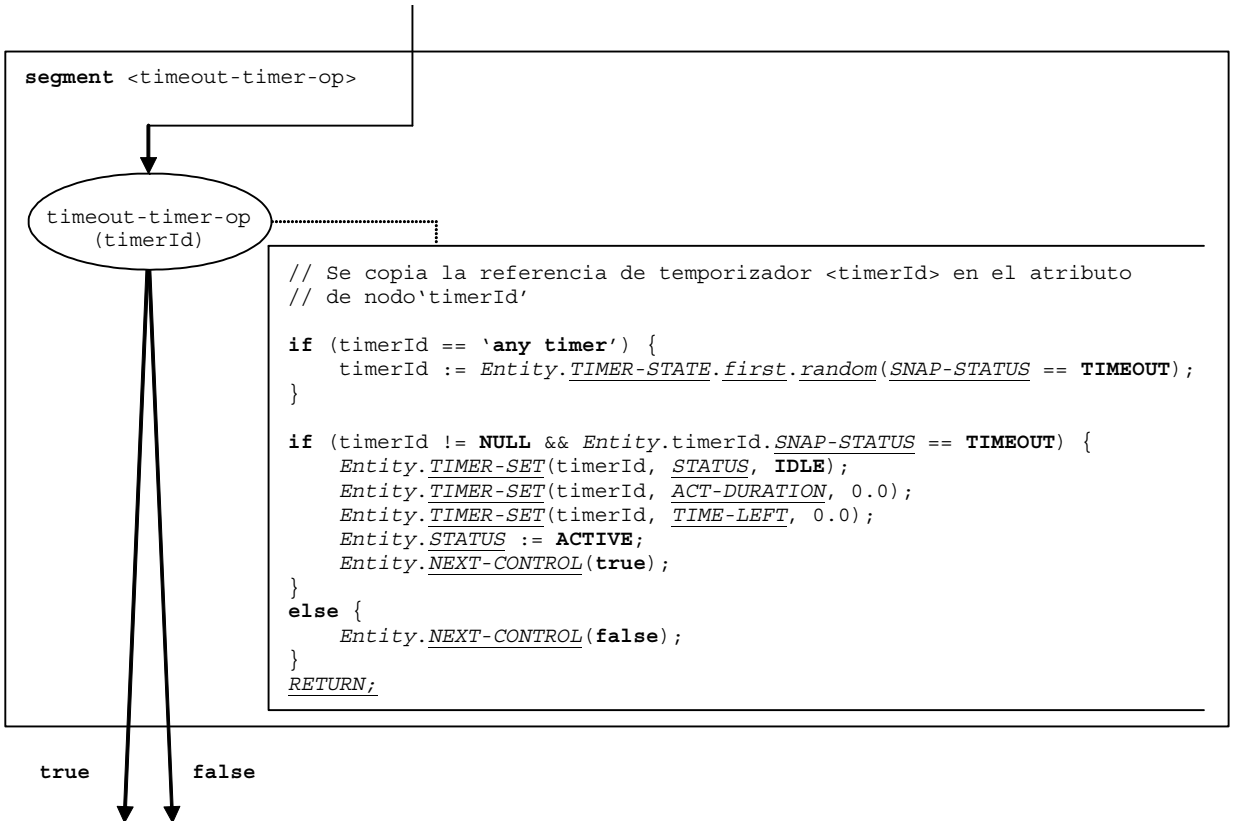
Figura 136/Z.143 – Segmento de diagrama de flujo <timer-decl-no-def>

9.55 Operación de temporizador timeout

La siguiente es la estructura sintáctica de las operaciones de temporizador **timeout**:

```
<timerId>.timeout
```

El segmento de diagrama de flujo <timeout-timer-op> de la figura 137 define la ejecución de las operaciones de temporizador **timeout**.



NOTA 1 – Se incluyen las operaciones **timeout** en instrucciones **alt**. Su evaluación se basa en el estado puntual efectivo, es decir, la decisión depende del valor de *SNAP-STATUS* de la vinculación de temporizador. Si la operación timeout es exitosa, es decir, *SNAP-STATUS* == TIMEOUT, se fija el temporizador al estado IDLE y el estado del componente cambia de SNAPSHOT a ACTIVE.

NOTA 2 – Cuando el resultado de **timeout** es **true** o **false**, o bien continúa la ejecución con la instrucción posterior a la operación **timeout** (rama **true**), o bien se debe verificar la siguiente alternativa de la instrucción **alt** (rama **false**).

NOTA 3 – Se trata la palabra clave any como un valor especial de timerId.

Figura 137/Z.143 – Segmento de diagrama de flujo <timeout-timer-op>

9.56 Operación unmap

La siguiente es la estructura sintáctica de las operaciones **unmap**:

```
unmap(<component expression>:<portId1>,system:<portId2>)
```

Se considera que los identificadores <portId1> y <portId2> son identificadores de puerto del componente de prueba y de la interfaz del sistema de pruebas correspondientes. Los componentes a que pertenece <portId1> se referencian por medio de la referencia de componente <component-expression>. La referencia puede almacenarse en variables o puede devolverse en una función, es decir, se trata de una expresión que da como resultado una referencia de componente. Se utiliza la pila de valores para almacenar la referencia de componente.

NOTA – Para la operación **unmap** es irrelevante si la instrucción **system**:<portId> aparece como primero o como segundo parámetro. En aras de simplicidad, se supone que siempre es el segundo parámetro.

En el segmento de diagrama de flujo <unmap-op> de la figura 138 se define la ejecución de la operación **unmap**.

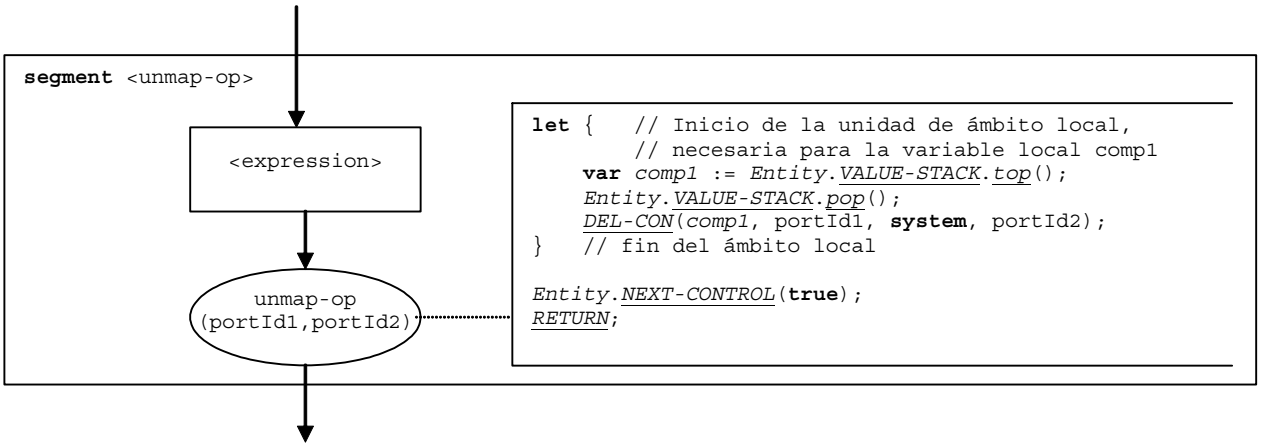


Figura 138/Z.143 – Segmento de diagrama de flujo <unmap-op>

9.57 Declaración de variable

La siguiente es la estructura sintáctica de las declaraciones de variables:

```
var <varType> <varId> [:= <varType-expression>]
```

Es opcional inicializar las variables proporcionando un valor inicial (en la forma de una expresión). Se considera que el valor inicial es una expresión que da como resultado un valor del mismo tipo que la variable.

El segmento de diagrama de flujo <variable-declaration> de la figura 139 define la ejecución de las declaraciones de variables.

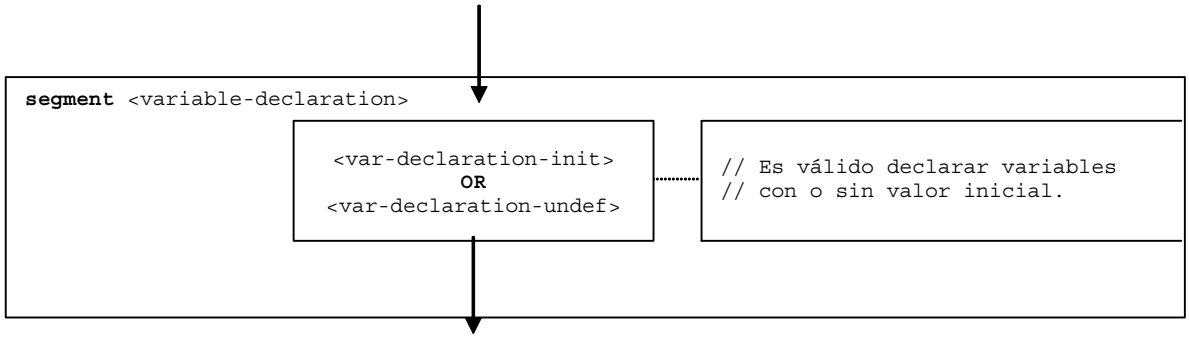


Figura 139/Z.143 – Segmento de diagrama de flujo <variable-declaration>

9.57.1 Segmento de diagrama de flujo <var-declaration-init>

El segmento de diagrama de flujo <var-declaration-init> de la figura 140 define la ejecución de declaraciones de variables en las que se proporciona un valor inicial en la forma de una expresión.

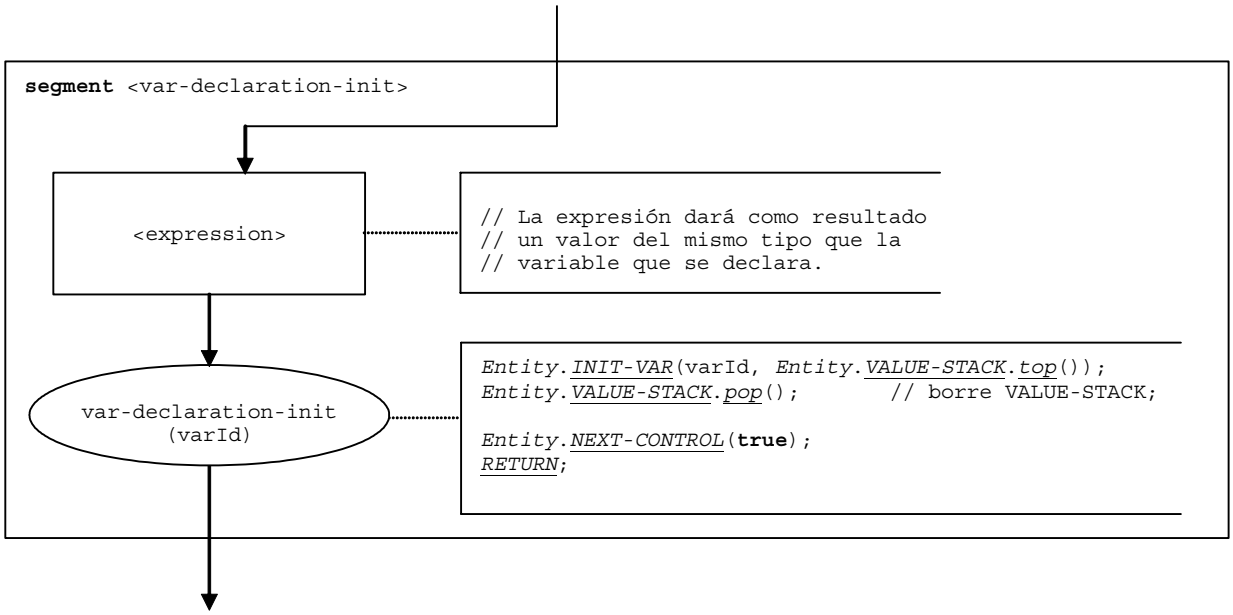


Figura 140/Z.143 – Segmento de diagrama de flujo <var-declaration-init>

9.57.2 Segmento de diagrama de flujo <var-declaration-undef>

El segmento de diagrama de flujo <var-declaration-undef> de la figura 141 define la ejecución de las declaraciones de variables en las que no se proporciona un valor inicial, es decir, el valor de la variable no está definido.

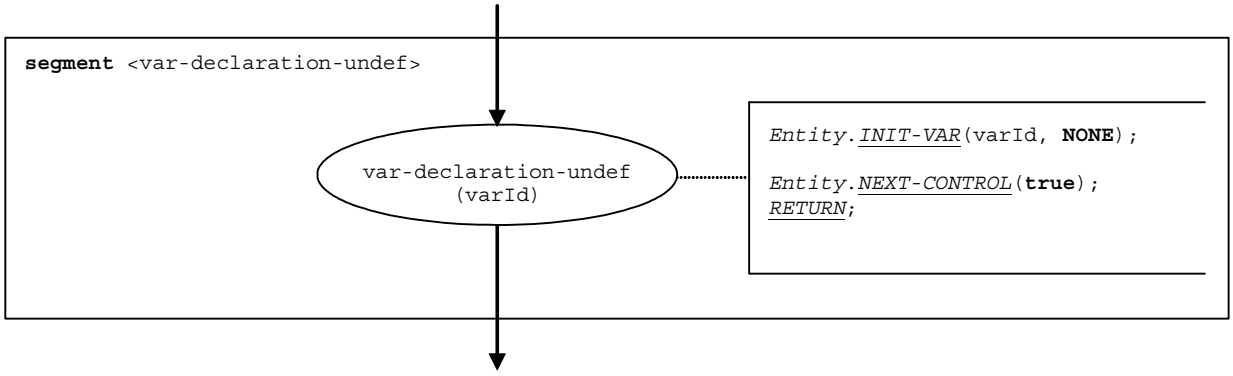


Figura 141/Z.143 – Segmento de diagrama de flujo <var-declaration-undef>

9.58 Instrucción while

La siguiente es la estructura sintáctica de las instrucciones **`while`**:

```
while (<boolean-expression>)  <statement-block>
```

El segmento de diagrama de flujo `<while-stmt>` de la figura 142 define la ejecución de las instrucciones **`while`**.

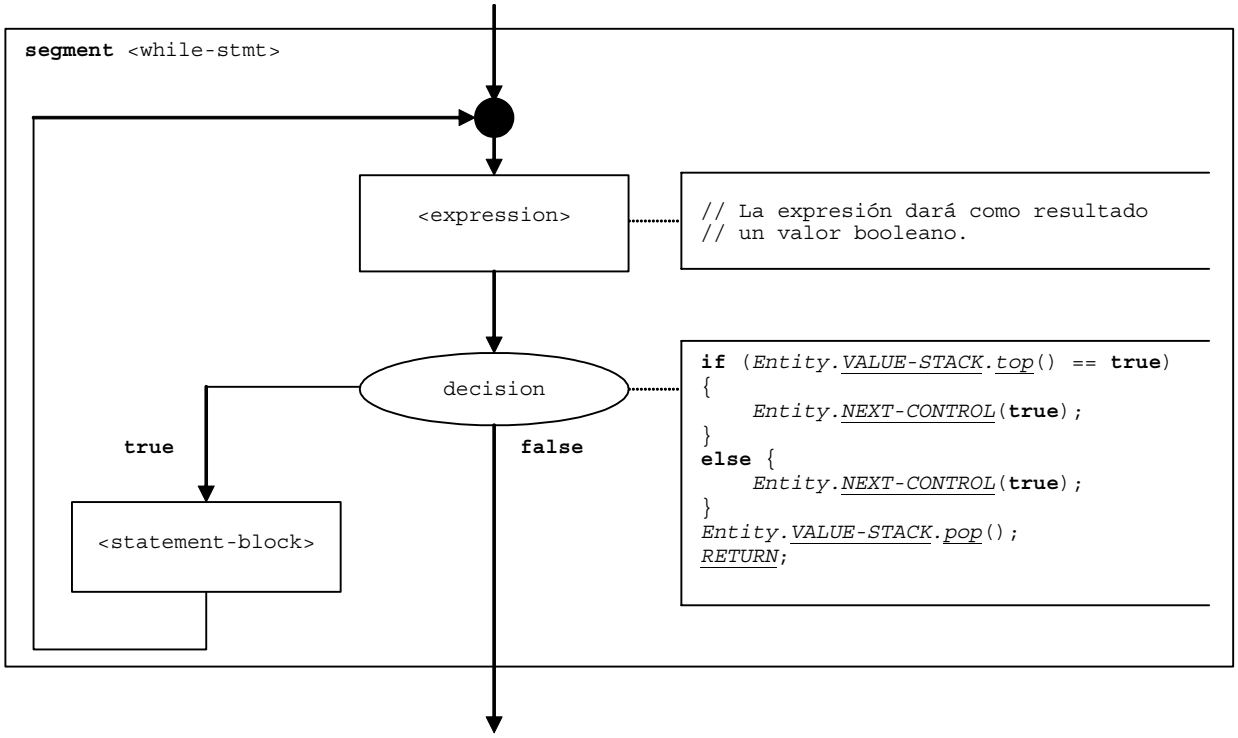


Figura 142/Z.143 – Segmento de diagrama de flujo <while-stmt>

10 Listas de componentes de la semántica operacional

10.1 Funciones y estados

Nombre	Descripción	Cláusula
ACT-DURATION	Duración con la que se ha iniciado un temporizador activo	8.3.2.4
add	Operación de listas: añade un elemento como primer elemento de la lista	8.3.1.1
ADD-CON	Añade una conexión a un estado de puerto	8.3.3.2
ALL-ENTITY-STATES	Estados de los componentes en el estado del módulo	8.3.1
ALL-PORT-STATES	Estado de los puertos en el estado del módulo	8.3.1
append	Operación de listas: añade un elemento como último elemento de la lista	8.3.1.1
APPLY-OPERATOR	Aplicación de operadores como +, – o /	8.6.2
change	Operación de listas: modifica todos los elementos de una lista	8.3.1.1
clear	Operación de pilas 'clear': borra una pila	8.3.2.1
clear	Operación de colas 'clear': suprime todos los elementos de una cola	8.3.3.2
clear-until	Operación de pila 'clear-until': extrae elementos hasta que un elemento en particular quede en la cima de la pila	8.3.2.1
CONNECTIONS-LIST	Lista de conexiones de un puerto	8.3.3
CONSTRUCT-ITEM	Construye un elemento que ha de enviarse	8.4.4
CONTINUE-COMPONENT	Continúa la ejecución del componente efectivo	8.6.2

Nombre	Descripción	Cláusula
CONTROL-STACK	Pila de nodos del diagrama de flujo que indica el estado de control efectivo de una entidad	8.3.2
DATA-STATE	Estado de datos en un estado de entidad	8.3.2
DEF-DURATION	Duración por defecto de un temporizador	8.3.2.4
DEFAULT-LIST	Lista de opciones por defecto de un estado de entidad	8.3.2
DEFAULT-POINTER	Apunta a la opción por defecto efectiva durante la evaluación por defecto	8.3.2
DEL-CON	Suprime una conexión de un estado de puerto	8.3.3.2
DEL-ENTITY	Suprime una entidad del estado de módulo	8.3.4
DEL-TIMER-SCOPE	Suprime un ámbito de temporizador	8.3.2.5
DEL-VAR-SCOPE	Suprime un ámbito de variable	8.3.2.3
delete	Operación de listas: borra un elemento de una lista	8.3.1.1
dequeue	Operación de colas: borra el primer elemento de una cola	8.3.3.2
DONE	Identificadores de componentes de prueba que se han detenido (parte del estado del módulo)	8.3.1
E-VERDICT	Veredicto local de la prueba de un componente de pruebas	8.3.2
enqueue	Operación de colas: incluye un elemento como ultimo elemento de una cola	8.3.3.2
first	Operación de colas 'first': devuelve el primer elemento de una cola	8.3.3.2
first	Operación de listas: devuelve el primer elemento de un lista	8.3.1.1
GET-FLOW-GRAPH	Recupera el nodo inicial de un diagrama de flujo	8.2.7
GET-PORT	Recupera una referencia de puerto	8.3.3.2
GET-REMOTE-PORT	Recupera la referencia de un puerto distante	8.3.3.2
GET-TIMER-LOC	Recupera la ubicación de un temporizador	8.3.2.5
GET-UNIQUE-ID	Cuando se le solicita, devuelve un identificador único	8.6.2
GET-VAR-LOC	Recupera la ubicación de una variable	8.3.2.3
INIT-CALL-RECORD	Inicializa variables de parámetros en comunicaciones basadas en procedimientos en la unidad de ámbito efectiva del componente de prueba	8.5.1
INIT-FLOW-GRAPHS	Inicializa el tratamiento de diagramas de flujo	8.6.2
INIT-TIMER	Crea una nueva vinculación de temporizador	8.3.2.5
INIT-TIMER-LOC	Crea una nueva vinculación de temporizador con una ubicación definida	8.3.2.5
INIT-TIMER-SCOPE	Inicializa un nuevo ámbito de temporizador	8.3.2.5
INIT-VAR	Crea una nueva vinculación de variable	8.3.2.3
INIT-VAR-LOC	Crea una nueva vinculación de variable con una ubicación definida	8.3.2.3
INIT-VAR-SCOPE	Inicializa un nuevo ámbito de variable	8.3.2.3
length	Operación de listas: devuelve la longitud de una lista	8.3.1.1
M-CONTROL	Identificador del control del módulo en el estado del módulo	8.3.1
MATCH-ITEM	Verifica si un mensaje, solicitud, respuesta o excepción corresponde con una operación de recepción	8.4.5
member	Operación sobre listas: verifica si cierto elemento forma parte de una lista	8.3.1.1
MTC	Referencia a MTC en el estado del módulo	8.3.1
NEW-CALL-RECORD	Crea un registro de solicitud para una solicitud de función	8.5.1
NEW-ENTITY	Crea un nuevo estado de entidad	8.3.2.1
NEW-PORT	Crea un nuevo puerto	8.3.3.2
NEXT	Recupera el nodo sucesor de un nodo en particular en un diagrama de flujo	8.1.6
next	Operación sobre listas: devuelve el siguiente elemento de una lista	8.3.1.1
NEXT-CONTROL	Extrae el nodo de diagrama de flujo de la cima de la pila de control e inserta el siguiente nodo de diagrama de flujo en la pila de control	8.3.2.1
OWNER	Dueño de un puerto	8.3.3
pop	Operación de pilas 'pop': extrae un elemento de una pila	8.3.2.1

Nombre	Descripción	Cláusula
PORT-NAME	El nombre de un puerto	8.3.3
push	Operación de pilas 'push': pone un elemento en una pila	8.3.2.1
random	Operación de listas: devuelve un elemento aleatorio de una lista	8.3.1.1
REMOTE-ENTITY	Entidad distante de una conexión en el estado de puerto	8.3.3.1
REMOTE-PORT-NAME	Nombre de un puerto de una conexión en el estado de puerto	8.3.3.1
RETRIEVE-INFO	Recupera información de un mensaje, solicitud, respuesta o excepción recibida	8.4.6
RETURN	Devuelve el control al procedimiento de evaluación del módulo	8.6.2
SNAP-ACTIVE	Número de componentes de prueba activos en el momento en que el MTC determina el estado puntual (forma parte del estado del módulo)	8.3.1
SNAP-DONE	Lista de componentes de los prueba que se han detenido en el momento en que se determina un estado puntual	8.3.2
SNAP-PORTS	Proporciona la funcionalidad para determinar el estado puntual, es decir, actualiza SNAP-VALUE	8.3.3.2
SNAP-STATUS	Estado puntual de un temporizador	8.3.2.4
SNAP-TIMER	Proporciona la funcionalidad para determinar el estado puntual y actualiza SNAP-VALUE y SNAP-STATUS	8.3.2.5
SNAP-VALUE	Valor del estado puntual de un temporizador	8.3.2.4
SNAP-VALUE	Para la semántica de estado puntual, se actualiza cuando se determina el estado puntual	8.3.3
STATUS	Estado (**ACTIVE**, **SNAPSHOT**, **REPEAT** o **BLOCKED**) del control del módulo de un componente de prueba	8.3.2
STATUS	Estado (**IDLE**, **RUNNING** o **TIMEOUT**) de un temporizador	8.3.2.4
STATUS	Estado (**STARTED** o **STOPPED**) de un puerto	8.3.3
TC-VERDICT	Veredicto del caso de prueba en el estado del módulo	8.3.1
TIME-LEFT	Tiempo de ejecución que le resta a un temporizador activo antes de expirar	8.3.2.4
TIMER-GUARD	Temporizador que controla las instrucciones **`execute`** y las operaciones **`call`**	8.3.2
TIMER-NAME	Nombre de un temporizador	8.3.2.4
TIMER-SET	Asignación de valores a un temporizador	8.3.2.5
TIMER-STATE	Estado de un temporizador en un estado de entidad	8.3.2
top	Operación de pila 'top': devuelve el elemento de la cima de una pila	8.3.2.1
UPDATE-REMOTE-REFERENCES	Actualiza con un mismo valor los temporizadores y variables con la misma ubicación en entidades diferentes	8.3.4
VALUE	Valor de una variable	8.3.2.2
VALUE-QUEUE	Cola de puerto	8.3.3
VALUE-STACK	Pila de valores utilizada para almacenar los resultados de expresiones, operandos, operaciones y funciones	8.3.2
VAR-NAME	Nombre de una variable	8.3.2.2
VAR-SET	Para fijar el valor de una variable	8.3.2.3
DYNAMIC-ERROR	Indica que ocurrió un error dinámico	8.6.2
<identifier>	Identificador único de un componente de pruebas	8.3.2
<location>	Soporta unidades de ámbito, parámetros de temporizador y por referencia. Representa una ubicación para el almacenamiento de temporizadores y variables	8.3.2.2, 8.3.2.4

10.2 Palabras clave especiales

Palabra clave	Descripción	Cláusula
ACTIVE	*STATUS* de un estado de entidad	8.3.2
BLOCKED	*STATUS* de un estado de entidad	8.3.2
IDLE	*STATUS* de un estado de temporizador	8.3.2.4
MARK	Utilizado como marca para la pila *VALUE-STACK*	8.3.2
NONE	Utilizado para describir un valor indefinido	8.3.2.3, 8.3.2.5, 8.3.3.2
NULL	Valor simbólico de los tipos apuntador o similares a apuntador. Se utiliza para indicar que no se está apuntando a nada.	8.3.1.1, 8.3.2.1, 8.3.3, 8.3.3.2, 8.6.1.1
REPEAT	*STATUS* de un estado de entidad	8.3.2
RUNNING	*STATUS* de un estado de temporizador	8.3.2.4
SNAPSHOT	*STATUS* de un estado de entidad	8.3.2
STARTED	*STATUS* de un puerto	8.3.3
STOPPED	*STATUS* de un puerto	8.3.3
TIMEOUT	*STATUS* de un estado de temporizador	8.3.2.4

10.3 Diagramas de flujo de descripciones de comportamiento de TTCN-3

	Referencia	
	Figura	Cláusula
Control del módulo	18	8.2.2
Casos de prueba	19	8.2.3
Funciones	20	8.2.4
Altstep	21	8.2.5
Definiciones de tipo de componente	22	8.2.6

10.4 Segmentos de diagrama de flujo

Identificador	Construcción asociada de TTCN-3	Referencia	
		Figura	Cláusula
<action-stmt>	Instrucción **`action`**	36	9.1
<activate-stmt>	Instrucción **`activate`**	37	9.2
<alt-stmt>	Instrucción **`alt`**	38	9.3
<altstep-call>	Invocación de un altstep	44	9.4
<altstep-call-branch>	Instrucción **`alt`**	41	9.3.3
<assignment-stmt>	Asignación	45	9.5
<b-call-with-duration>	Operación **`call`**	52	9.6.4
<b-call-without-duration>	Operación **`call`**	51	9.6.3
<blocking-call-op>	Operación **`call`**	47	9.6
<call-op>	Operación **`call`**	46	9.6
<call-reception-part>	Operación **`call`**	53	9.6.5
<catch-op>	Operación **`catch`**	55	9.7
<catch-timeout-exception>	Operación **`call`**	54	9.6.6
<check-op>	Operación **`check`**	56	9.8
<check-with-sender>	Operación **`check`**	57	9.8.1
<check-without-sender>	Operación **`check`**	58	9.8.2
<clear-port-op>	Operación **`clear`** port	59	9.9
<connect-op>	Operación **`connect`**	60	9.10

Identificador	Construcción asociada de TTCN-3	Referencia	
		Figura	Cláusula
<constant-definition>	Definición de **constantes**	61	9.11
<create-op>	Operación **create**	62	9.12
<deactivate-stmt>	Instrucción **deactivate**	63	9.13
<default-evocation>	Instrucción **alt**	43	9.3.5
<disconnect-op>	Operación **disconnect**	64	9.14
<do-while-stmt>	Instrucción **do-while**	65	9.15
<done-component-op>	Operación de componente **done**	66	9.16
<else-branch>	Instrucción **alt**	42	9.3.4
<execute-stmt>	Instrucción **execute**	67	9.17
<execute-timeout>	Instrucción **execute**	69	9.17.2
<execute-without-timeout>	Instrucción **execute**	68	9.17.1
<expression>	Expresión	70	9.18
<finalize-component-init>	Se utiliza en definiciones de tipo de componente	75	9.19
<for-stmt>	Instrucción **for**	79	9.23
<func-op-call>	Expresión	73	9.18.3
<function-call>	Solicitud de una función	80	9.24
<getcall-op>	Operación **getcall**	86	9.25
<getreply-op>	Operación **getreply**	87	9.26
<getverdict-op>	Operación **getverdict**	88	9.27
<goto-stmt>	Instrucción **goto**	89	9.28
<if-else-stmt>	Instrucción **if-else**	90	9.29
<init-component-scope>	Se utiliza en definiciones de tipo de componente	76	9.20
<label-stmt>	Instrucción **label**	91	9.30
<lit-value>	expresión	71	9.18.1
<log-stmt>	Instrucción **log**	92	9.31
<map-op>	Operación **map**	93	9.32
<mtc-op>	Operación **mtc**	94	9.33
<nb-call-without-receiver>	Operación **call**	50	9.6.2
<nb-call-with-receiver>	Operación **call**	49	9.6.1
<non-blocking-call-op>	Operación **call**	48	9.6
<operator-appl>	Expresión	74	9.18.4
<parameter-handling>	Manejo de parámetros de funciones, altstep y casos de prueba	77	9.21
<port-declaration>	Declaración de puertos	95	9.34
<predef-ext-func-call>	Solicitud de un función (solicitud de un función predefinida o externa)	85	9.24.5
<raise-op>	Operación **raise**	96	9.35
<raise-with-receiver-op>	Operación **raise**	97	9.35.1
<raise-without-receiver-op>	Operación **raise**	98	9.35.2
<read-timer-op>	Operación de temporizador **read**	99	9.36
<receive-assignment>	Operación **receive**	103	9.37.3
<receive-op>	Operación **receive**	100	9.37
<receive-with-sender>	Operación **receive**	101	9.37.1
<receive-without-sender>	Operación **receive**	102	9.37.2
<receiving-branch>	Instrucción **alt**	40	9.3.2
<ref-par-var-calc>	Solicitud de una función (tratamiento de parámetros por referencia)	82	9.24.2
<ref-par-timer-calc>	Solicitud de una función (tratamiento de parámetros de temporizador)	83	9.24.3

Identificador	Construcción asociada de TTCN-3	Referencia	
		Figura	Cláusula
<repeat-stmt>	Instrucción **repeat**	104	9.38
<reply-op>	Operación **reply**	105	9.39
<reply-with-receiver-op>	Operación **reply**	106	9.39.1
<reply-without-receiver-op>	Operación **reply**	107	9.39.2
<return-stmt>	Instrucción **return**	108	9.40
<return-with-value>	Instrucción **return**	109	9.40.1
<return-without-value>	Instrucción **return**	110	9.40.2
<running-component-op>	Operación de componente **running**	111	9.41
<running-comp-act>	Operación de componente **running**	112	9.41.1
<running-comp-snap>	Operación de componente **running**	113	9.41.2
<running-timer-op>	Operación de temporizador **running**	114	9.42
<self-op>	Operación **self**	115	9.43
<send-op>	Operación **send**	116	9.44
<send-with-receiver-op>	Operación **send**	117	9.44.1
<send-without-receiver-op>	Operación **send**	118	9.44.2
<setverdict-op>	Operación **setverdict**	119	9.45
<start-component-op>	Operación de componente **start**	120	9.46
<start-port-op>	Operación de puerto **start**	121	9.47
<start-timer-op>	Operación de temporizador **start**	122	9.48
<start-timer-op-default>	Operación de temporizador **start**	123	9.48.1
<start-timer-op-duration>	Operación de temporizador **start**	124	9.48.2
<statement-block>	Bloque de instrucciones en instrucciones compuestas	78	9.22
<stop-component-op>	Operación de componente **stop**	125	9.49
<stop-mtc>	Operación de componente **stop** (detener el MTC)	126	9.49.1
<stop-component>	Operación de componente **stop** (detener un solo componente de prueba)	127	9.49.2
<stop-all-comp>	Operación de componente **stop** (detener todos los componentes)	128	9.49.3
<stop-exec-stmt>	Instrucción de ejecución **stop**	129	9.50
<stop-control>	Instrucción de ejecución **stop** (detener el control del módulo)	130	9.50.1
<stop-port-op>	Operación de puerto **stop**	131	9.51
<stop-timer-op>	Operación de temporizador **stop**	132	9.52
<system-op>	Operación **system**	133	9.53
<take-snapshot>	Instrucción **alt**	39	9.3.1
<timeout-timer-op>	Operación **timeout**	137	9.55
<timer-declaration>	Declaración de temporizadores	134	9.54
<timer-decl-default>	Declaración de temporizadores	135	9.54.1
<timer-decl-no-def>	Declaración de temporizadores	136	9.54.2
<timeout-timer-op>	Operación **timeout**	137	9.55
<unmap-op>	Operación **unmap**	138	9.56
<user-def-func-call>	Solicitud de una función (solicitud de una función definida por el usuario)	84	9.24.4
<value-par-calculation>	Solicitud de una función (manejo de parámetros por valor)	81	9.24.1
<var-declaration-init>	Declaración de variables	140	9.57.1
<var-declaration-undef>	Declaración de variables	141	9.57.2
<var-value>	Expresión	72	9.18.2
<variable-declaration>	Declaración de variables	139	9.57
<while-stmt>	Instrucción **while**	140	9.58

SERIES DE RECOMENDACIONES DEL UIT-T

Serie A	Organización del trabajo del UIT-T
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 multimedia
Serie I	Red digital de servicios integrados
Serie J	Redes de cable y transmisión de programas radiofónicos y televisivos, y de otras señales multimedia
Serie K	Protección contra las interferencias
Serie L	Construcción, instalación y protección de los cables y otros elementos de planta exterior
Serie M	Gestión de las telecomunicaciones, incluida la RGT y el mantenimiento de redes
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, comunicaciones de sistemas abiertos y seguridad
Serie Y	Infraestructura mundial de la información, aspectos del protocolo Internet y Redes de la próxima generación
Serie Z	**Lenguajes y aspectos generales de soporte lógico para sistemas de telecomunicación**

Impreso en Suiza
Ginebra, 2007