Unión Internacional de Telecomunicaciones

UIT-T Z.142

SECTOR DE NORMALIZACIÓN
DE LAS TELECOMUNICACIONES
DE LA UIT

(02/2003)

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: Formato de presentación gráfica

Recomendación UIT-T Z.142

UIT-T

ITU Unión Internacional de Telecomunicaciones

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.142

Notación de pruebas y de control de pruebas, versión 3: Formato de presentación gráfica

Resumen

En la presente Recomendación se describe el formato de representación gráfica para la especificación y aplicación del lenguaje TTCN-3 (Notación de pruebas y de control de pruebas) conocido como GFT. El GFT se utiliza para representar gráficamente el comportamiento de pruebas en forma de diagramas secuenciales. Consta de una serie de símbolos que permiten representar gráficamente los casos de prueba, funciones, alternativas y partes de control TTCN-3. El GFT puede aplicarse cuando sea necesario definir o documentar gráficamente el comportamiento de una prueba.

Orígenes

La Recomendación UIT-T Z.142 fue aprobada el 13 de febrero de 2003 por la Comisión de Estudio 17 (2001-2004) 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.

ÍNDICE

Introducción

El formato de presentación gráfica de TTCN-3 (GFT) se basa en la Rec. UIT-T Z.120 [3] en la que se definen los gráficos de secuencias de mensajes (MSC). El GFT utiliza un subconjunto de MSC con ampliaciones específicas para pruebas. La mayor parte de las ampliaciones son únicamente de texto. Las ampliaciones gráficas se definen para facilitar la lectura de los diagramas GFT. En la medida de lo posible, el GFT se define de manera análoga al MSC, de modo que para representar gráficamente casos de prueba TTCN-3 en el formato GFT puedan utilizarse con ligeras modificaciones las herramientas MSC existentes.

El lenguaje núcleo de TTCN-3, definido en la Rec. UIT-T Z.140, proporciona una sintaxis textual, una semántica estática y una semántica operacional completas, así como una definición para utilizar el lenguaje con ASN.1. El formato de presentación GFT es una forma alternativa de visualizar el lenguaje núcleo (figura 1).

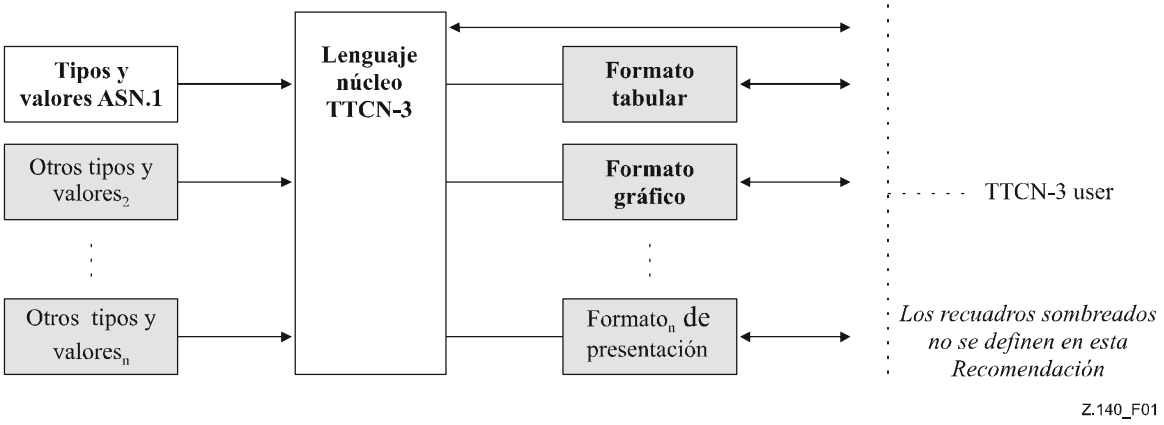


Figura 1/Z.142 – El lenguaje núcleo y los diversos formatos de presentación desde el punto de vista del usuario

El lenguaje núcleo puede utilizarse con independencia del GFT. Sin embargo, el GFT no puede utilizarse sin el lenguaje núcleo. La utilización e implementación del GFT se realizará sobre la base del lenguaje núcleo.

En esta Recomendación se definen:

- los conceptos del lenguaje del GFT;
- las directrices para la utilización del GFT;
- la gramática del GFT;
- la correspondencia, en los dos sentidos, con el lenguaje núcleo TTCN-3.

El conjunto de todas estas características constituye el GFT – formato de presentación gráfica de TTCN-3.

Recomendación UIT-T Z.142

Notación de pruebas y de control de pruebas, versión 3: Formato de presentación gráfica

1 Alcance

La presente Recomendación define el formato de presentación gráfica para el lenguaje núcleo TTCN-3, definido en la Rec. UIT-T Z.140. Este formato de presentación utiliza un subconjunto de los gráficos de secuencias de mensaje definidos en la Rec. UIT-T Z.120 [3] con ampliaciones específicas para pruebas.

La Recomendación se basa en el lenguaje núcleo TTCN-3 definido en la Rec. UIT-T Z.140. Es especialmente adecuado para visualizar pruebas en el formato GFT y no se limita a ningún tipo concreto de especificación de pruebas.

La especificación en otros formatos queda fuera del alcance de esta Recomendación.

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 (2003), *Notación de pruebas y de control de pruebas versión 3: Lenguaje núcleo*. Esta Recomendación también está disponible como norma ES 201 873-1 V2.2.1 (2003-02) de la ETSI.

[2] Recomendación UIT-T Z.141 (2003), *Notación de pruebas y de control de pruebas versión 3: Formato de presentación tabular*.

[3] Recomendación UIT-T Z.120 (1999), *Gráficos de secuencias de mensajes*.

[4] Recomendación UIT-T X.292 (2002), *Metodología y marco de las pruebas de conformidad para interconexión de sistemas abiertos de las Recomendaciones sobre los protocolos para aplicaciones del UIT-T – Notación combinada arborescente y tabular*.

3 Abreviaturas, siglas o acrónimos

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

BNF Forma Backus-Naur (*Backus-Naur Form*)

CATG Generación de pruebas asistidas por ordenador (*computer-aided test generation*)

ETSI Instituto Europeo de Normas de Telecomunicación (*European Telecommunication Standards Institute*)

GFT Formato de presentación gráfica de TTCN-3 (*graphical presentation format of TTCN-3*)

HMSC Gráfico de secuencias de mensajes de alto nivel (*high-level message sequence chart*)

MSC Gráfico de secuencias de mensajes (*message sequence chart*)

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

PTC Componente de prueba paralelo (*parallel test component*)

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

TFT Formato de presentación tabular TTCN-3 (*tabular presentation format of TTCN-3*)

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

4 Descripción general

Según la metodología de pruebas de conformidad OSI definidas en la Rec. UIT-T X.292 [4], las pruebas comienzan normalmente con la descripción del objetivo de las mismas, que se define del modo siguiente:

"Una descripción en palabras del objetivo concreto de la prueba, destinada a comprobar un requisito o un conjunto de requisitos de conformidad con la correspondiente especificación OSI."

Una vez definidos todos los objetivos de la prueba, se elabora un conjunto de pruebas abstractas que podrá consistir en uno o varios casos de prueba abstracta. Un caso de prueba abstracta define las acciones de los procesos de comprobación necesarios para validar una parte (o todos) los objetivos de la prueba.

Al aplicar estos términos a los gráficos de secuencias de mensajes (MSC) podemos definir dos tipos de categorías de utilización

1) *Utilización de los MSC para definir los objetivos de la prueba* – Normalmente, la especificación MSC elaborada como un caso de utilización o una parte de una especificación del sistema puede considerarse el objetivo de la prueba; es decir, describe un requisito del SUT en forma de una descripción de comportamiento que puede comprobarse. Por ejemplo, la figura 2 representa un MSC sencillo que describe la interacción entre ejemplares que representan el SUT y sus interfaces A, B y C. En una implementación real de dicho sistema las interfaces A, B y C pueden corresponder a puertos o puntos de acceso al servicio o puertos. El MSC de la figura 2 sólo describe la interacción con el SUT y no las acciones de los componentes de prueba necesarios para validar el comportamiento del SUT; es decir, es una descripción del objetivo de la prueba.

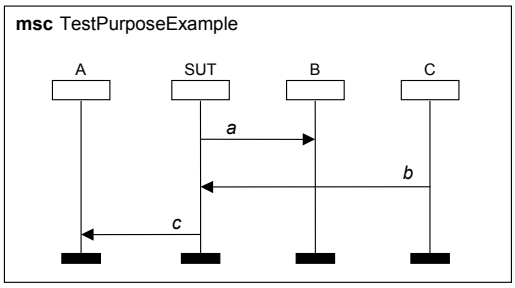


Figura 2/Z.142 – Descripción en MSC de la interacción del STU con sus interfaces

2) *Utilización del MSC para definir casos de pruebas abstractos* – Una especificación mediante MSC que describe un caso de prueba abstracto especifica el comportamiento de los componentes de prueba necesarios para validar un determinado objetivo de la prueba. En la figura 3 se muestra una descripción en MSC de un caso sencillo de prueba abstracto en el que un componente de prueba principal (MTC) intercambia los mensajes *a*, *b* y *c* con el SUT a través de los puertos PortA, PortB y PortC a fin de alcanzar los objetivos de prueba mostrados en la figura 2. Los mensajes *a* y *c* son enviados por el SUT a través de los

puertos A y B (figura 2) y recibidos por el MTC (figura 3) por esos mismos puertos. El mensaje *b* es enviado por el MTC y recibido por el SUT.

NOTA – Los ejemplos en las figuras 2 y 3 son ejemplos sencillos destinados únicamente a ilustrar las diferentes formas de utilizar el MSC para pruebas. Los diagramas serían más complicados para el caso de un SUT distribuido consistente en varios procesos o una configuración de prueba distribuida que constase de varios componentes de prueba.

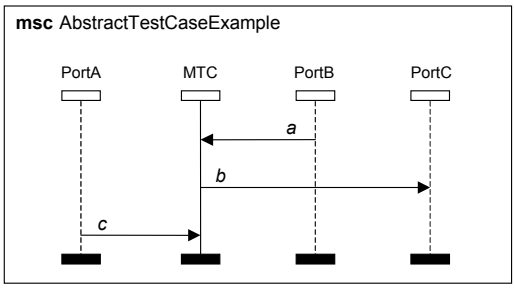


Figura 3/Z.142 – Descripción en MSC de la interacción de un MTC con interfaces SUT

Al describir estas dos categorías de utilización del MSC, pueden distinguirse dos esferas distintas de trabajo (véase figura 4):

a) *Generación de casos de pruebas abstractos para descripciones de objetivos de prueba en MSC* – El lenguaje núcleo TTCN-3 o GFT puede utilizarse para representar los casos de pruebas abstractas. Ahora bien, la generación de casos de prueba a partir de objetivos de prueba no es trivial e implica la utilización y desarrollo de técnicas de generación de pruebas asistidas por ordenador (CATG).

b) *Creación de un formato de presentación gráfica para TTCN-3 (GFT) y definición de la correspondencia entre GFT y TTCN-3.*

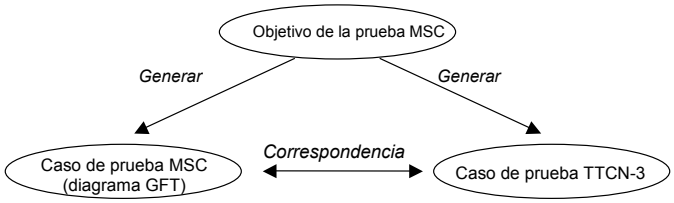


Figura 4/Z.142 – Relación entre la descripción del objetivo de la prueba MSC, descripciones de los casos de prueba MSC y TTCN-3

Esta Recomendación versa sobre el punto b), es decir, define el GFT y la correspondencia entre el GFT y el lenguaje núcleo de TTCN-3.

5 Conceptos de lenguaje GFT

El GFT representa gráficamente el comportamiento de TTCN-3 como si se tratase de un caso de prueba o una función. No describe gráficamente los aspectos de los datos tales como la declaración de tipos y plantillas.

El GFT no define la representación gráfica de la estructura de un módulo TTCN-3, sino que especifica los requisitos de tal representación gráfica (véase además la cláusula 7).

NOTA – El orden y la agrupación de las definiciones y declaraciones en la parte de definiciones del módulo define la estructura del módulo TTCN-3.

El GFT no define la representación gráfica de:

- definiciones de parámetro del módulo;
- definiciones de importación;
- definiciones de tipos;
- declaraciones de signaturas;
- declaraciones de plantillas;
- declaraciones de constantes;
- declaraciones de constante externas; y
- declaraciones de funciones externas.

Las definiciones y declaraciones TTCN-3 que no tienen una presentación GFT correspondiente pueden presentarse en el lenguaje núcleo TTCN-3 o en formato de presentación tabular para TTCN-3 (TFT) [2].

El GFT dispone de gráficos para las descripciones de comportamiento TTCN-3. Esto significa que un diagrama GFT proporciona una presentación gráfica de:

- la parte de control del módulo TTCN-3;
- un caso de prueba TTCN-3;
- una función TTCN-3; o
- una alternativa TTCN-3.

La figura 5 muestra la relación entre el módulo TTCN-3 y la correspondiente presentación GFT.

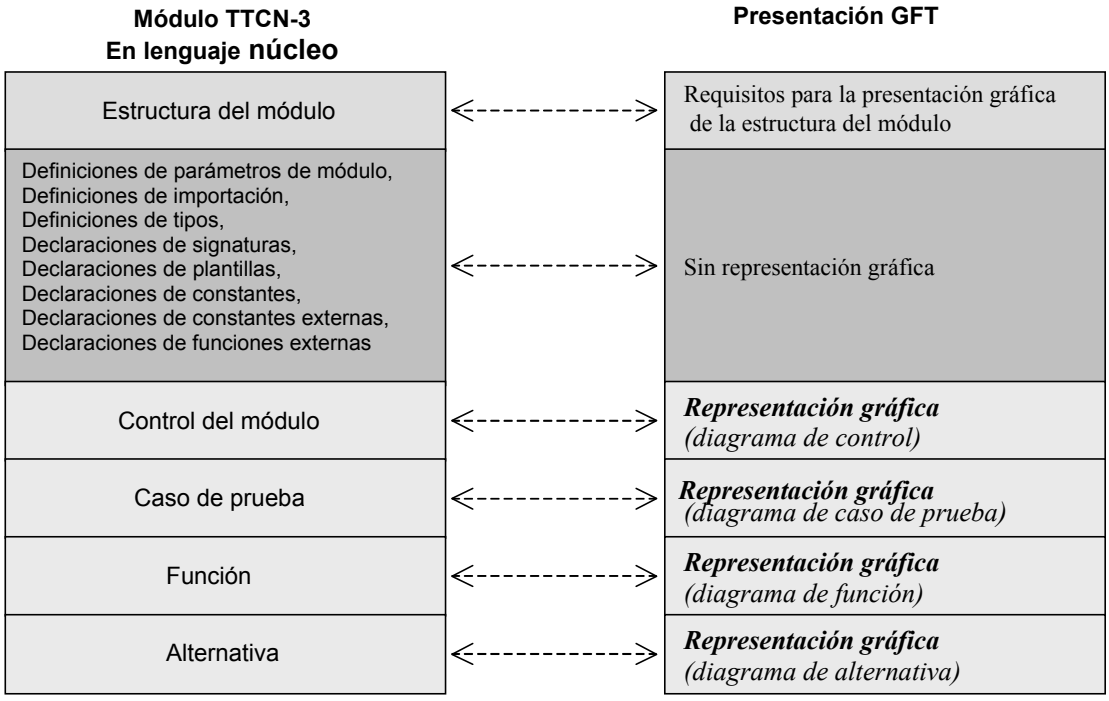


Figura 5/Z.142 – Relación entre el lenguaje núcleo TTCN-3 y la correspondiente descripción GFT

El GFT se basa en el MSC [3] y, por consiguiente, existe una correspondencia entre un diagrama GFT y un diagrama MSC. Aunque el GFT utiliza la mayor parte de los símbolos gráficos MSC, las inscripciones de algunos símbolos MSC se han tenido que adaptar a las necesidades de las pruebas y, además, se han definido nuevos símbolos para recalcar aspectos específicos de las pruebas. Ahora bien, los nuevos símbolos pueden hacerse corresponder en MSC válidos.

Los nuevos símbolos GFT se han definido para:

- la representación de ejemplares de puertos;
- la creación de componentes de prueba;
- el inicio de componentes de prueba;
- el argumento que devuelve la llamada a una función;
- la repetición de alternativas;
- la supervisión de tiempo de una llamada basada en procedimientos;
- la ejecución de casos de prueba;
- la activación y desactivación de valores por defecto;
- el etiquetado y la función salto;
- los temporizadores dentro de enunciados de llamada.

En la cláusula 8 figura una lista completa de todos los símbolos utilizados en GFT.

6 Correspondencia entre GFT y el lenguaje núcleo TTCN-3

El GFT proporciona mecanismos gráficos para definir el comportamiento TTCN-3. La parte de control y cada caso de función, alternativa y prueba de un módulo de lenguaje núcleo TTCN-3 puede hacerse corresponder con un diagrama GFT y viceversa, esto es:

- la parte de control del módulo puede hacerse corresponder con un diagrama de control (véase la cláusula 9.2) y viceversa;
- un caso de prueba puede hacerse corresponder con un diagrama de caso de prueba (véase cláusula 9.3) y viceversa;
- una función en el lenguaje núcleo puede hacerse corresponder con un diagrama de función (véase cláusula 9.4) y viceversa;
- una alternativa puede hacerse corresponder con un diagrama de alternativa (véase cláusula 9.5) y viceversa.

NOTA 1 – El GFT no proporciona presentaciones gráficas de las definiciones de los parámetros, tipos, constantes, signaturas, plantillas, constantes externas y funciones externas del módulo en la parte de definiciones del módulo. Estas definiciones pueden describirse directamente en lenguaje núcleo o utilizando otro formato de presentación, por ejemplo el formato de presentación tabular.

Cada declaración, operación y enunciado en el control del módulo y en cada caso de prueba, alternativa o función puede hacerse corresponder con una representación GFT y viceversa.

El orden de declaraciones, operaciones y enunciados en el control, caso de prueba, alternativa o definición de la función del módulo es idéntico al orden de las correspondientes representaciones GFT dentro del diagrama de control, caso de prueba, alternativa o función.

NOTA 2 – El orden de los elementos GFT en un diagrama GFT viene dado por el orden de los elementos GFT en el encabezamiento del diagrama (declaraciones únicamente) y en el orden de los elementos GFT a lo largo del ejemplar de control (diagrama de control) o ejemplar de componente (diagrama del caso de prueba, diagrama de alternativa o diagrama de función).

7 Estructura del módulo

Como se muestra en la figura 6, un módulo TTCN-3 tiene una estructura en árbol. El módulo TTCN-3 está estructurado en una parte de definiciones del módulo y una parte de control del módulo. La parte de definiciones del módulo consta de definiciones y declaraciones que pueden subdividirse nuevamente en grupos. En cambio, la parte del control del módulo no puede dividirse en subestructuras, ya que define el orden de ejecución y las condiciones de ejecución de los casos de prueba.

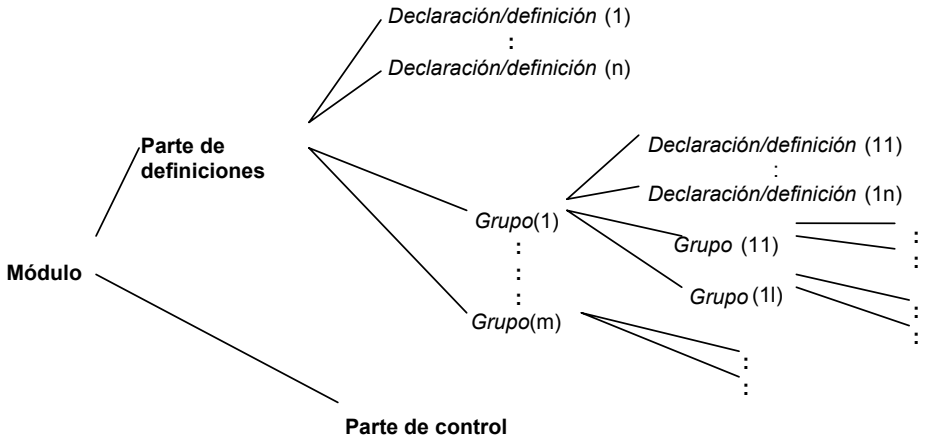


Figura 6/Z.142 – Estructura de los módulos TTCN-3

El GFT consta de diagramas de todas las ramas de "comportamiento" de la estructura en árbol del módulo, es decir, para la parte de control, las funciones, las alternativas y los casos de prueba del módulo. El GFT no define gráficos concretos para la estructura en árbol del módulo; sin embargo el soporte de una herramienta adecuada para GFT requiere la presentación gráfica de la estructura de un módulo TTCN-3. La estructura del módulo TTCN-3 puede mostrarse en la forma de un diagrama de organización (figura 7) o de una representación similar a la de un documento MSC (figura 8). Una herramienta avanzada podría además soportar diferentes presentaciones del mismo objeto, por ejemplo, el diagrama de organización de la figura 7 indica que algunas definiciones se proporcionan en diferentes formatos de presentación, por ejemplo, la función MySpecialFunction está disponible en lenguaje núcleo, en la forma de un cuadro TFT y como un diagrama GFT.

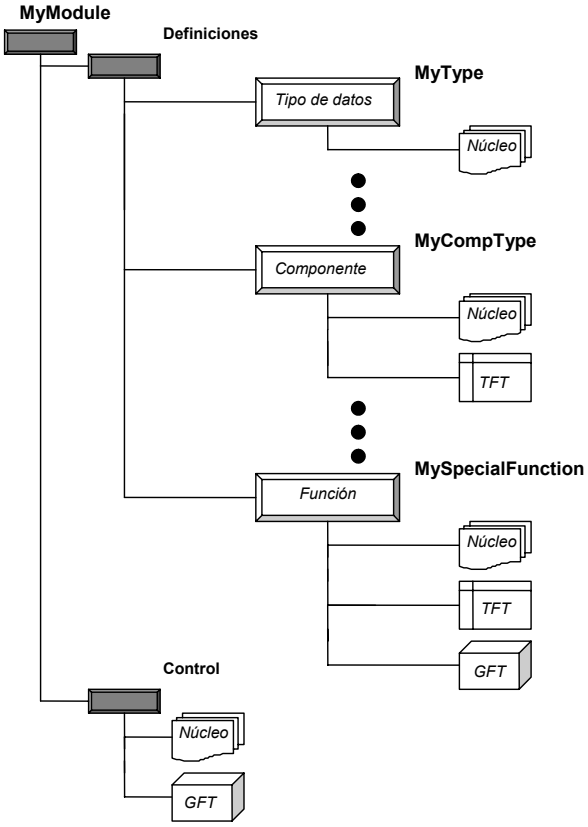


Figura 7/Z.142 – Varios formatos de presentación en un diagrama de organización de una estructura de módulo TTCN-3

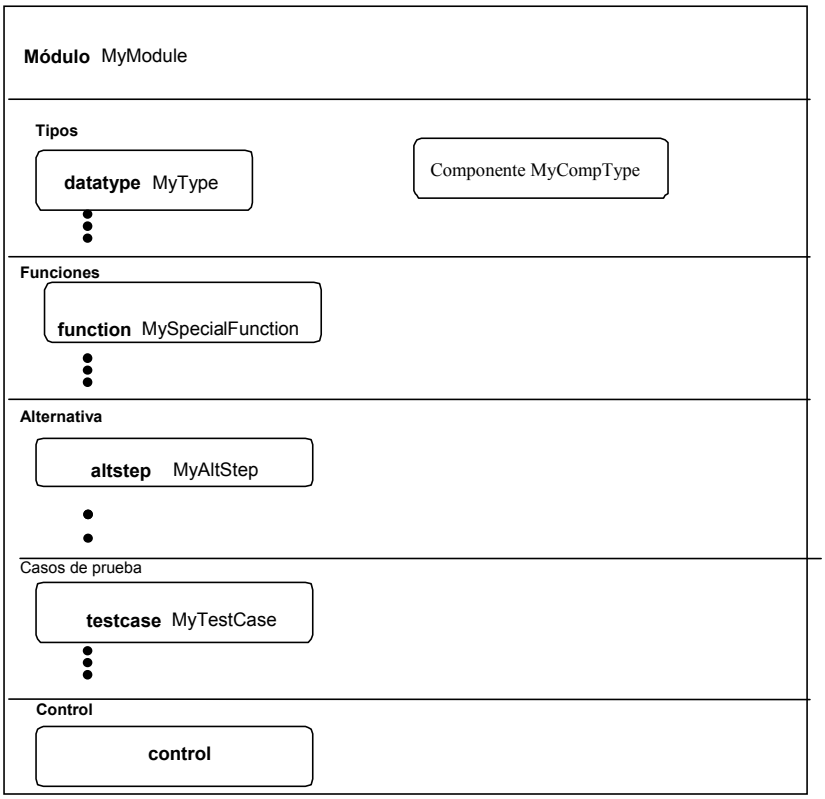
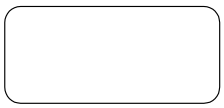
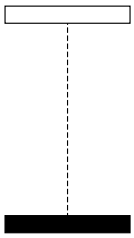
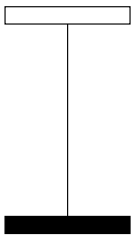
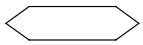
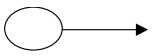
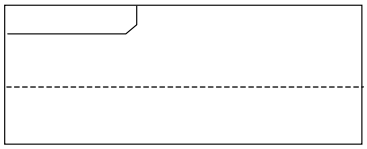
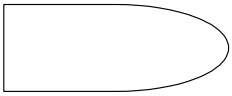
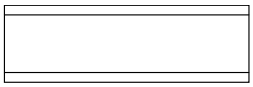
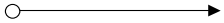


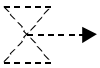
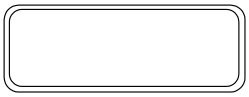
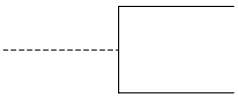
Figura 8/Z.142 – Presentación de una estructura de módulo TTCN-3 en forma de tipo documento MSC gráfico

8 Símbolos del GFT

En esta cláusula figuran todos los símbolos gráficos que se emplean en los diagramas GFT con sus respectivos comentarios de su utilización típica dentro de GFT.

Elemento GFT	Símbolo	Descripción
Símbolo marco		Utilizado para encuadrar diagramas GFT
Símbolo referencia		Utilizado para representar la llamada a funciones y alternativas
Símbolo ejemplar de puerto		Utilizado para representar ejemplares de puerto
Símbolo ejemplar de componente		Utilizado para representar componentes de prueba y la instancia de control
Símbolo recuadro de acción		Utilizado para las declaraciones y enunciados TTCN-3 textuales; se conectará a un símbolo componente
Símbolo condición		Utilizada en expresiones booleanas TTCN-3 textuales, establecimiento de veredictos, operaciones en puertos (iniciar, detener y borrar) y el enunciado done (fin); se conectará a un símbolo componente
Símbolo de etiquetado		Utilizado para etiquetado TTCN-3 y el enunciado goto (saltar a), se conectará a un símbolo componente
Símbolo saltar a		Utilizado para el etiquetado TTCN-3 y el enunciado goto (saltar a); se conectará a un símbolo componente

Elemento GFT	Símbolo	Descripción
Símbolo expresión en línea		Utilizado para enunciados los if-else, for, while, do-while, alt, call e interleave TTCN-3; se conectará a un símbolo componente
Símbolo valor por defecto		Utilizado para los enunciados activate (activación) y desactivate (desactivación) TTCN-3; se conectará a un símbolo componente
Símbolo detener		Utilizado para el enunciado stop (detener) TTCN-3, adjunto a un símbolo de componente
Símbolo volver		Utilizado para el enunciado volver TTCN-3; se conectará a un símbolo componente
Símbolo repetir		Utilizado para el enunciado repeat (repetir) TTCN-3; se conectará a un símbolo componente
Símbolo crear		Utilizado para el enunciado create (crear) TTCN-3; se conectará a un símbolo componente
Símbolo iniciar		Utilizado para el enunciado start (iniciar) TTCN-3, se conectará a un símbolo componente
Símbolo mensaje		Utilizado para los enunciados TTCN-3 send, call, replay, raise, receive, getcall, getreplay, match, trigger y check; se conectará a un símbolo componente y a un símbolo puerto
Símbolo hallado		Utilizado para representar los enunciados TTCN-3 receive, getcall, getreply, match, trigger y check desde cualquier puerto; se conectará a un símbolo componente
Símbolo región de suspensión		Utilizado junto con llamada con bloqueo, se utilizará en una expresión en línea de llamada y se conectará a un símbolo componente
Símbolo iniciar temporizador		Utilizado para la operación start timer (iniciar temporizador) TTCN-3, se conectará a un símbolo componente
Símbolo expiración del temporizador		Utilizado para la operación timeout (expiración del temporizador) TTCN-3, se conectará a un símbolo componente

Elemento GFT	Símbolo	Descripción
Símbolo detener temporizador		Utilizado para la operación stop timer (detener temporizador) TTCN-3, se conectará a un símbolo componente
Símbolo temporizador de inicio implícito		Utilizado para el inicio de temporizador implícito TTCN-3 en la llamada con bloqueo, se ubicará dentro de una expresión en línea de llamada y se conectará a un símbolo componente
Símbolo temporizador de expiración implícita		Utilizado para la excepción timeout (temporización) TTCN-3 en las llamadas con bloqueo, dentro de la expresión en línea de llamada y se conectará a un símbolo componente
Símbolo ejecutar		Utilizado para el enunciado execute test case (ejecutar caso de prueba) TTCN-3; se conectará a un símbolo ejemplar de componente
Símbolo texto		Utilizado para incluir enunciados y comentarios en TTCN-3; se ubicará dentro del diagrama GFT
Símbolo comentario de evento		Utilizado para indicar comentarios TTCN-3 relativos a eventos; se conectará a los eventos en los símbolos ejemplar de componente o instancia de puerto

9 Diagrama GFT

El GFT proporciona los siguientes tipos de diagramas:

a) *diagramas de control* para la presentación gráfica de la parte de control del módulo TTCN-3;

b) *diagrama de caso de prueba* para la presentación gráfica de caso de prueba TTCN-3;

c) *diagrama de alternativa* para la presentación gráfica de una alternativa (alternativa) TTCN-3; y

d) *diagrama de función* para la presentación gráfica de una función TTCN-3.

Los diferentes tipos de diagramas tienen algunas propiedades comunes.

9.1 Propiedades comunes

Las propiedades comunes de los diagramas GFT están relacionadas con el área, encabezamiento y paginación del diagrama.

9.1.1 Área del diagrama

Cada diagrama de control GFT, de caso de prueba, alternativa y función tendrá un símbolo marco (también llamado marco de diagrama) para definir el área del diagrama. Todos los símbolos y textos necesarios para definir de manera completa y sintácticamente correcta el diagrama GFT deberá estar dentro de la zona del diagrama.

NOTA – El GFT no dispone de constructivos de lenguaje como las puertas MSC, que se ubican en el exterior de las mismas pero están conectadas al marco de diagrama.

9.1.2 Encabezamiento del diagrama

Cada diagrama GFT dispone de un encabezamiento. El encabezamiento de diagrama deberá ubicarse en la esquina superior de izquierda del marco del diagrama.

El encabezamiento del diagrama identificará inequívocamente cada tipo de diagrama GFT. La regla general para ello es construir el encabezamiento mediante las palabras clave **testcase**, **altstep** o **function** seguido de la signatura TTCN-3 del caso de prueba, alternativa o función que debe presentarse gráficamente. Para un diagrama de control GFT, el encabezamiento inequívoco se crea con la palabra clave **module** seguido del nombre del módulo.

NOTA – En MSC, la palabra clave **msc** siempre va precedida del nombre del diagrama para identificar los diagramas MSC. Los diagramas GFT no disponen de una palabra clave común para identificar los diagramas GFT.

9.1.3 Paginación

Los diagramas GFT pueden constar de varias páginas, de modo que un diagrama GFT grande pueda dividirse en varias páginas. Cada página de un diagrama debe estar numerada, de manera inequívoca, en la esquina superior derecha. La numeración es opcional si el diagrama no está dividido.

NOTA 1 – El método de numeración concreto que se utilice se considera un asunto que depende de la herramienta y, por consiguiente, queda fuera del alcance de esta Recomendación. Un método de numeración sencillo podría consistir únicamente en asignar un número a la página, mientras que uno avanzado podría permitir la reconstrucción de un diagrama basándose únicamente en la información relativa a la numeración de las diferentes páginas.

NOTA 2 – Los requisitos de la paginación aparte de la numeración general se considera un asunto que depende de la herramienta y, por consiguiente, quedan fuera del alcance de esta Recomendación. Para facilitar la lectura, en cada página podría mostrarse el encabezamiento del diagrama; la línea de ejemplar de un ejemplar que continuará en otra página podría adjuntarse al borde inferior de la página; y el encabezamiento del ejemplar de un ejemplar que continúa puede repetirse en la página que describe la continuación.

9.2 Diagrama de control

Un diagrama de control GFT representa gráficamente la parte de control de un módulo TTCN-3. El encabezamiento de un diagrama de control debe constar de la palabra clave **module** seguido del nombre del módulo. El diagrama de control GFT solo incluirá un ejemplar del componente (a veces denominado ejemplar de control) con el nombre del ejemplar **control** sin ningún otro tipo de información. El ejemplar de control describe el comportamiento de la parte de control del módulo TTCN-3. Los atributos relacionados con la parte de control del módulo TTCN-3 deben especificarse dentro de un símbolo texto dentro el diagrama de control. La configuración básica de un diagrama de control GFT y la correspondiente descripción en lenguaje núcleo TTCN-3 se muestra en la figura 9.

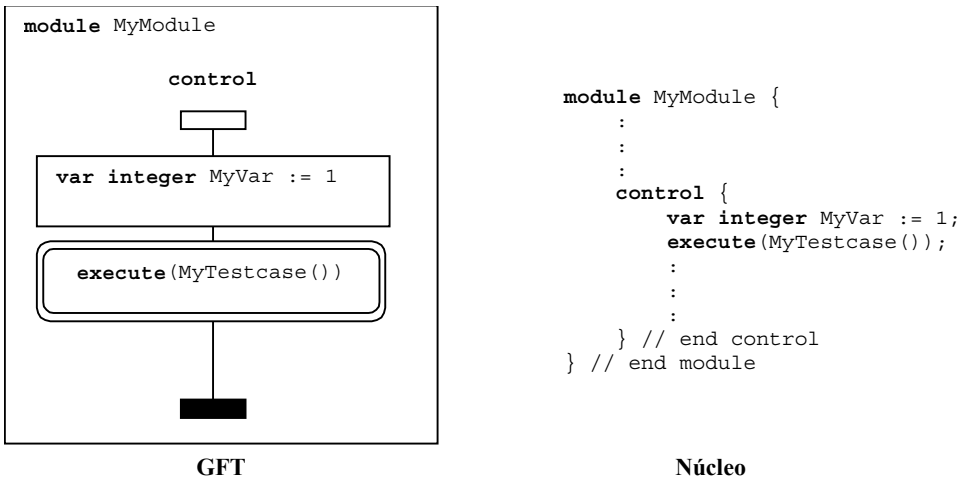


Figura 9/Z.142 – Configuración básica de un diagrama de control GFT y su correspondiente lenguaje núcleo

Dentro de la parte de control, los casos de prueba pueden seleccionarse o deseleccionarse para ejecutar casos de prueba mediante la utilización de expresiones booleanas. Para controlar la ejecución de los casos de prueba pueden utilizarse expresiones, asignaciones, enunciados **log**, enunciados **label** y **goto**, enunciados **if-else**, ciclos **for**, ciclos **while**, ciclos **do while**, enunciados de ejecución **stop** y enunciados de temporización. Además, pueden emplearse funciones para agrupar los casos de prueba junto con sus condiciones previas a la ejecución, funciones a las que se llama desde la parte de control del módulo.

La representación GFT de estas características del lenguaje se describe en las correspondientes cláusulas que figuran a continuación excepto para la parte de control de módulo en la que los símbolos gráficos están anexos al ejemplar de control en lugar de a un ejemplar de componente de prueba.

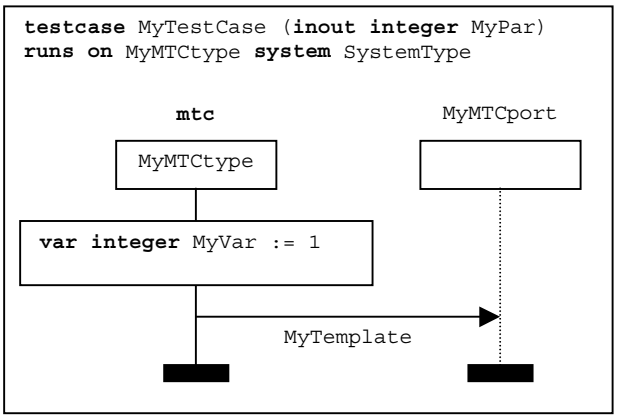
Véase 11.4 para la representación GFT de expresiones, asignaciones, proposiciones **log**, **label** y **goto**, **if-else**, ciclos **for**, ciclos **while**, ciclos **do while** y **stop**, 11.9 para operaciones de temporización y 9.4 y 11.2.2 para las funciones y su invocación.

9.3 Diagrama de caso de prueba

Un diagrama de caso de prueba GFT es una presentación gráfica del caso de prueba TTCN-3. En el encabezamiento del diagrama de caso de prueba debe figurar la palabra clave **testcase** seguido de la signatura completa del caso de prueba. Por completa se entiende que al menos figurará el nombre y la lista de parámetros. La cláusula **runs on** es obligatoria y la cláusula **system** es opcional en el lenguaje núcleo. Si la cláusula system se especifica en el correspondiente lenguaje núcleo, también deberá figurar en el encabezamiento del diagrama del caso de prueba.

El diagrama del caso de prueba GFT incluirá un ejemplar del componente de prueba que describe el comportamiento del **mtc** (también llamado ejemplar mt) y un ejemplar del puerto para cada puerto que pertenezca el **mtc**. El nombre asociado al ejemplar mtc será **mtc**. El tipo asociado con el ejemplar mtc es opcional, pero de estar presente en la información de tipo, éste será idéntico al tipo de componente al que se hace referencia en la cláusula **runs on** de la signatura del caso de prueba. Los nombres asociados a los ejemplares del puerto serán idénticos a los nombres del puerto definidos en la definición del tipo de componente del **mtc**. La información de tipo asociada para los ejemplares del puerto es opcional. De estar presente la información de tipo, los nombres de los puertos y los tipos de puertos serán coherentes con la definición del tipo de componente del **mtc**. Los tipos **mtc** y de puerto se muestran en el símbolo del encabezamiento del ejemplar del puerto o del componente.

Los atributos asociados al caso de prueba presente en el GFT se especificarán dentro del símbolo de texto en el diagrama del caso de prueba. La configuración básica de un diagrama de caso de prueba GFT y su correspondiente descripción en lenguaje núcleo TTCN-3 se muestran en la figura 10.




```
testcase MyTestCase (inout integer MyPar)
runs on MyMTCtype system SystemType {

    var integer MyVar := 1;

    MyMTCPort.send(MyTemplate);

        :
        :
}
```

GFT **Núcleo**

Figura 10/Z.142 – Configuración básica de un diagrama de caso de prueba GFT y su correspondiente lenguaje núcleo

El caso de prueba representa el comportamiento dinámico de prueba y puede crear componentes de prueba. Un caso de prueba puede contener declaraciones, enunciados, operaciones de comunicación y de temporización e invocación de funciones o alternativas.

9.4 Diagrama de función

El GFT representa las funciones TTCN-3 mediante diagramas de funciones. El encabezamiento de un diagrama de función será la palabra clave **function** seguido de la signatura completa de la función. Por completa se entiende que al menos figurará el nombre y la lista de parámetros. La cláusula **return** y la cláusula **runs on** son opcionales en lenguaje núcleo. Si estas cláusulas se especifican en el correspondiente lenguaje núcleo, también deberán figurar en el encabezamiento del diagrama de la función.

Los diagramas de la función GFT deben incluir un ejemplar del componente de prueba que describe el comportamiento de la función y un ejemplar de puerto para cada puerto que emplee la función.

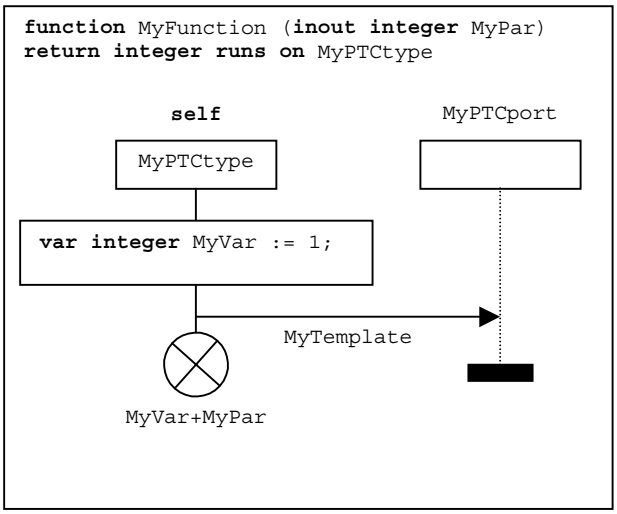
NOTA – Los nombres y tipos de los puertos que utilizan la función se pasan como argumentos o son nombres y tipos de puertos que están definidos en la definición del tipo de componente a que hace referencia la cláusula **runs on**.

El nombre asociado al ejemplar del componente de prueba será **self**. El tipo asociado al ejemplar del componente de prueba es opcional, pero de estar presente la información de tipo, éste deberá ser coherente con el tipo de componente en la cláusula **runs on**.

Los nombres y tipos asociados al ejemplar del puerto deben ser coherentes con los parámetros del puerto (en caso de que los puertos que utiliza la función se pasen como argumentos) o con las declaraciones del puerto en una definición del tipo de componente a que se hace referencia en la cláusula **runs on**. La información de tipo para los ejemplares de puerto es opcional.

Los nombres **self** y del puerto figurarán en la parte superior del componente y en el símbolo del encabezamiento del respectivo ejemplar del puerto. Los tipos de componente y tipos de puerto aparecerán dentro del componente y símbolo de encabezamiento del respectivo ejemplar de puerto.

Los atributos asociados a la función presentes en GFT se especificarán dentro de un símbolo de texto en el diagrama de función. La configuración básica de un diagrama de función GFT y la correspondiente descripción del núcleo TTCN-3 se muestran en la figura 11.




```
function MyFunction (inout integer MyPar)
 return integer runs on MyPTCtype {

        var integer MyVar := 1;

        MyPTCport.send(MyTemplate);

        :
        :

        return MyVar+MyPar
}
```

GFT **Núcleo**

Figura 11/Z.142 – Configuración básica de un diagrama de función GFT y su correspondiente lenguaje núcleo

Las funciones se utilizan para especificar y estructurar el comportamiento de la prueba, definir el comportamiento por defecto o estructurar el cálculo en un módulo. Las funciones pueden contener declaraciones, enunciados, operaciones de comunicación y temporización e invocación de esas funciones o alternativas y, como opción, un enunciado volver.

9.5 Diagrama de alternativas

El GFT presenta las alternativas TTCN-3 mediante diagramas de alternativas. El encabezamiento de un diagrama de alternativas deberá ser la palabra clave **altstep** seguida de la signatura completa de la alternativa. Por completa se entiende que al menos figurará el nombre y la lista de parámetros. La cláusula **runs on** es opcional en el lenguaje núcleo. Si la cláusula **runs on** se especifica en el correspondiente lenguaje núcleo, también deberá figurar en el encabezamiento del diagrama de alternativas.

El diagrama de alternativas GFT deberá incluir un ejemplar del componente de prueba que describe el comportamiento de la alternativa y un ejemplar a puerto para cada puerto que la utilice.

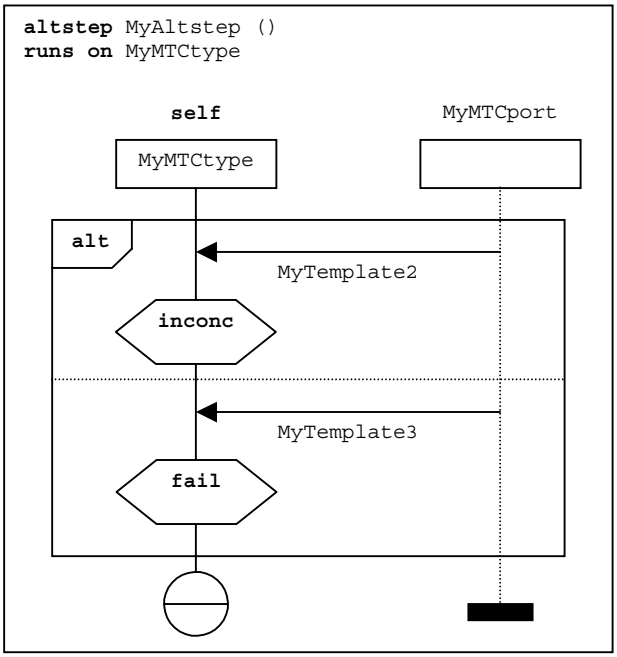
NOTA – Los nombres y tipos de los puertos que utiliza una alternativa se pasan como parámetros o son los nombres y tipos de puerto que se definen en la definición del tipo de componente a que se hace referencia en la cláusula **runs on**.

El nombre asociado con el ejemplar del componente de prueba deberá ser **self**. El tipo asociado al ejemplar del componente de pruebas es opcional, pero de estar presente la información de tipo, éste deberá ser coherente con el tipo de componente que figura en la cláusula **runs on**.

Los nombres y tipos asociados a los ejemplares del puerto deberán ser coherentes con los parámetros de puerto (si los puertos que utiliza la alternativa se pasan como parámetros) o las declaraciones de puerto en la definición de tipo de componente a que se hace referencia en la cláusula **runs on**. La información de tipo para los ejemplares de puerto es opcional.

Los nombres **self** y puerto figurarán en la parte superior del componente y en el respectivo símbolo de encabezamiento del respectivo ejemplar del puerto. Los tipos de componentes y los tipos de puertos aparecerán dentro del componente y el respectivo símbolo en el encabezamiento del ejemplar de puerto.

Los atributos asociados a la alternativa se especificarán dentro del símbolo de texto del diagrama de alternativa GFT. La configuración básica del diagrama GFT y su correspondiente lenguaje núcleo TTCN-3 se muestran en la figura 12.



GFT

```
altstep MyAltstep () runs on MyMTCtype {

    alt {

        []  MyMTCport.receive(MyTemplate2) {
                setverdict(inconc)
            }
        []  MyMTCport.receive(MyTemplate3) {
                setverdict(fail)
            }
    }
        :
        :
    repeat
}
```

Núcleo

Figura 12/Z.142 – Configuración básica del diagrama de alternativa GFT y su correspondiente lenguaje núcleo

La alternativa se utiliza para verificar el comportamiento por defecto o estructurar las alternativas de un enunciado **alt**. Las alternativas pueden contener enunciados, operaciones de comunicación y temporización y llamadas o funciones u otras alternativas.

10 Ejemplares en diagramas GFT

Los diagramas GFT incluyen los siguientes tipos de ejemplares:

- *ejemplares de control* que describen el flujo de control de la parte de control del módulo;
- *ejemplares del componente de prueba* que describen el flujo de control para el componente de prueba que ejecuta el caso de prueba, función o alternativa;
- *ejemplares de puerto* que representan los puertos utilizados por los diferentes componentes de prueba.

10.1 Ejemplares de control

El diagrama de control GFT sólo puede contener un ejemplar de control (véase 9.2). El ejemplar de control describe el flujo de control de la parte de control del módulo. El ejemplar de control GFT se describirá gráficamente mediante un símbolo del ejemplar de componente con el nombre obligatorio de **control** en la parte superior del símbolo del encabezamiento del ejemplar. El

ejemplar de control no tiene asociada información de tipo de ejemplar. La configuración básica del ejemplar de control se muestra en la figura 13 a).

10.2 Ejemplares del componente de prueba

Cada diagrama de caso de prueba, función o alternativa en GFT incluye un ejemplar del componente de prueba que describe el flujo de control de ese ejemplar. El ejemplar del componente de prueba GFT se describirá gráficamente mediante el símbolo de ejemplar del modo siguiente:

- el nombre obligatorio **mtc** estará en la parte superior del símbolo del encabezamiento del ejemplar en el caso de un diagrama de caso de prueba;
- el nombre obligatorio **self** estará situado en la parte superior del símbolo del encabezamiento del ejemplar en el caso de un diagrama de función o alternativa.

Dentro del símbolo de encabezamiento de ejemplar puede incluirse el tipo de componente de prueba. Éste habrá de ser coherente con el tipo de componente de prueba que figura después de la palabra clave **runs on** en el encabezamiento del diagrama GFT.

La configuración básica del ejemplar del componente de prueba en un diagrama de caso de prueba se muestra en la figura 13 b). La configuración básica de un ejemplar del componente de prueba en un diagrama de función o alternativa se muestran en la figura 13 c).

10.3 Ejemplares de puerto

Los ejemplares de puerto GFT pueden utilizarse dentro de los diagramas de caso de prueba, alternativa y función. El ejemplar de puerto representa un puerto que pueden utilizar los componentes de prueba que ejecutan ese determinado caso de prueba, alternativa o función. El ejemplar de puerto GFT se describe gráficamente mediante un símbolo de ejemplar de componente con una línea discontinua de ejemplar. El nombre del puerto representado es obligatorio y deberá figurar en la parte superior del símbolo del encabezamiento del ejemplar. El símbolo del encabezamiento del ejemplar puede contener un tipo de puerto (opcional). En la figura 13 d) se muestra la configuración básica de un ejemplar de puerto.

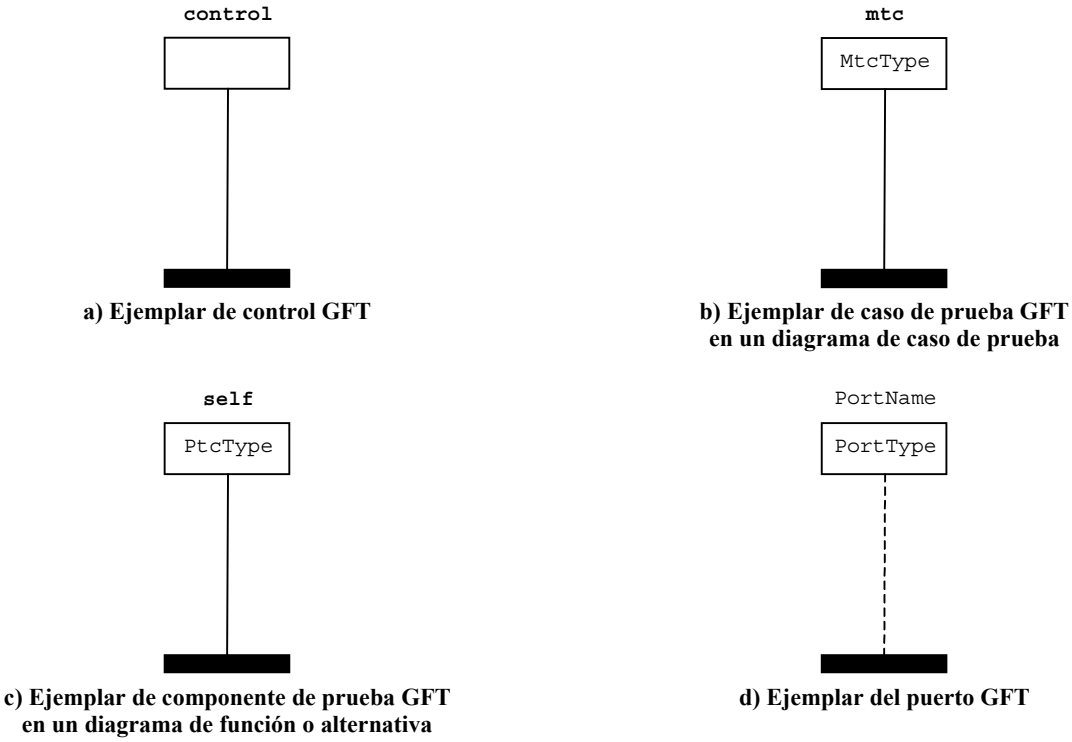


Figura 13/Z.142 – Configuración básica de los tipos de ejemplar en diagramas GFT

11 Elementos de los diagramas GFT

En esta cláusula se definen las reglas de dibujo generales para la representación de elementos sintácticos TTCN-3 concretos (puntos y coma, comentarios). Se describe cómo dibujar la ejecución de diagramas GFT y los símbolos gráficos correspondientes a los elementos de lenguaje TTCN-3.

11.1 Reglas de dibujo generales

Las reglas de dibujo generales en GFT están relacionadas con la utilización de puntos y coma, enunciados TTCN-3 en símbolos de acción y comentarios.

11.1.1 Utilización de puntos y coma

Todos los símbolos GFT salvo el símbolo acción deberán incluir un único enunciado en el lenguaje núcleo TTCN-3. Únicamente los símbolos de acción pueden incluir una secuencia de enunciados TTCN-3 (véase 11.1.2).

El punto y coma es opcional si el símbolo GFT incluye solamente un enunciado en lenguaje núcleo TTCN-3 (véanse las figuras 14 a) y 14 b)).

Los puntos y coma separan los enunciados en una secuencia de enunciados dentro de un símbolo de acción. El punto y coma es opcional para el último enunciado en la secuencia (figura 14 c)).

Una secuencia de variables, constantes y declaraciones de temporización puede también especificarse en lenguaje núcleo TTCN-3 después del encabezamiento de un diagrama GFT. Estas declaraciones también deberán estar separadas mediante puntos y coma. El punto y coma es opcional para la última declaración de la secuencia.

11.1.2 Utilización de símbolos de acción

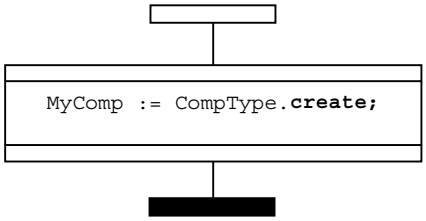
Las siguientes declaraciones, enunciados y operaciones TTCN-3 se especifican dentro de los símbolos de acción: declaraciones (con las restricciones definidas en 11.3), asignaciones, **log**, **connect**, **disconnect**, **map**, **unmap** y **action**.

Una secuencia de declaraciones, enunciados y operaciones que deberán de especificarse dentro de variables de símbolos de acción pueden especificarse en un solo símbolo de acción. No es necesario utilizar símbolos de acción separados para cada declaración, enunciado u operación.

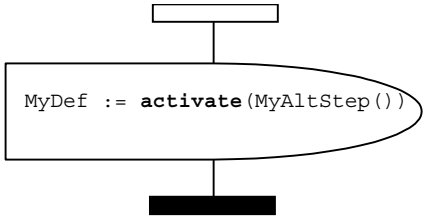
11.1.3 Comentarios

El GFT proporciona tres posibilidades para incluir comentarios en los diagramas GFT:

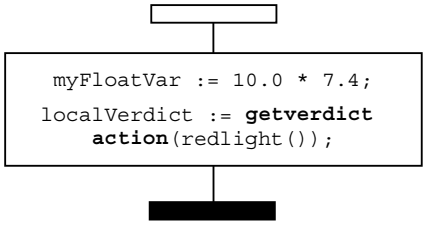
- Pueden incluirse en los símbolos GFT después de la inscripción del símbolo y mediante la sintaxis de comentarios del lenguaje núcleo de TTCN-3 (figura 14 d)).
- Los comentarios en la sintaxis de comentarios del lenguaje núcleo TTCN-3 pueden figurar en símbolos de texto y situarse en cualquier lugar de la zona del diagrama GFT (figura 14 e)).
- El símbolo de comentario puede utilizarse para relacionar comentarios con símbolos GFT. Los comentarios en los símbolos comentario pueden escribirse como texto normal, es decir, sin utilizar los delimitadores de comentario '/*', '*/' y '//' (figura 14 f)).



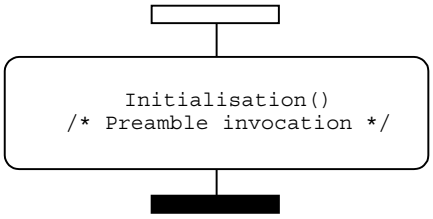
a) Creación de un componente con un punto y coma opcional de terminación



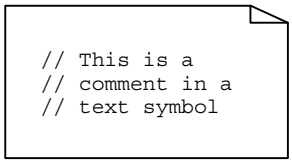
b) Activación por efecto sin un punto y coma de terminación



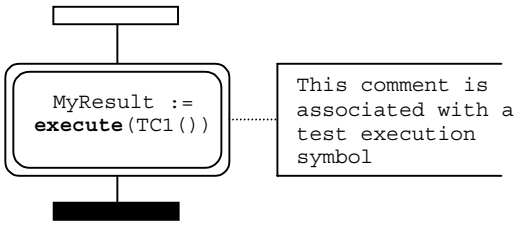
c) Secuencia de enunciados en un símbolo de acción



d) Comentario dentro un símbolo de referencia GFT



e) Comentario en un símbolo de texto



f) Comentario dentro de un símbolo comentario relacionado con un símbolo de ejecución

Figura 14/Z.142 – Ejemplo de los efectos de reglas de dibujo generales

11.2 Invocación a diagramas GFT

Esta cláusula describe cómo se invoca a los diferentes tipos de diagrama GFT. Dado que no existe un enunciado para ejecutar la parte de control en TTCN-3 (puesto que es comparable a la ejecución

de un programa mediante el enunciado main y queda fuera del alcance de TTCN-3), en esta cláusula se describe la ejecución de casos de prueba, funciones y alternativas.

11.2.1 Ejecución de casos de prueba

La ejecución de casos de prueba se representa mediante la utilización del símbolo caso de prueba execute (ejecutar) (véase figura 15). La sintaxis del enunciado **execute** se incluye dentro del símbolo. El símbolo puede contener:

- un enunciado **execute** para el caso de prueba con parámetros opcionales y supervisión de tiempo;
- como opción, la asignación del veredicto/devuelto a una variable **verdicttype**; y
- como opción, la declaración en línea de la variable **verdicttype**.

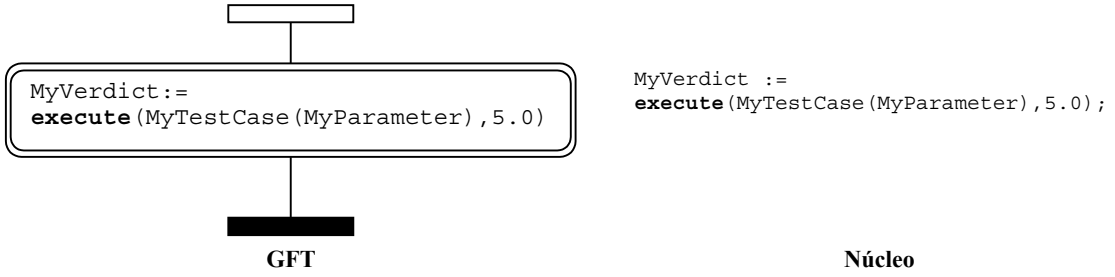


Figura 15/Z.142 – Ejecución del caso de prueba

11.2.2 Invocación a funciones

La invocación a funciones se representa mediante el símbolo referencia (figura 16), salvo funciones externas y predefinidas (figura 17) y cuando se invoca a la función desde dentro del elemento de lenguaje TTCN-3 que tiene una representación GFT (figura 18).

La sintaxis de la invocación función se ubica dentro del símbolo de referencia. El símbolo puede contener:

- la invocación a una función con parámetros opcionales;
- como opción, la asignación a una variable del valor que devuelve la función; y
- como opción, la declaración en línea de la variable.

El símbolo de referencia sólo se utiliza para las funciones definidas por el usuario definidas dentro del módulo en cuestión. No debe utilizarse para funciones externas o funciones TTCN-3 predefinidas, las cuales se representarán en forma de texto dentro de un símbolo acción (figura 17) u otros símbolos GFT (véase el ejemplo en la figura 18).

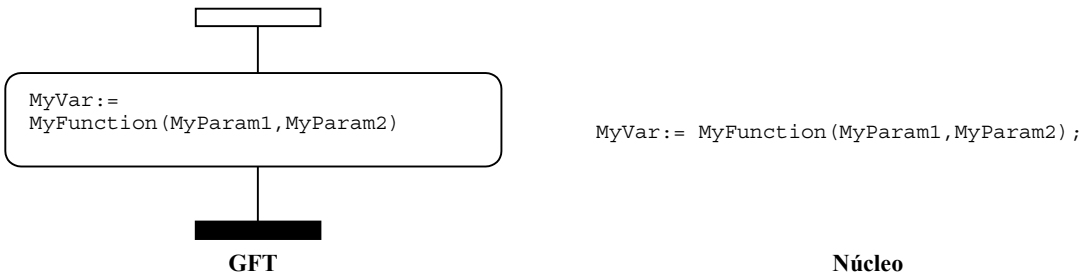


Figura 16/Z.142 – Invocación a funciones definidas por el usuario

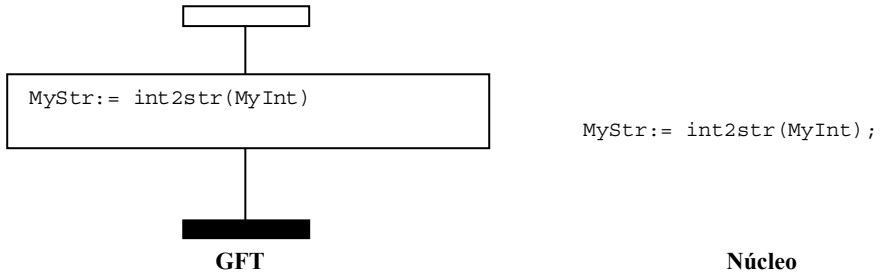


Figura 17/Z.142 – Invocación a funciones predefinidas/externa

Las funciones que se invocan dentro de un constructivo TTCN-3 con un correspondiente símbolo GFT se representa en forma de texto dentro de ese símbolo.

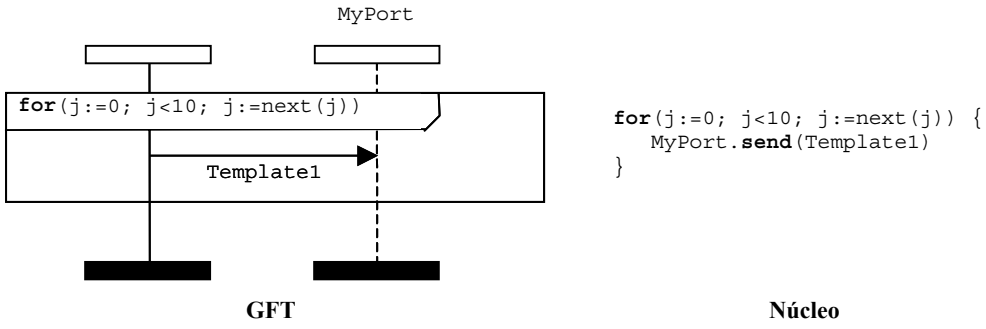


Figura 18/Z.142 – Invocación a una función definida por el usuario dentro de un símbolo GFT

11.2.3 Invocación a alternativas

La invocación de alternativas se representa mediante la utilización del símbolo referencia (véase la figura 19). La sintaxis de invocación a alternativas se sitúa dentro del símbolo. El símbolo puede contener la invocación de una alternativa con parámetros opcionales. Se utilizará dentro del comportamiento alternativo únicamente, cuando la invocación a la alternativa sea uno de los operandos del enunciado alternativo (véase también la figura 32 en 11.5.2).

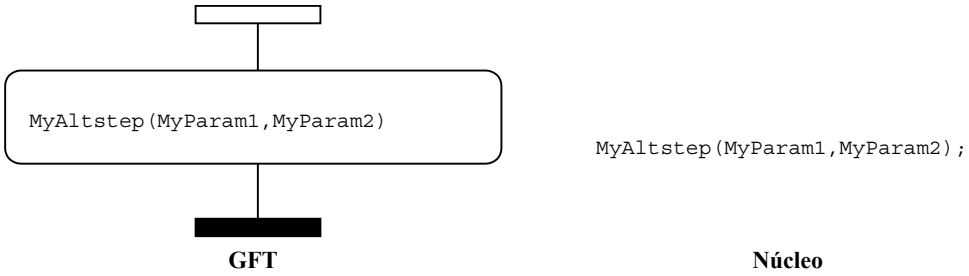


Figura 19/Z.142 – Invocación a una alternativa

Otra posibilidad es la invocación implícita a alternativas mediante valores por defecto activados. Para mayor información, véase 11.6.2.

11.3 Declaraciones

El TTCN-3 permite la declaración e inicialización de temporizadores, constantes y variables al principio de los bloques de enunciado. El GFT utiliza la sintaxis del lenguaje núcleo TTCN-3 para declaración en diversos símbolos. El tipo de un símbolo depende de la especificación de la inicialización, por ejemplo, una variable de tipo **default** que se inicializa mediante la operación **activate** se especificará dentro del símbolo por defecto (véase 11.6).

11.3.1 Declaración de temporizadores, constantes y variables dentro del símbolo acción

Las siguientes declaraciones deberán realizarse dentro de símbolos acción:

- declaración de temporizadores;
- declaración de variables sin inicialización;
- declaración de variables y constantes con inicialización;
  - si la inicialización no se realiza mediante funciones que incluyen funciones de comunicación; o
  - si la declaración es:
    - de un tipo de componente que no se inicializa mediante la operación **create**;
    - de un tipo **default** que no se inicializa mediante la operación **activate**;
    - de un tipo **verdicttype** que no se inicializa mediante un enunciado **execute**;
    - de un tipo básico simple;
    - de un tipo cadena básico;
    - de un tipo **anytype**;
    - de un tipo puerto;
    - de un tipo **address**; o
    - de un tipo estructurado definido por el usuario con campos que cumplen todas las restricciones indicadas en esta lista para el caso de "declaración de variables y constantes con inicialización".

NOTA – Para una descripción general de los tipos TTCN-3, véase el cuadro 3/Z.140 [1].

Una secuencia de declaraciones que se hará dentro de los símbolos acción pueden ponerse dentro de un mismo símbolo de acción, esto es, no es necesario emplear varios símbolos acción separados. Pueden verse ejemplos de declaraciones dentro de símbolos de acción en las figuras 20 a) y 20 b).

11.3.2 Declaración de constantes y variables dentro del símbolo expresiones en línea

Las declaraciones de constantes y variables de un tipo de componente que se inicialicen dentro de enunciados **if-else**, **for**, **while**, **do-while**, **alt** o **interleave** figurarán dentro del mismo símbolo de expresión en línea.

11.3.3 Declaración de constantes y variables dentro del símbolo crear

Las declaraciones de constantes y variables de un tipo de componente que se inicialicen mediante operaciones **create** se harán dentro del símbolo crear. A diferencia de las declaraciones dentro del símbolo acción, cada declaración que se inicialice mediante las operaciones **create** debe figurar en un símbolo crear separado. En la figura 20 c) puede verse un ejemplo de la declaración de una variable dentro de un símbolo crear.

11.3.4 Declaración de constantes y variables dentro del símbolo valor por defecto

Las declaraciones de constantes y variables del tipo **default** que se inicialicen mediante operaciones **activate** se realizarán dentro del símbolo default. A diferencia de declaraciones dentro del símbolo action, cada declaración que se inicialice mediante una operación **activate** se deberán presentar en

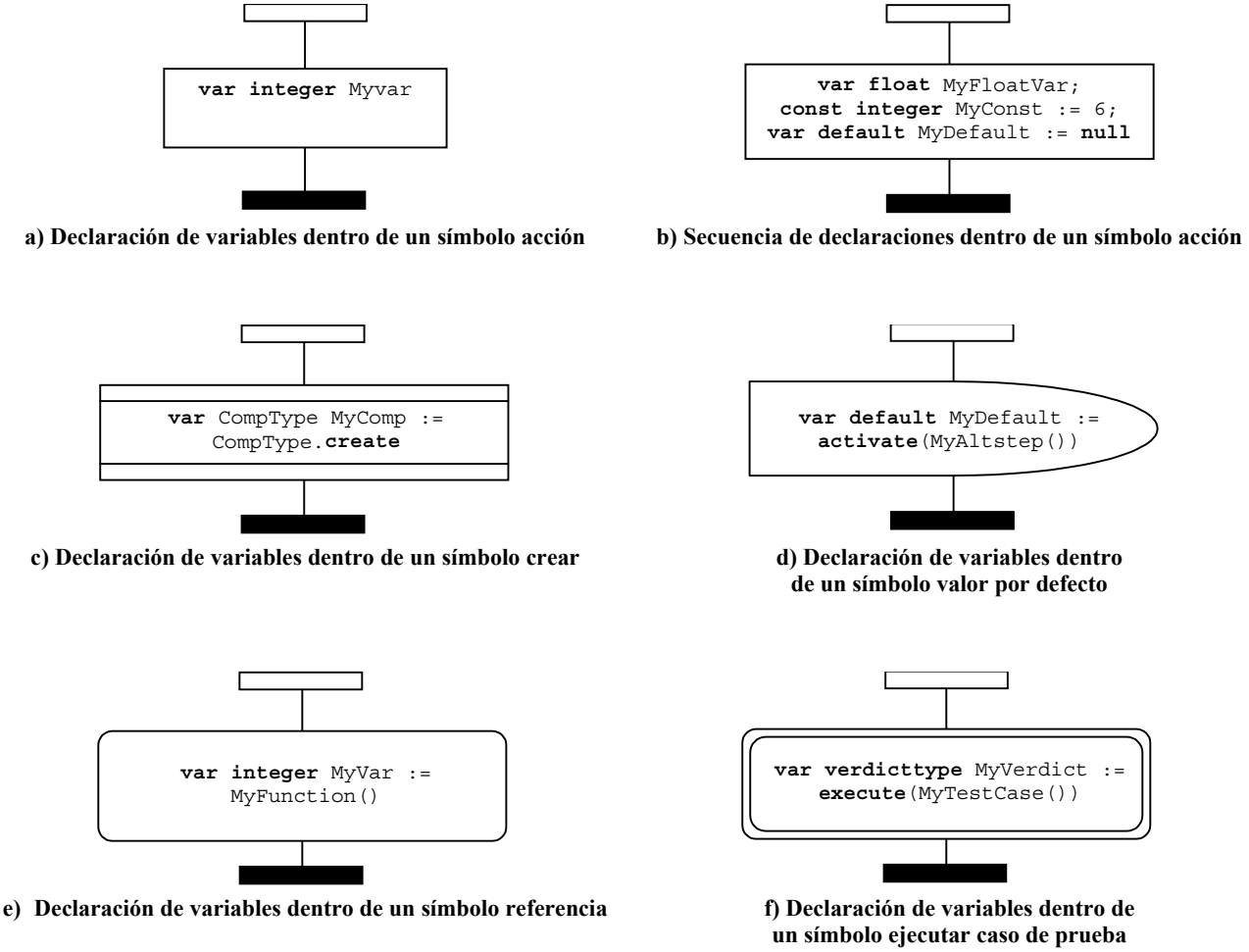
símbolos default separados. En la figura 20 d) puede verse un ejemplo de una declaración de variable dentro de un símbolo default.

11.3.5 Declaración de constantes y variables dentro del símbolo referencia

Las declaraciones de constantes y variables que se inicialice mediante una función, que incluye operaciones de comunicación, se realizarán dentro de símbolos referencia. A diferencia de las declaraciones dentro de símbolos acción, cada declaración que se inicialice mediante una función, que incluya funciones de comunicación, se presentará en un símbolo referencia separado. En la figura 20 e) puede verse un ejemplo de declaración de variables dentro de un símbolo referencia.

11.3.6 Declaración de constantes y variables dentro del símbolo ejecutar caso de prueba

Las declaraciones de constantes y variables del tipo **verdicttype** que se inicialicen mediante enunciados **execute** se harán dentro de símbolos ejecutar caso de prueba. A diferencia de las declaraciones dentro de símbolos acción, cada declaración que se inicialice mediante un enunciado **execute** deberá figurar en un símbolo ejecutar caso de prueba separado. En la figura 20 f) puede verse un ejemplo de declaración de variable dentro de un símbolo ejecutar caso de prueba.



a) Declaración de variables dentro de un símbolo acción

b) Secuencia de declaraciones dentro de un símbolo acción

c) Declaración de variables dentro de un símbolo crear

d) Declaración de variables dentro de un símbolo valor por defecto

e) Declaración de variables dentro de un símbolo referencia

f) Declaración de variables dentro de un símbolo ejecutar caso de prueba

Figura 20/Z.142 – Ejemplos de declaraciones en GFT

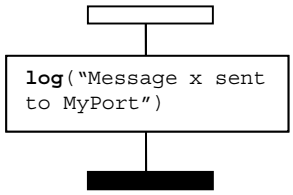
11.4 Enunciados de programa básicos

Los enunciados de programa básicos son expresiones, asignaciones, operaciones, bucles, etc. Todos los enunciados de programas básicos pueden utilizarse dentro de los diagramas GFT para la parte de control, casos de prueba, funciones y alternativas.

El GFT no proporciona una representación gráfica de expresiones y asignaciones. Éstas se indican mediante texto en los lugares donde se van a utilizar. Los gráficos se proporcionan para las proposiciones **log**, **label**, **goto**, **if-else**, **for**, **while** y **do-while**.

11.4.1 El enunciado log

El enunciado `log` se representará dentro de un símbolo acción (véase la figura 21).



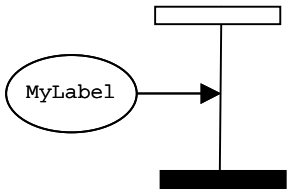

```
log("Message x sent to MyPort");
```

GFT **Núcleo**

Figura 21/Z.142 – Enunciado log

11.4.2 El enunciado label

El enunciado **label** se representará con el símbolo etiqueta, conectado a un ejemplar de componente. En la figura 22 se ilustra un ejemplo sencillo de enunciado **label** denominado `MyLabel`.



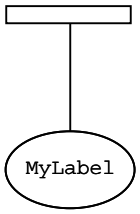

```
label MyLabel;
```

GFT **Núcleo**

Figura 22/Z.142 – Enunciado label

11.4.3 El enunciado goto

El enunciado **goto** se representará mediante el símbolo goto (saltar a). Deberá ubicarse al final de un ejemplar de componente o al final de un operando en un símbolo expresión en línea. La figura 23 ilustra un ejemplo sencillo de enunciado **goto**.



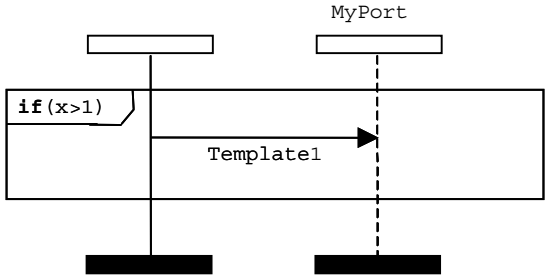

```
goto MyLabel;
```

GFT Núcleo

Figura 23/Z.142 – Enunciado goto

11.4.4 El enunciado if-else

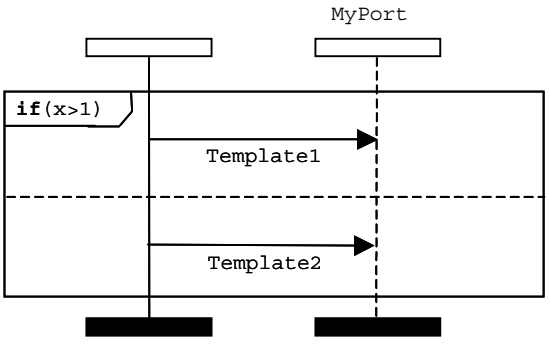
El enunciado **if-else** se representará mediante un símbolo expresión en línea etiquetado con la palabra clave **`if`** y una expresión booleana como se define en 19.6/Z.140 [1]. El símbolo expresión en línea if-else puede contener uno o dos operandos, separados mediante una línea discontinua. La figura 24 ilustra un ejemplo de enunciado **`if`** con un solo operando, que se ejecuta cuando la expresión booleana `x>1` es verdadera. La figura 25 ilustra un ejemplo de enunciado **if-else** en el que el operando superior se ejecuta cuando la expresión booleana `x>1` es verdadera y el operando inferior se ejecuta cuando dicha expresión es falsa.




```
if (x>1) {
  MyPort.send(Template1)
}
```

GFT Núcleo

Figura 24/Z.142 – Enunciado if




```
if (x>1) {
  MyPort.send(Template1)
}
else {
  MyPort.send(Template2)
}
```

GFT Núcleo

Figura 25/Z.142 – Enunciado if-else

11.4.5 El enunciado for

El enunciado **for** se representará mediante un símbolo expresión en línea etiquetado con la palabra clave **`for`** como se define en 19.7/Z.140 [1]. El contenido **for** se representará como el operando de un símbolo expresión en línea for. La figura 26 ilustra un ejemplo de bucle **for** sencillo en el que la variable de bucle se declara e inicializa dentro del enunciado **`for`**.

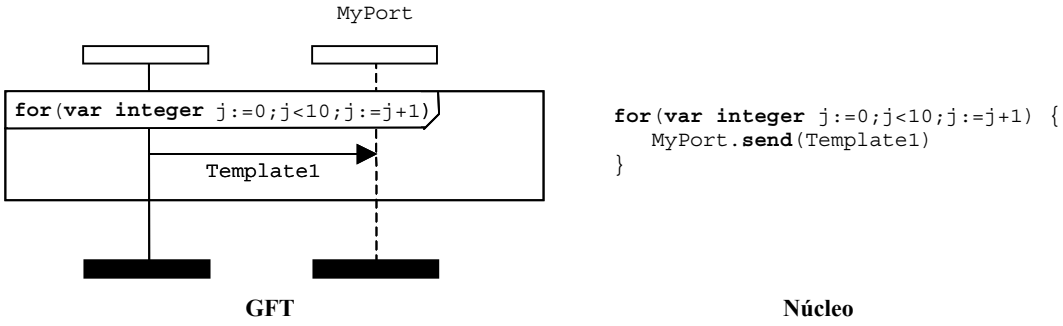


Figura 26/Z.142 – Enunciado for

11.4.6 El enunciado while

El símbolo **`while`** se representará mediante un símbolo expresión en línea etiquetado con la palabra clave **`while`** como se define en 19.8/Z.140 [1]. El contenido de **`while`** se representará como el operando en el símbolo expresión en línea while. La figura 27 ilustra un ejemplo de enunciado **`while`**.

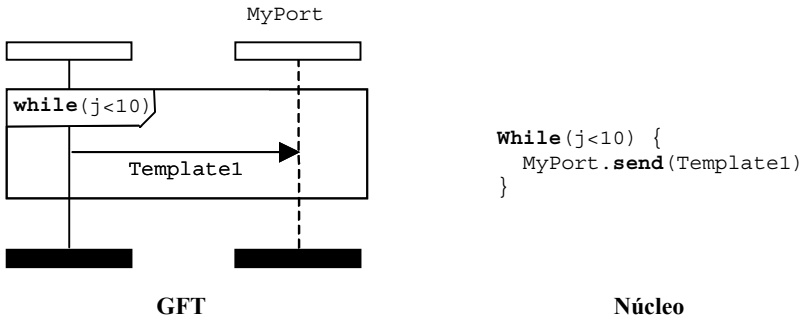


Figura 27/Z.142 – Enunciado while

11.4.7 El enunciado do-while

El enunciado **`do-while`** se representará mediante un símbolo expresión en línea etiquetado con la palabra clave **`do-while`** como se define en 19.9/Z.140 [1]. El contenido de **`do-while`** se representará como el operando en el símbolo expresión en línea **`do-while`**. La figura 28 ilustra un ejemplo de enunciado do-while.

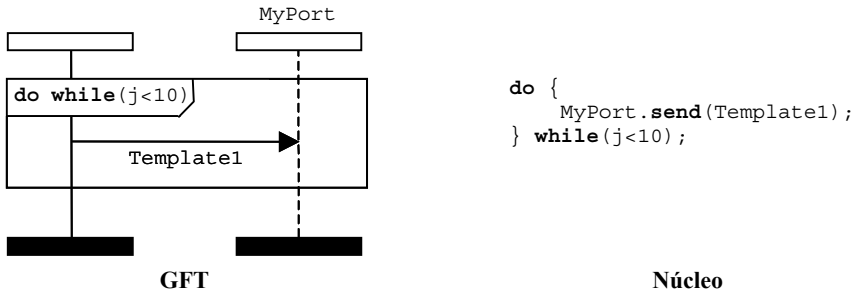


Figura 28/Z.142 – El enunciado do-while

11.5 Los enunciados de comportamiento del programa

Los enunciados de comportamiento pueden utilizarse dentro de los casos de prueba, funciones y alternativas, con la única excepción del enunciado return, que sólo puede utilizarse dentro de funciones. El comportamiento puede expresarse secuencialmente, como un conjunto de alternativas o mediante un enunciado interleaving. Los enunciados return y repeat se utilizan para el control del flujo del comportamiento.

11.5.1 Comportamiento secuencial

El comportamiento secuencial se representa mediante el orden de eventos situados en un ejemplar del componente de prueba. La ordenación de eventos es de arriba a abajo, de modo que los componentes que figuran más arriba en el símbolo del ejemplar del componente se ejecutan primero. En la figura 29 se ilustra un ejemplo sencillo en el que el componente de prueba ejecuta en primer lugar la expresión que figura dentro del símbolo acción y luego envía un mensaje al puerto `MyPort`.

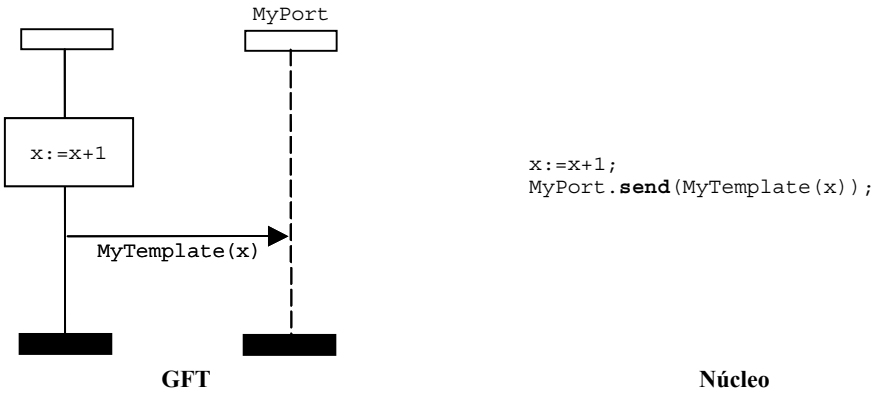


Figura 29/Z.142 – Comportamiento secuencial

La secuencia también puede describirse utilizando referencias a casos de prueba, funciones y alternativas. En este caso, el orden en el que aparecen las referencias en el eje de un ejemplar del componente determina el orden de ejecución. En la figura 30 se representa un diagrama GFT sencillo en el que primero se llama la función `MyFunction1` y después a `MyFunction2`.

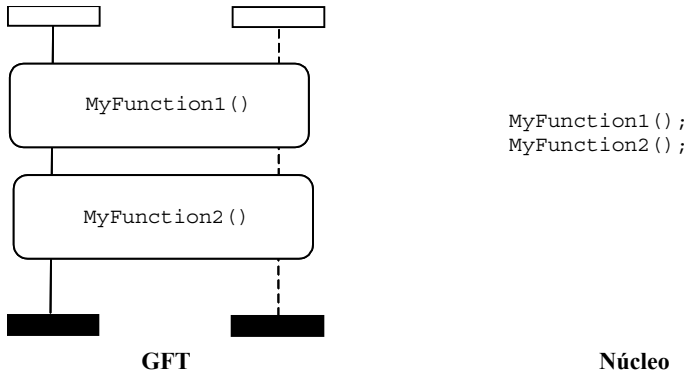


Figura 30/Z.142 – Secuenciación mediante referencias

11.5.2 Comportamiento con alternativas

El comportamiento con alternativas se representa utilizando el símbolo expresión en línea con la palabra clave **`alt`** ubicada en la esquina superior izquierda. Cada operando en la alternativa debe estar separado mediante una línea discontinua. Los operandos se comprueban de arriba a abajo.

Obsérvese que si se utilizan operadores de comunicación, la expresión en línea con alternativas debe abarcar siempre todos los ejemplares del puerto. En la figura 31 se ilustra un comportamiento con alternativas en el que se recibe un evento o mensaje con el valor definido por `Template1`, o se recibe el evento mensaje con el valor definido mediante `Template2`. En la figura 32 se muestra la llamada a una alternativa en una expresión en línea con alternativas.

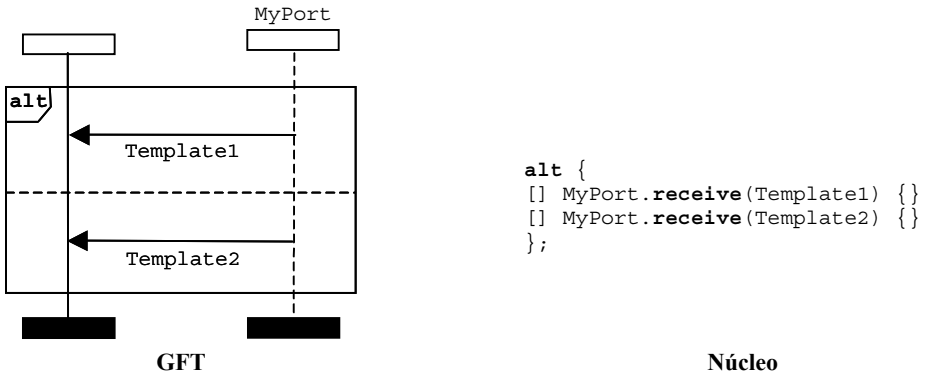


Figura 31/Z.142 – Proposición de comportamiento con alternativas

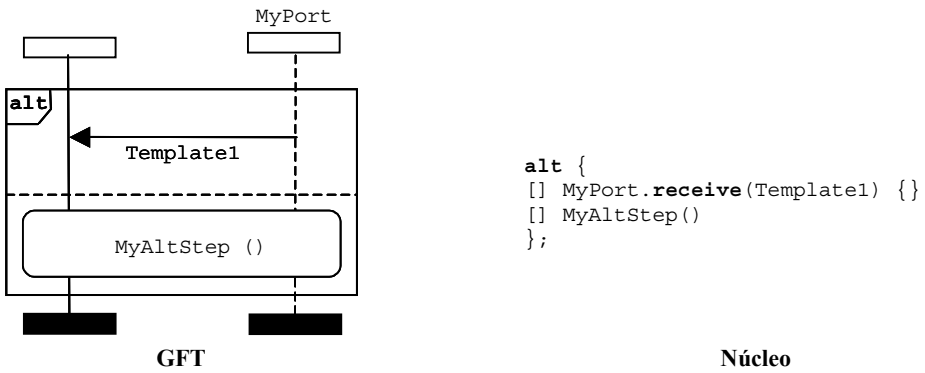
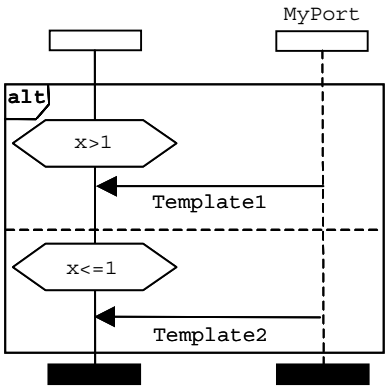


Figura 32/Z.142 – Comportamiento con alternativas con una llamada a una alternativa

Por otra parte, es posible llamar a una alternativa con sólo un evento dentro del operando de alternativa. Esto se dibuja utilizando un símbolo referencia (véase 11.2.3).

11.5.2.1 Selección/deselección de una alternativa

Es posible habilitar/deshabilitar un operando de alternativa mediante una expresión booleana dentro de un símbolo condición ubicado en el ejemplar del componente de prueba. En la figura 33 se ilustra un ejemplo sencillo de proposición de alternativa en el que el primer operando se compara con la expresión `x>1,` y el segundo con la expresión `x<=1`.



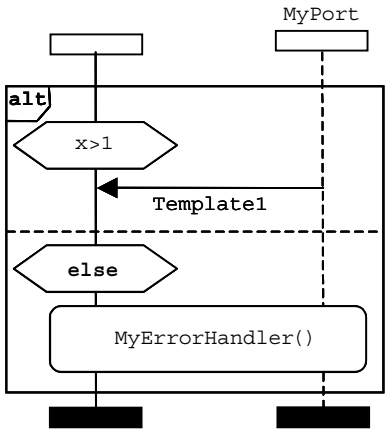

```
alt {
[x>1]  MyPort.receive(Template1) {}
[x<=1] MyPort.receive(Template2) {}
};
```

Núcleo

Figura 33/Z.142 – Selección/deselección de una alternativa

11.5.2.2 El enunciado else en alternativas

El enunciado **`else`** se indica mediante un símbolo de condición ubicado en el eje del ejemplar del componente de prueba etiquetado con la palabra clave **`else`**. En la figura 34 se ilustra un ejemplo sencillo de enunciado de alternativa en el que el segundo operando representa el enunciado **`else`**.




```
alt {
[x>1] MyPort.receive(Template1) {}
[else]MyErrorHandler()
};
```

Núcleo

Figura 34/Z.142 – El enunciado else dentro de una alternativa

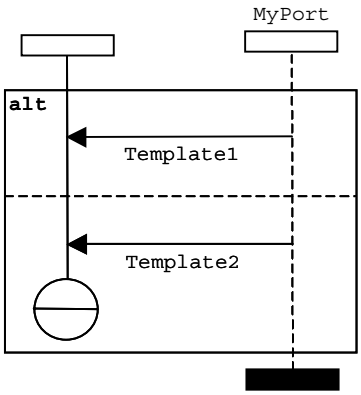
Obsérvese que, si se utilizan operaciones de comunicación, el símbolo de referencia dentro del enunciado else debe abarcar siempre todos los ejemplares del puerto.

La revaluación de un enunciado de alternativa puede especificarse utilizando un enunciado repeat, que se representa mediante el símbolo repetir (véase 11.5.3).

La invocación a alternativas dentro de alternativas se representa utilizando el símbolo referencia (véase 11.2.3).

11.5.3 El enunciado repeat

El enunciado **repeat** se representa mediante un símbolo repetir. Este símbolo se utilizará únicamente como el último evento de un operando de alternativas en un enunciado **`alt`** o como el último evento de un operando de la alternativa superior de una definición de alternativas. En la figura 35 se ilustra un ejemplo de enunciado de alternativas en la que el segundo operando, después de haber recibido satisfactoriamente un mensaje con un valor que corresponde `Template2`, causa la repetición de la alternativa.



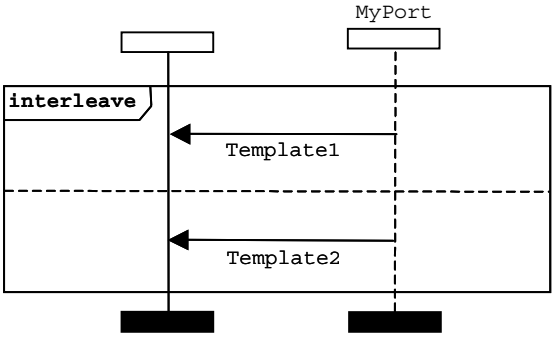

```
alt {
[] MyPort.receive(Template1) {}
[] MyPort.receive(Template2) { repeat; }
};
```

GFT **Núcleo**

Figura 35/Z.142 – Repetición dentro de una alternativa

11.5.4 Comportamiento entrelazado

El comportamiento entrelazado se representa utilizando un símbolo de expresión en línea con la palabra clave **interleave** ubicada en la esquina superior izquierda (véase la figura 36). Cada operando debe estar separado mediante líneas discontinuas. Los operandos se comprueban de arriba abajo.




```
interleave {
[] MyPort.receive(Template1) {}
[] MyPort.receive(Template2) {}
};
```

GFT **Núcleo**

Figura 36/Z.142 – Enunciado interleave

NOTA – La expresión en línea interleave debe abarcar siempre todos los ejemplares del puerto, si se utilizan operadores de comunicación.

11.5.5 El enunciado return

El enunciado **return** se representa mediante el símbolo volver. Este enunciado puede tener asociado un valor de retorno. El símbolo volver sólo debe utilizarse en un diagrama de función GFT, y figurará únicamente como el último evento de un ejemplar de componente o el último evento de un operando en el símbolo expresión en línea. En la figura 37 se ilustra una función sencilla que utiliza un enunciado return sin devolver un valor, y en la figura 38 se ilustra el ejemplo de una función que devuelve un valor.

Figura 37/Z.142 – Símbolo volver sin un valor de retorno

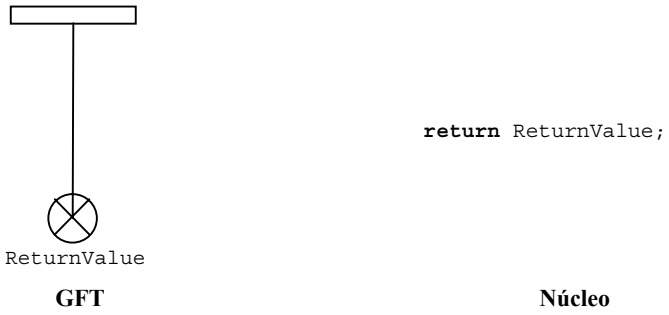


Figura 38/Z.142 –Símbolo volver con un valor de retorno

11.6 Tratamiento de valores por defecto

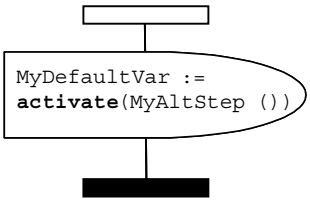
El GFT proporciona una representación gráfica para la activación y desactivación de valores por defecto (véase 21/Z.140 [1]).

11.6.1 Referencias a valores por defecto

Las variables de tipo **default** pueden declararse dentro de un símbolo acción o de un símbolo valor por defecto como parte de un enunciado activar. Las cláusulas 11.3.1 y 11.3.3 ilustran cómo se declaran una variable denominada `MyDefaultType` dentro del GFT.

11.6.2 La operación activate

La activación de valores por defecto se representa mediante la ubicación del enunciado **activate** dentro del símbolo default (véase la figura 39).



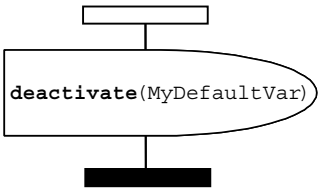

```
MyDefaultVar:=activate(MyAltStep());
```

GFT **Núcleo**

Figura 39/Z.142 – Activación de valores por defecto

11.6.3 Operación deactivate

La desactivación de valores por defecto se representa mediante el enunciado **deactivate** dentro del símbolo valor por defecto (véase la figura 40). Si no se indican operandos en el enunciado **deactivate**, se desactivan todos los valores por defecto.




```
deactivate(MyDefaultVar);
```

GFT **Núcleo**

Figura 40/Z.142 – Desactivación de valores por defecto

11.7 Operaciones de configuración

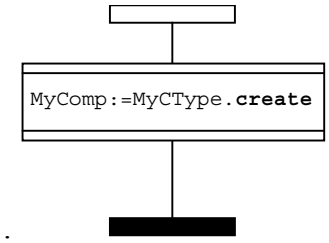
Las operaciones de configuración se utilizan para configurar y controlar los componentes de prueba. Estas operaciones sólo deben utilizarse en diagramas GFT caso de prueba, función y alternativa.

Las operaciones **mtc**, **self** y **system** no tienen representación gráfica; se indican mediante texto en los lugares donde se van a utilizar.

El GFT no proporciona representación gráfica alguna para la operación de ejecución (cuando se trata de una expresión booleana). Ésta se indica mediante texto en el lugar donde se va a utilizar.

11.7.1 La operación create

La operación **create** se representa mediante el símbolo crear, que se adjunta al ejemplar del componente de prueba que realiza la operación create (véase la figura 41). El símbolo crear contiene el enunciado create.



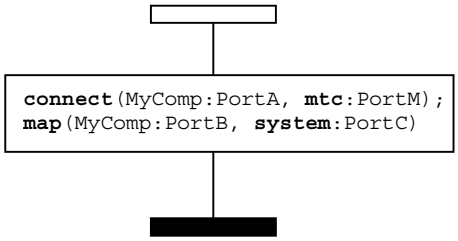

```
MyComp:=MyCType.create;
```

GFT Núcleo

Figura 41/Z.142 – Operación create

11.7.2 Las operaciones connect y map

Las operaciones **connect** y **map** se representarán mediante un símbolo recuadro de acción, que se une al ejemplar del componente de prueba que realiza la operación **connect** o **map** (véase la figura 42). El símbolo del recuadro de acción contiene el enunciado **connect** o **map**.



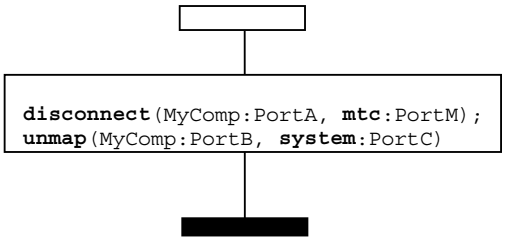

```
connect(MyComp:PortA, mtc:PortM);
map(MyComp:PortB, system:PortC);
```

GFT Núcleo

Figura 42/Z.142 – Operación connect y map

11.7.3 Las operaciones disconnect y unmap

Las operaciones **disconnect** y **unmap** se representarán mediante un símbolo recuadro de acción que se une al ejemplar del componente de prueba que realiza la operación **disconnect** o **unmap** (véase la figura 43). El símbolo recuadro de acción contiene el enunciado **disconnect** o **unmap**.



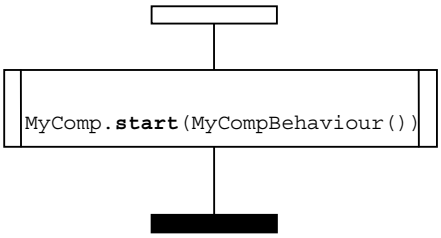

```
disconnect(MyComp:PortA, mtc:PortM);
unmap(MyComp:PortB, system:PortC);
```

GFT Núcleo

Figura 43/Z.142 – Operación disconnect y unmap

11.7.4 La operación start (iniciar) componente de prueba

La operación **start** componente de prueba se representará dentro del símbolo **iniciar**, que se une al ejemplar del componente de prueba que realiza la operación **start** (véase la figura 44). El símbolo iniciar contiene el enunciado **start**.




```
MyComp.start(MyCompBehaviour());
```

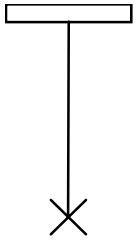
GFT **Núcleo**

Figura 44/Z.142 – Operación start

11.7.5 Las operaciones stop (detener) ejecución y stop (detener) componente de prueba

El TTCN-3 tiene dos operaciones detener: el control de módulo y los componentes de prueba pueden detenerse por sí mismos utilizando *detener operaciones de ejecución*, o bien el componente de prueba puede detener otros componentes mediante la utilización de *detener operaciones del componente de prueba*.

La operación **stop** de ejecución se representa mediante el símbolo detener, que se une al ejemplar del componente de prueba que realiza la operación **stop** de la ejecución (véase la figura 45). Esta operación figurará como el último evento de un ejemplar de componente o el último evento de un operando en un símbolo expresión en línea.



```
stop;
```

GFT **Núcleo**

Figura 45/Z.142 – Operación stop de la ejecución

La operación **stop** del componente de prueba se representa mediante un símbolo detener, que se une al ejemplar del componente de prueba que realiza la operación **stop** del componente de prueba. Esta operación tendrá asociada una expresión que identifica el componente que se desea detener (véase la figura 46). El MTC puede detener todos los PTC en una sola instrucción utilizando la operación stop del componente con la palabra clave **all** (véase figura 47 a)). El PTC puede detener la ejecución de la prueba deteniendo el MTC (véase la figura 47 b)). La operación de componente de prueba **stop** figurará como el último evento de un ejemplar de componente o como último evento de un operando en un símbolo expresión en línea, si el componente se detiene por sí mismo (por ejemplo, `self.stop`) o detiene la ejecución de la prueba (por ejemplo, `mtc.stop`) (véanse las figuras 47 c) y d)).

NOTA – El símbolo stop tiene una expresión asociada. No siempre es posible determinar estáticamente si una operación stop de componente detiene el ejemplar que ejecuta la operación stop o detiene la ejecución de la prueba.

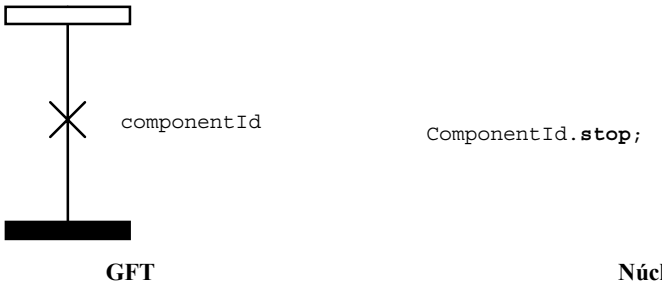


Figura 46/Z.142 – Operación stop componente de prueba

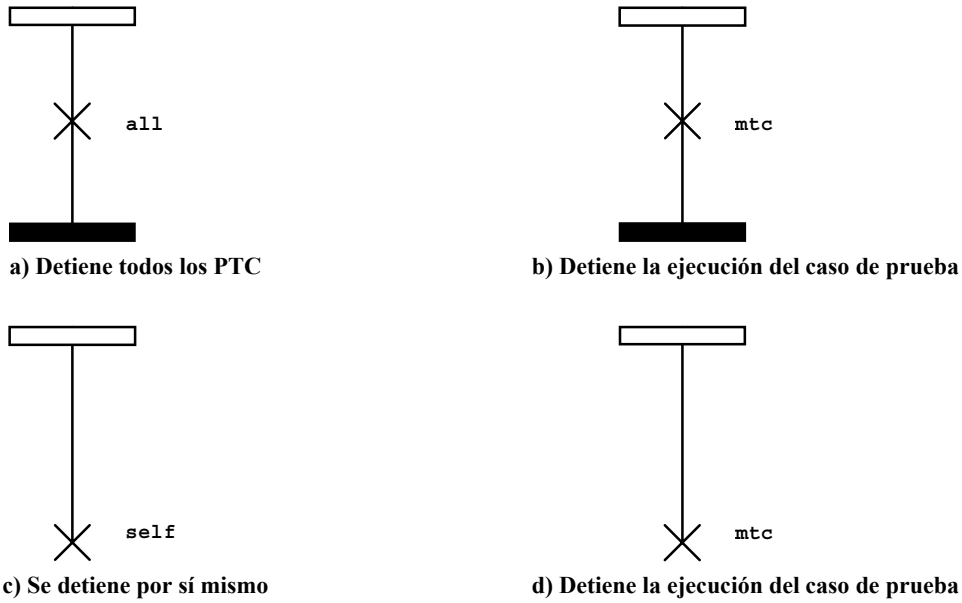


Figura 47/Z.142 – Casos especiales de utilización de la operación stop del componente de prueba

11.7.6 La operación done

La operación **done** (fin) se representa mediante un símbolo condición, que se une al ejemplar del componente de prueba que realiza la operación **done** (véase la figura 48). El símbolo de condición contiene el enunciado **done**.

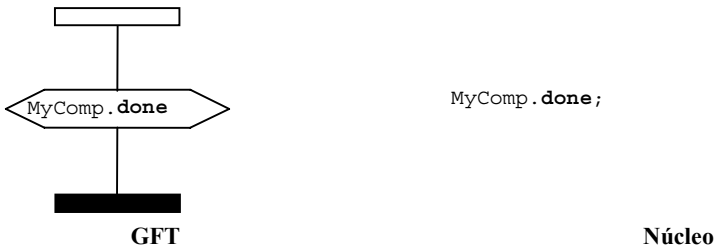


Figura 48/Z.142 – Operación done

Pueden utilizarse las palabras claves **any** y **all** para las operaciones **running** y **done** pero solamente desde un ejemplar MTC. No tienen una representación gráfica sino que se indica en palabras en los lugares donde se van a utilizar.

11.8 Operaciones de comunicación

Las operaciones de comunicación se dividen en dos grupos:

a) *operaciones de envío*: un componente de prueba envía un mensaje (operación **`send`**), llama a un procedimiento (operación **`call`**), responde a una llamada aceptada (operación **`reply`**) o genera una excepción (operación **`raise`**).

b) *operaciones de recepción*: un componente recibe un mensaje (operación **`receive`**), acepta una llamada a procedimiento (operación **`getcall`**), recibe una respuesta a un procedimiento llamado anteriormente (operación **`getreply`**) o acepta una excepción (operación **`catch`**).

11.8.1 Formato general de las operaciones de envío

Todas las operaciones de envío utilizan un símbolo mensaje que se traza desde el ejemplar del componente de prueba que realiza la operación de envío hasta el ejemplar de puerto a la que se transmite la información (véase la figura 49).

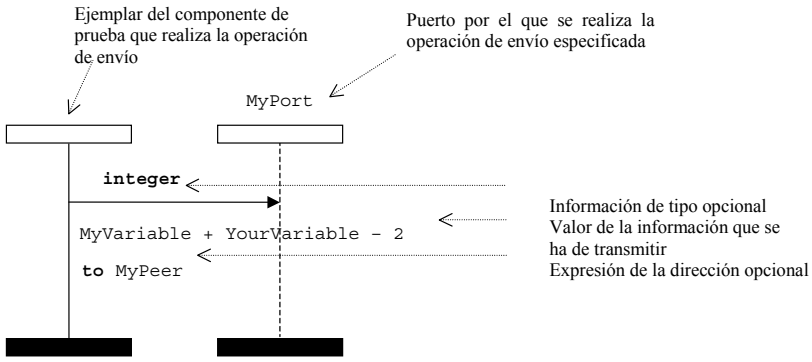


Figura 49/Z.142 – Formato general de las operaciones de envío

Las operaciones de envío constan de una parte *send* (envía) y en el caso de una operación **`call`** basada en un procedimiento en bloque, una parte *response* (respuesta) y otra *exception handling* (tratamiento de excepciones).

La parte envío:

- especifica el puerto por el que se realiza la operación especificada;
- define el tipo opcional y el valor de la información que se ha de transmitir;
- indica una dirección opcional que identifica inequívocamente el homólogo de comunicación en el caso de una conexión uno a muchos.

El puerto estará representado mediante un ejemplar de puerto. El nombre de la operación para las operaciones **call**, **reply**, y **raise** se indicarán en la parte superior del símbolo del mensaje a la altura de la información de tipo opcional. La operación **send** es implícita, es decir, la palabra clave **send** no tiene que indicarse. El valor de la información a transmitir figurará debajo del símbolo de mensaje. La expresión de la dirección opcional (indicado mediante la palabra clave **to**) figurará debajo del valor de la información que se ha de transmitir.

La estructura de la operación **call** es más específica. Para más información, véase 11.8.4.1.

11.8.2 Formato general de las operaciones de recepción

Todas las operaciones de recepción utilizan un símbolo mensaje que se traza desde el ejemplar del puerto hasta el ejemplar del componente de prueba que recibe la información (véase la figura 50).

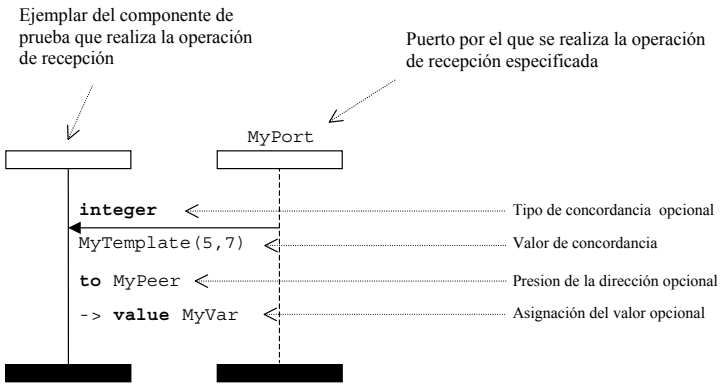


Figura 50/Z.142 – Formato general de las operaciones de recepción con la asignación de dirección y valor

Una operación de recepción consiste en una parte *receive* (recibe) y una parte opcional *assignment* (asignación).

La parte recibe:

a) especifica el puerto por el que se realiza la operación;

b) define una parte de concordancia que consiste en la información del tipo opcional y el valor de concordancia que especifica la entrada aceptable con la que concuerda el enunciado;

c) indica una expresión de dirección (opcional) que identifica inequívocamente el homólogo de comunicación (en caso de las conexiones uno a muchos).

El puerto estará representado mediante un ejemplar de puerto. El nombre de la operación para las operaciones **getcall**, **getreply**, y **catch** se indicarán en la parte superior del símbolo del mensaje a la altura de la información de tipo (opcional). La operación **receive** es implícita, es decir, no es necesario indicar la palabra clave **receive**. El valor de concordancia para la entrada aceptable se ubicará debajo del símbolo de mensaje. La expresión de dirección (opcional) (indicada mediante la palabra clave **from**) se ubicará debajo del valor de la información que se ha de transmitir.

La parte de asignación (opcional) (que se indica mediante el símbolo **"->"**) se ubicará debajo del valor de la información que se ha de transmitir o debajo de la expresión de dirección, si la hubiere. Esta parte puede constar de varias líneas, por ejemplo es posible escribir el valor, el parámetro y la asignación del emisor en líneas separadas (véase la figura 51).

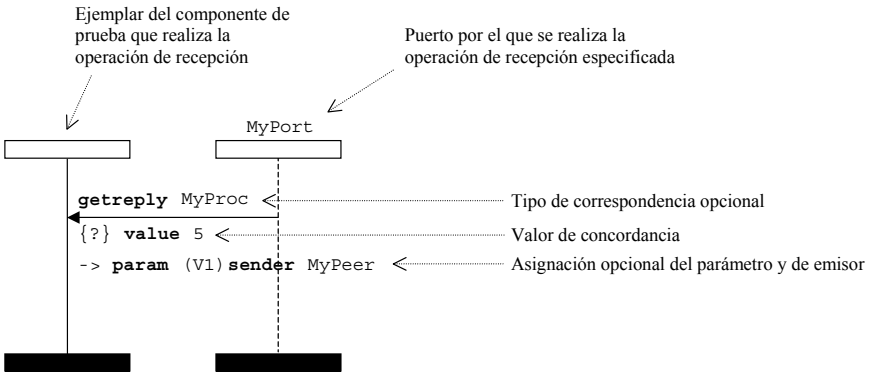
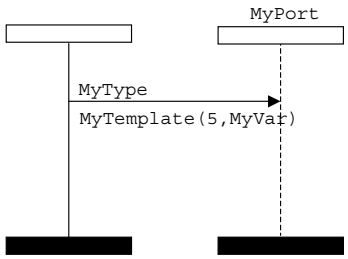


Figura 51/Z.142 – Formato general de las operaciones de recepción con asignación de parámetro y de emisor

11.8.3 Comunicación basada en mensajes

11.8.3.1 La operación send

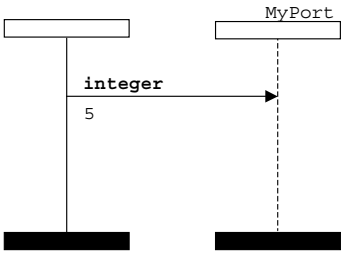
La operación **send** (enviar) se representa mediante un símbolo de mensaje que sale del componente de prueba y llega hasta el ejemplar del puerto. La información de tipo opcional figurará encima de la flecha del mensaje. La plantilla (en línea) se ubicará debajo de la flecha del mensaje (véanse las figuras 52 y 53).




```
MyPort.send(MyType: MyTemplate(5,MyVar));
```

Núcleo

Figura 52/Z.142 – Operación send con referencia a la plantilla



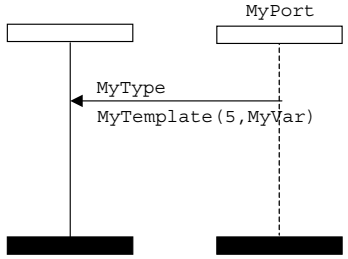

```
MyPort.send(integer:5);
```

Núcleo

Figura 53/Z.142 – Operación send con plantilla en línea

11.8.3.2 La operación receive

La operación **receive** (recibir) se representará mediante un símbolo del mensaje que apunta al ejemplar del puerto procedente del componente de prueba. La información de tipo opcional se ubicará encima de la flecha del mensaje. La plantilla (en línea) se ubicará debajo de la flecha del mensaje (véanse las figuras 54 y 55).




```
MyPort.receive(MyType: MyTemplate(5, MyVar));
```

Núcleo

Figura 54/Z.142 – Operación receive con referencia de plantilla

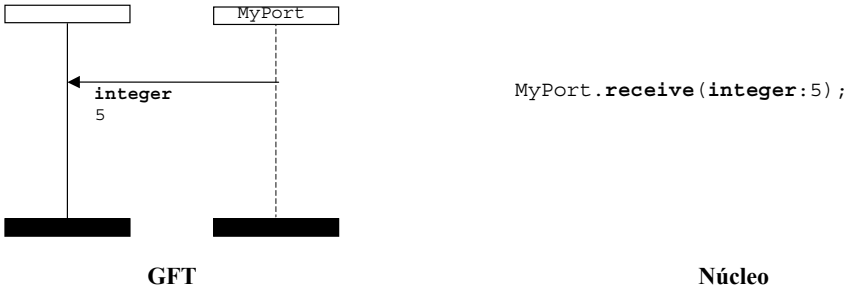


Figura 55/Z.142 – Operación receive con plantilla en línea

11.8.3.2.1 Operación receive any message

La operación recepción de cualquier mensaje (***receive any message***) se representará mediante un símbolo mensaje que apunta al ejemplar del puerto procedente del componente de prueba sin añadir ninguna información adicional (véase la figura 56).

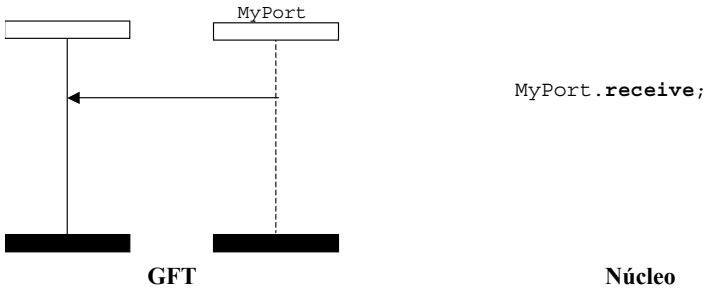


Figura 56/Z.142 – Operación mensaje receive any

11.8.3.2.2 Operación receive on any part

La operación recepción desde cualquier puerto (receive on any part) se representará mediante un símbolo hallado que representa cualquier puerto conectado al componente de prueba (véase la figura 57).

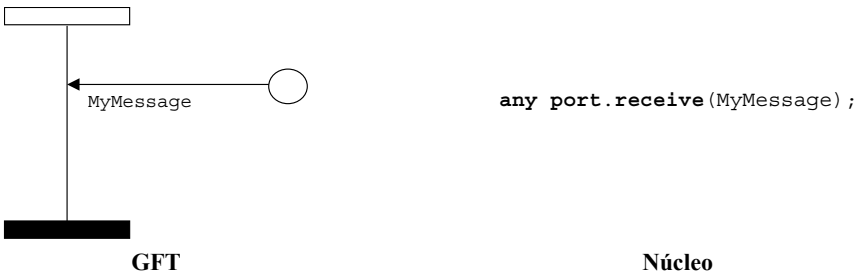


Figura 57/Z.142 – Operación receive on any port

11.8.3.3 Operación trigger

La operación trigger se representará mediante un símbolo mensaje que sale del ejemplar del puerto y apunta al componente de prueba; la palabra clave **trigger** deberá figurar encima de la flecha del mensaje que precede la información del tipo, si la hubiera. La información de tipo opcional se ubica

encima de la flecha mensaje subsiguiente a la palabra clave **trigger**. La plantilla (en línea) se ubica debajo de la flecha del mensaje (véanse las figuras 58 y 59).

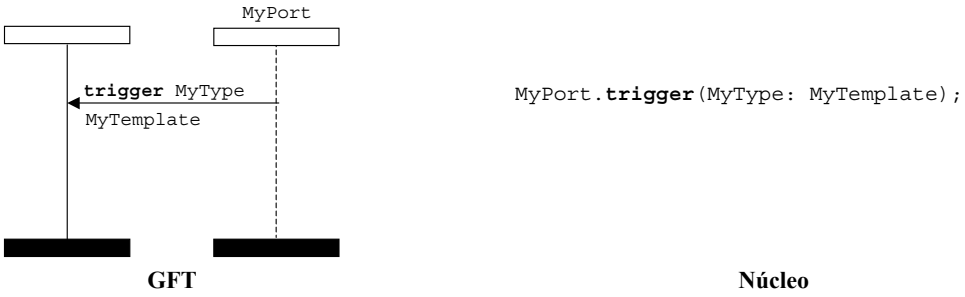


Figura 58/Z.142 – Operación trigger con referencia a plantilla

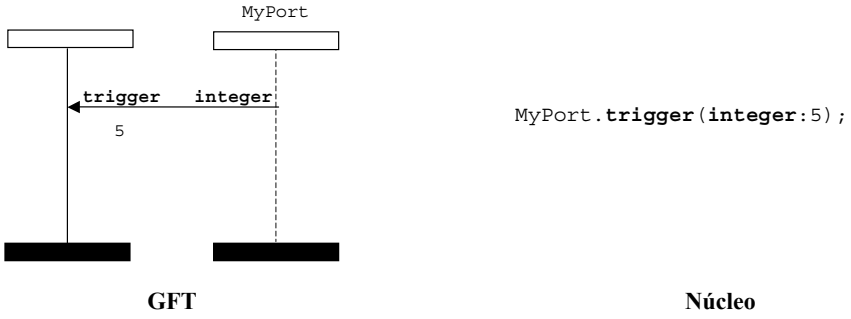


Figura 59/Z.142 – Operación trigger con plantilla en línea

11.8.3.3.1 Operación trigger on any message

La operación activar lectura por cualquier mensaje (trigger on any message) se representará mediante un símbolo mensaje que sale desde el ejemplar del puerto y apunta al componente de prueba; la palabra clave **trigger** deberá figurar encima de la flecha del mensaje sin añadir ninguna información adicional (véase la figura 60).

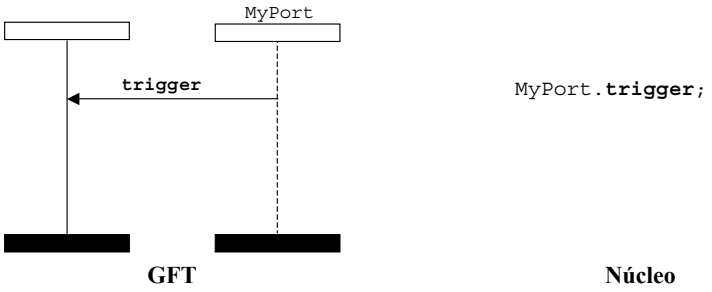


Figura 60/Z.142 – Operación trigger on any message

11.8.3.3.2 Operación trigger on any port

La operación activar lectura desde cualquier puerto (trigger on any port) se representará mediante un símbolo hallado que representa cualquier puerto conectado al componente de prueba (véase la figura 61).

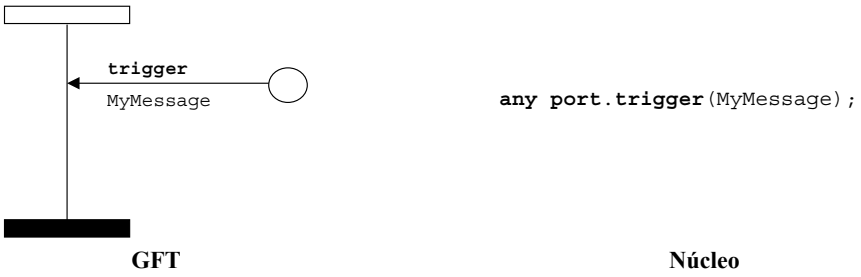


Figura 61/Z.142 – Operación trigger on any port

11.8.4 Comunicación basada en procedimientos

11.8.4.1 La operación call

11.8.4.1.1 Llamada a procedimientos con bloqueo

La operación **call** con bloqueo se representa mediante un símbolo mensaje que sale del componente de prueba y llega hasta el ejemplar del puerto con una subsiguiente región de suspensión sobre el componente de prueba; la palabra clave **call** deberá figurar encima de la flecha del mensaje que precede a la signatura, si la hubiere. La plantilla (en línea) se ubica debajo de la flecha del mensaje (véanse las figuras 62 y 63).

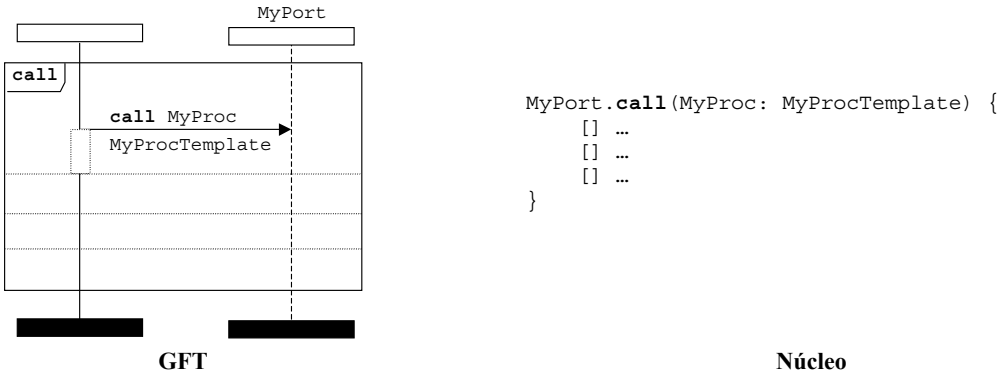


Figura 62/Z.142 – Operación llamada con bloqueo con referencia a plantilla

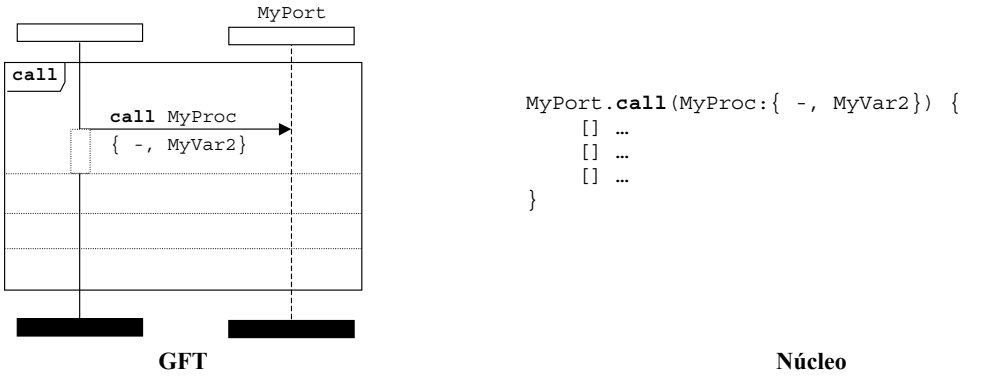
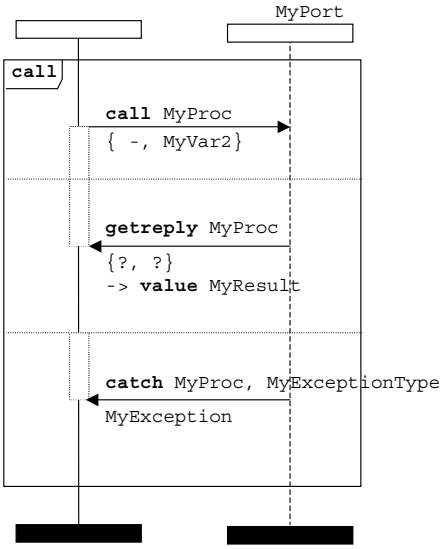


Figura 63/Z.142 – Operación llamada con bloqueo con plantilla en línea

La expresión en línea de llamada se introduce para facilitar la especificación de alternativas a las posibles respuestas de la operación de llamada con bloqueo. Después de la operación de llamada

podrán figurar alternativas de getreply, catch y timeout. Las respuestas a una llamada se especifican dentro de la expresión en línea de llamada que figura después de la operación de llamada separada por líneas discontinuas (véase la figura 64).

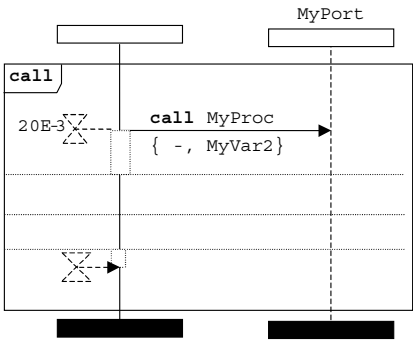



```
MyPort.call(MyProc:{ -, MyVar2}) {
    [] MyPort.getreply(MyProc:{?, ?})
       -> value MyResult { }
    [] MyPort.catch
       (MyProc, MyExceptionType: MyException) { }
}
```

Núcleo

Figura 64/Z.142 – Operación llamada con bloqueo seguida de alternativas de getreply y catch

La operación de llamada puede incluir como opción una temporización. Para ello se utiliza el símbolo de inicio de temporizador implícito para iniciar este periodo de temporización. El símbolo expiración de temporizador implícita se utiliza para representar la excepción de temporización (véase la figura 65).




```
MyPort.call(MyProc:{ -, MyVar2},20E-3) {
    [] …
    [] …
    [] MyPort.catch(timeout) {
       …
       }
}
```

Núcleo

Figura 65/Z.142 – Operación llamada con bloqueo seguida de una excepción de temporización

11.8.4.1.2 Llamada a procedimientos sin bloqueo

La operación de llamada sin bloqueo se representará mediante un símbolo mensaje que sale del componente de prueba y llega al puerto; la palabra clave **call** deberá figurar arriba de la flecha del mensaje que precede a la signatura. No habrá un símbolo región de suspensión adjunto al símbolo mensaje. La signatura opcional se representa encima de la flecha del mensaje. La plantilla (en línea) se ubica debajo de la flecha del mensaje (véanse las figuras 66 y 67).

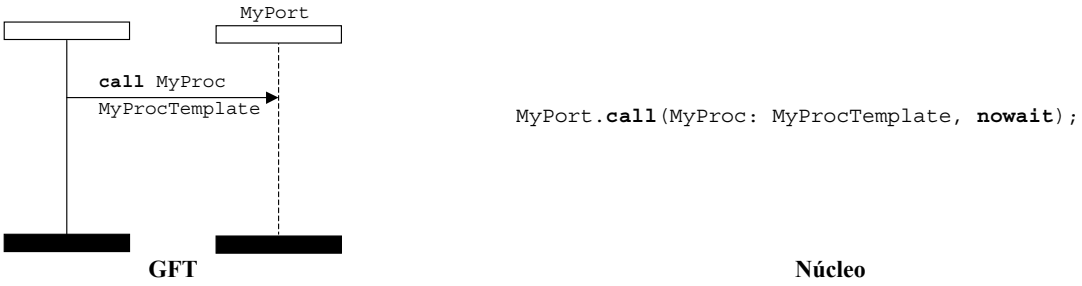


Figura 66/Z.142 – Operación de llamada sin bloqueo con referencia a plantilla

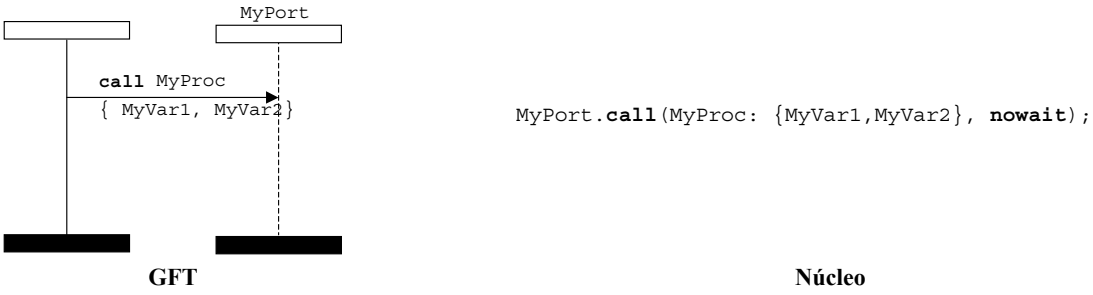


Figura 67/Z.142 – Operación de llamada sin bloqueo con plantilla en línea

11.8.4.2 La operación getcall

La operación aceptar llamada (getcall) se representa mediante un símbolo mensaje que va desde el ejemplar del puerto hasta el componente de prueba; la palabra clave **getcall** deberá figurar arriba de la flecha del mensaje que precede a la signatura. La signatura se ubica encima de la flecha del mensaje subsiguiente a la palabra clave **getcall**. La plantilla (en línea) se ubica debajo de la flecha del mensaje (véanse las figuras 68 y 69).

Figura 68/Z.142 – Operación getcall con referencia a plantilla

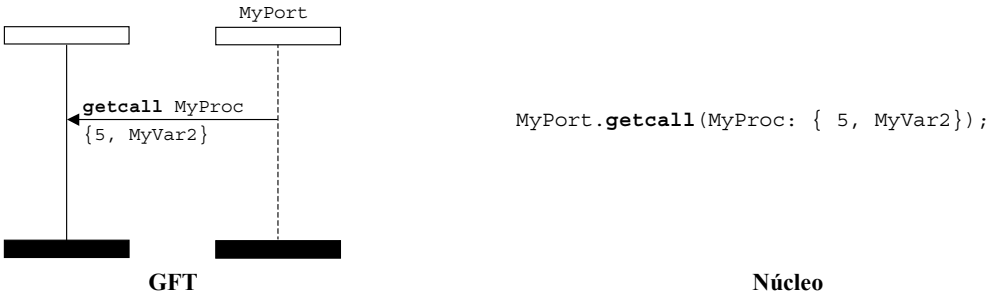


Figura 69/Z.142 – Operación getcall con plantilla en línea

11.8.4.2.1 Operación accepting any call

La operación aceptar cualquier llamada (accepting any call) se representará mediante un símbolo mensaje que va desde el ejemplar del puerto hasta el componente de prueba; la palabra clave **getcall** deberá figurar encima de la flecha del mensaje. No se añadirá información adicional al símbolo mensaje (véase la figura 70).

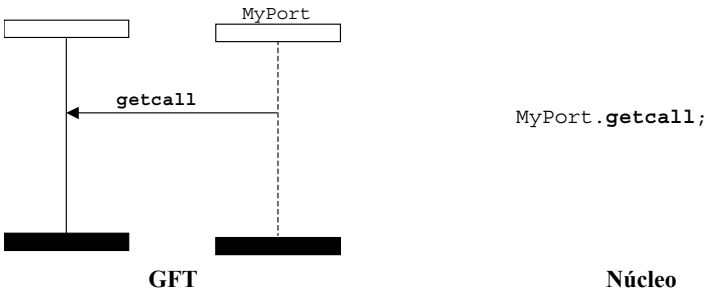


Figura 70/Z.142 – Operación getcall on any call

11.8.4.2.2 Operación getcall on any port

La operación aceptar llamada desde cualquier puerto (getcall on any port) se representa mediante un símbolo hallado (found) que representa cualquier puerto al que está conectado el componente de prueba; la palabra clave **getcall** deberá figurar encima de la flecha del mensaje que sigue a la signatura, si la hubiere. En la plantilla (en línea), si la hubiere, se ubicará debajo de la flecha del mensaje (véase la figura 71).

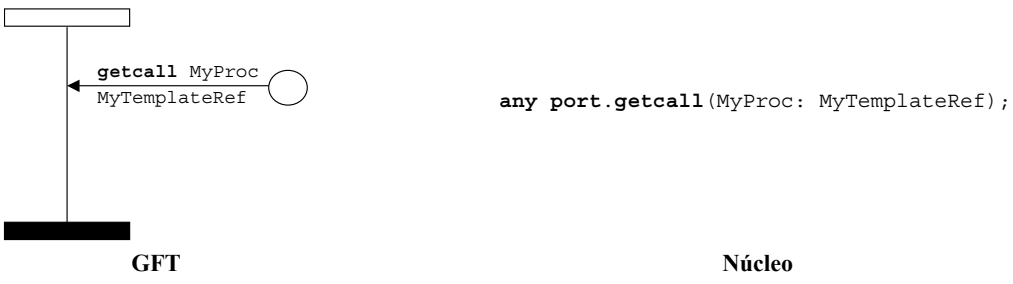


Figura 71/Z.142 – Operación getcall on any port con referencia a plantilla

11.8.4.3 La operación reply

La operación responder (reply) se representa mediante un símbolo mensaje que sale del componente de prueba y llega hasta el ejemplar del puerto; la palabra clave **reply** deberá figurar encima de la flecha de mensaje que precede a la signatura. La signatura deberá situarse encima de la flecha del mensaje subsiguiente a la palabra clave **reply**. La plantilla (en línea) se situará debajo de la flecha del mensaje (véanse las figuras 72 y 73).

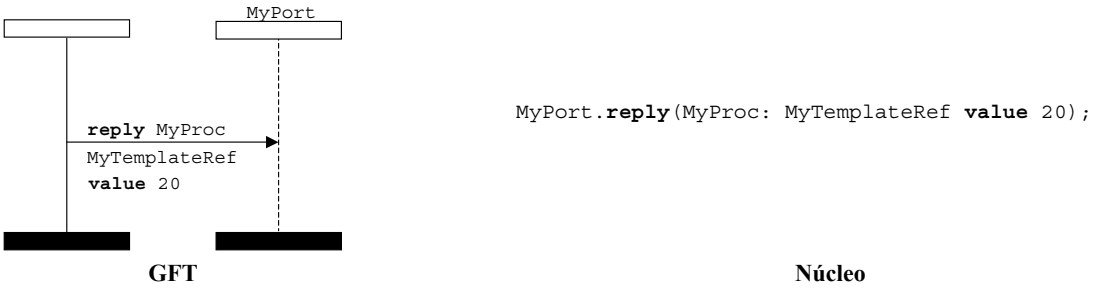


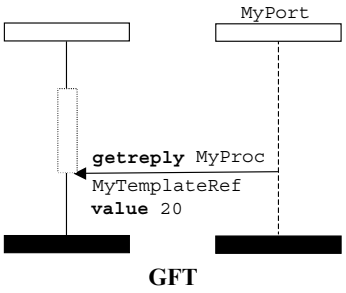
Figura 72/Z.142 – Operación reply con referencia de plantilla

Figura 73/Z.142 – Operación reply con plantilla en línea

11.8.4.4 La operación getreply

La operación aceptar respuesta (getreply) se representará mediante un símbolo mensaje que sale desde el ejemplar del puerto y llega hasta el componente de prueba; la palabra clave **getreply** figurará sobre la flecha del mensaje que precede a la signatura. Dentro del símbolo llamada, la punta de la flecha del mensaje tocará la parte inferior de la región de suspensión precedente sobre el componente de prueba (véanse las figuras 74 y 75). Fuera del símbolo de llamada, la punta de la flecha del mensaje no deberá conectarse a la región de suspensión precedente sobre el componente de prueba (véanse las figuras 76 y 77).

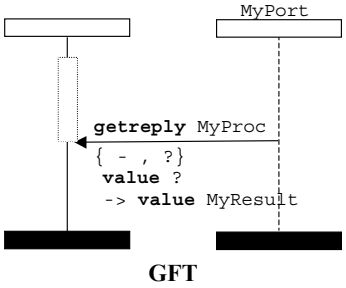
La signatura se situará encima de la flecha de mensaje subsiguiente a la palabra clave **getreply**. La plantilla (en línea) se ubicará debajo de la flecha del mensaje.




```
:
MyPort.getreply(MyProc: MyTemplateRef value 20);
:
```

Núcleo

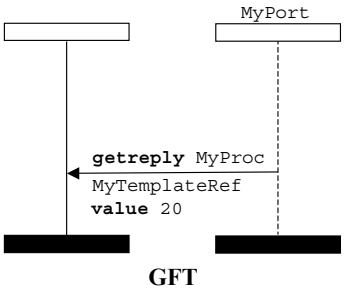
Figura 74/Z.142 – Operación getreply con referencia a plantilla (dentro de un símbolo llamada)




```
:
MyPort.getreply(MyProc: {-, ?}
value ?) -> value MyResult;
:
```

Núcleo

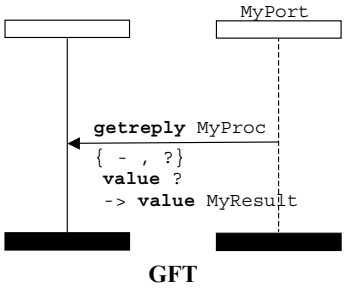
Figura 75/Z.142 – Operación getreply con plantilla en línea (dentro de un símbolo llamada)




```
MyPort.getreply(MyProc: MyTemplateRef value 20);
```

Núcleo

Figura 76/Z.142 – Operación getreply con referencia a plantilla (fuera del símbolo llamada)




```
MyPort.getreply(MyProc: {-, ?)}
value ?) -> value MyResult;
```

Núcleo

Figura 77/Z.142 – Operación getreply con plantilla en línea (fuera del símbolo llamada)

11.8.4.4.1 La operación get any reply from any call

La operación aceptar toda respuesta a cualquier llamada (get any reply from any call) se representará mediante un símbolo mensaje que sale del ejemplar del puerto y llega hasta el componente de prueba; la palabra clave **getreply** deberá figurar encima del mensaje. Después de la palabra clave **getreply** no figurará ninguna signatura.

Dentro del símbolo de llamada, la punta de la flecha del mensaje se conectará a la región de suspensión precedente sobre el componente de prueba (véase la figura 78). Fuera del símbolo de llamada, la punta de la flecha de mensaje no se conectará a la región de suspensión precedente sobre el componente de prueba (véase la figura 79).

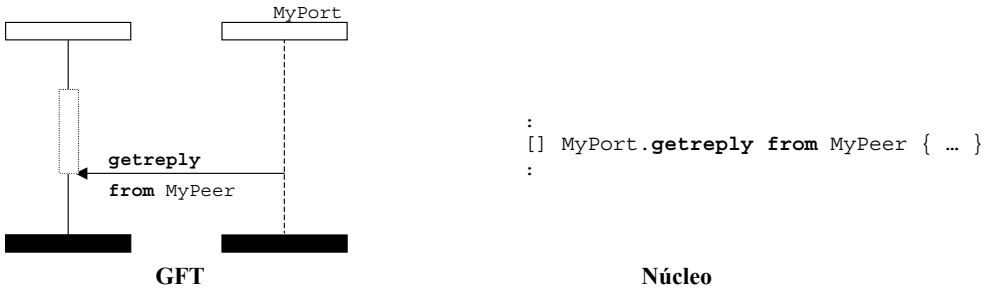


Figura 78/Z.142 – Operación get any reply from any call (dentro del símbolo de llamada)

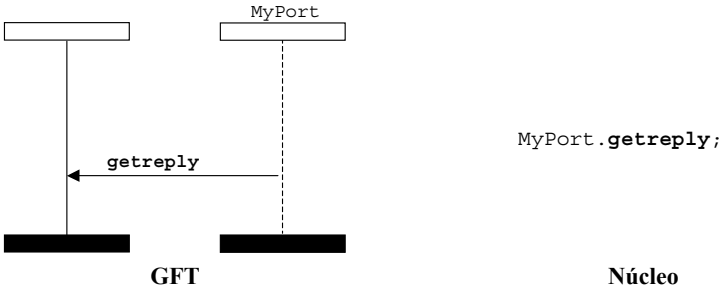


Figura 79/Z.142 – Operación getreply from any call (fuera del símbolo de llamada)

11.8.4.4.2 Operación get a reply on any port

La operación aceptar respuesta desde cualquier puerto (get a reply on any port) se representa mediante un símbolo hallado (found) que se refiere a cualquier puerto conectado al componente de prueba. La palabra clave **getreply** deberá figurar encima de la flecha del mensaje siguiente a la signatura, si la hubiere. Dentro del símbolo de llamada, la punta de la flecha del mensaje deberá conectarse a la región de suspensión precedente sobre el componente de prueba (véase la figura 80). Fuera del símbolo de llamada, la punta de la flecha del mensaje no deberá conectarse a la región de suspensión precedente sobre el componente de prueba (véase la figura 81).

La signatura, si la hubiere, deberá ubicarse encima de la flecha del mensaje subsiguiente a la palabra clave **getreply**. La plantilla (en línea) opcional se situará debajo de la flecha del mensaje.

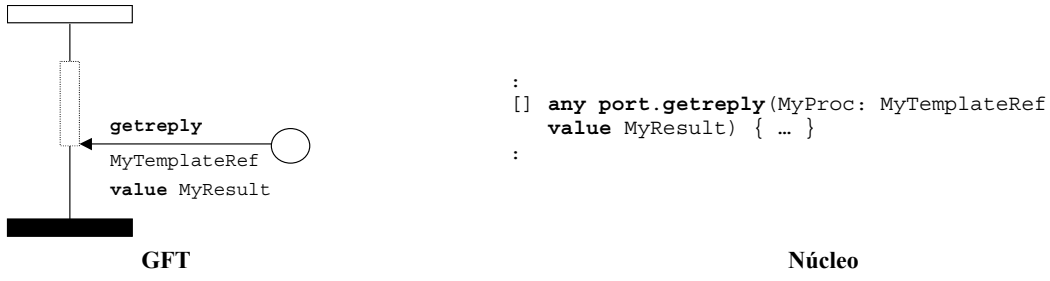


Figura 80/Z.142 – Operación get a reply on any port (dentro del símbolo de llamada)

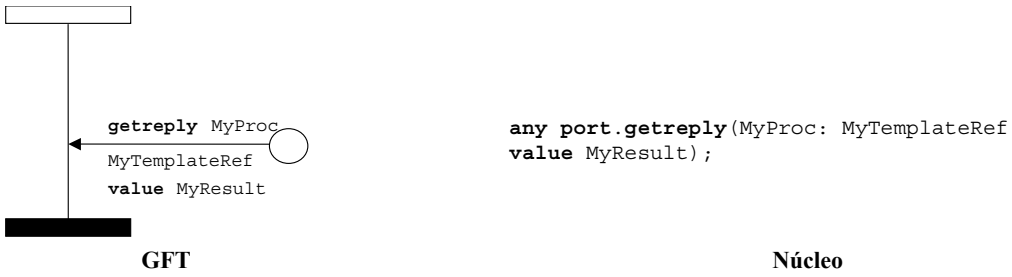


Figura 81/Z.142 – Operación get a reply on any port (fuera del símbolo de llamada)

11.8.4.5 La operación raise

La operación generar (raise) se representará mediante un símbolo mensaje que sale del componente de prueba y llega hasta el ejemplar de puerto. La palabra clave **raise** se situará encima de la flecha del mensaje que precede a la signatura y el tipo de excepción, separados por una coma. La plantilla (en línea) deberá situarse debajo de la flecha del mensaje (véanse las figuras 82 y 83).

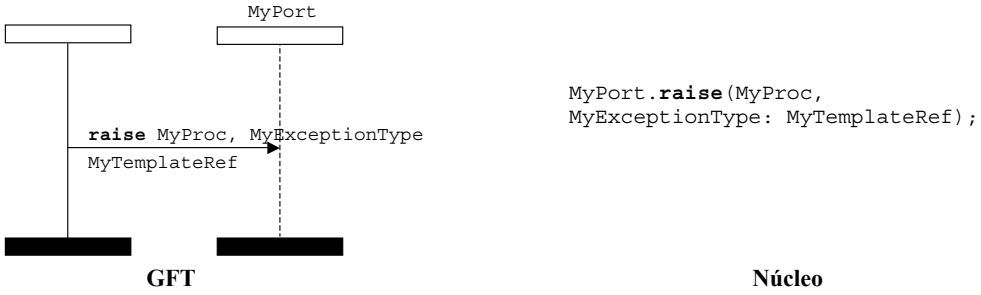


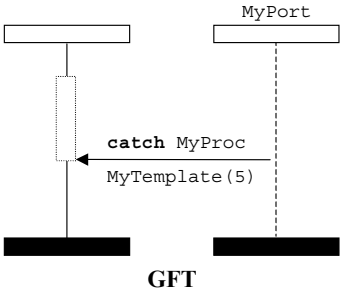
Figura 82/Z.142 – Operación raise con referencia a plantilla

Figura 83/Z.142 – Operación raise con plantilla en línea

11.8.4.6 La operación catch

La operación adquirir (catch) se representará mediante un símbolo mensaje que sale del ejemplar del puerto y llega hasta el componente de prueba; la palabra clave **catch** deberá situarse encima de la flecha del mensaje que precede a la signatura y el tipo de excepción (si lo hubiere). Dentro del símbolo de llamada, la punta de la flecha del mensaje se conectará a la región de suspensión precedente sobre el componente de prueba (véanse las figuras 84 y 85). Fuera del símbolo de llamada, la punta de la flecha del mensaje no se conectará a la región de suspensión precedente en el componente de prueba (véanse las figuras 86 y 87).

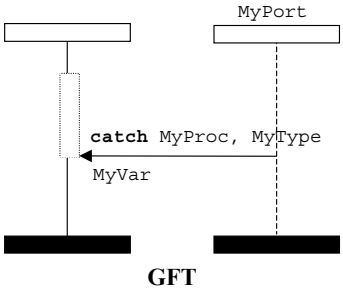
La signatura y la información de tipo de excepción opcional se ubicarán encima de la flecha del mensaje subsiguiente a la palabra clave **catch**, separada por coma, si el tipo de excepción está presente. La plantilla (en línea) se situará debajo de la flecha del mensaje.




```
:
[] MyPort.catch(MyProc, MyTemplate(5)) { … }
:
```

Núcleo

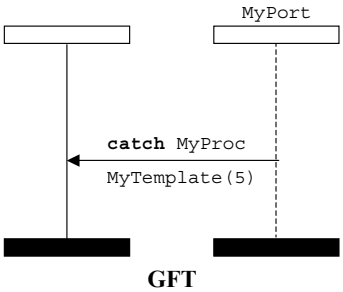
Figura 84/Z.142 – Operación catch con referencia a plantilla (dentro del símbolo de llamada)




```
:
[] MyPort.catch(MyProc, MyType: MyVar) { … }
:
```

Núcleo

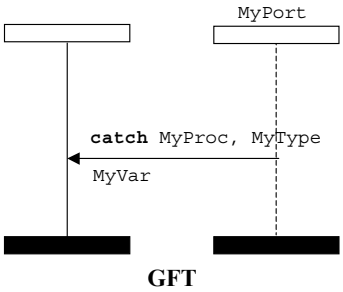
Figura 85/Z.142 – Operación catch con plantilla en línea (dentro del símbolo de llamada)




```
MyPort.catch(MyProc, MyTemplate(5));
```

Núcleo

Figura 86/Z.142 – Operación catch con referencia a plantilla (fuera del símbolo de llamada)




```
MyPort.catch(MyProc, MyType: MyVar);
```

Núcleo

Figura 87/Z.142 – Operación catch con plantilla en línea (fuera del símbolo de llamada)

11.8.4.6.1 La excepción temporización

La excepción temporización se representa mediante un símbolo temporización con la flecha conectada al componente de prueba (véase la figura 88). No debe añadirse información alguna al símbolo de temporización, y se utilizará dentro de un símbolo de llamada únicamente. La punta de la flecha del mensaje se conectará a las regiones de suspensión precedentes sobre el componente de prueba.

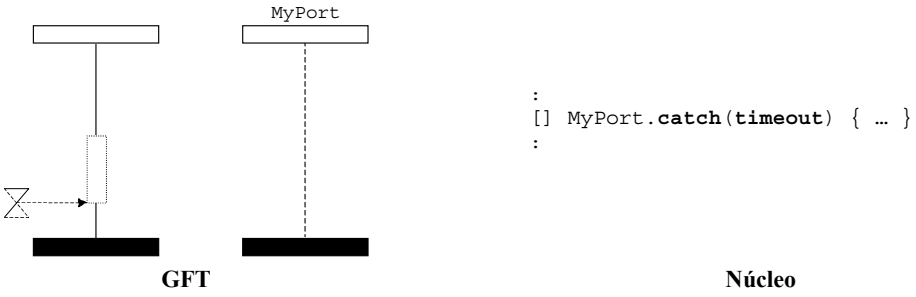


Figura 88/Z.142 – Excepción temporización (dentro de un símbolo de llamada)

11.8.4.6.2 Operación catch any exception

La operación adquirir cualquier excepción (catch any exception) se representa mediante un símbolo mensaje que sale del ejemplar del puerto y llega hasta el componente de prueba; la palabra **catch** deberá figurar sobre la flecha de mensaje. Dentro del símbolo de llamada, la punta de la flecha del mensaje estará conectada a la región de suspensión precedente sobre el componente de prueba (véase la figura 89). Fuera del símbolo de llamada, la punta de la flecha del mensaje no deberá estar conectado a las regiones de suspensión precedente sobre el componente de prueba (véase la figura 90). La operación catch any exception no tendrá plantilla alguna ni tipo de excepción.

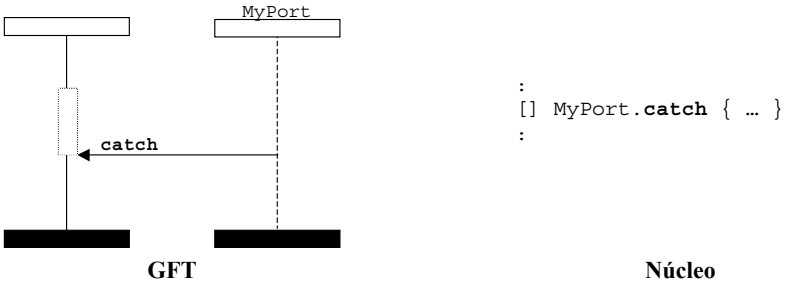


Figura 89/Z.142 – Operación catch any exception (dentro de un símbolo de llamadas)

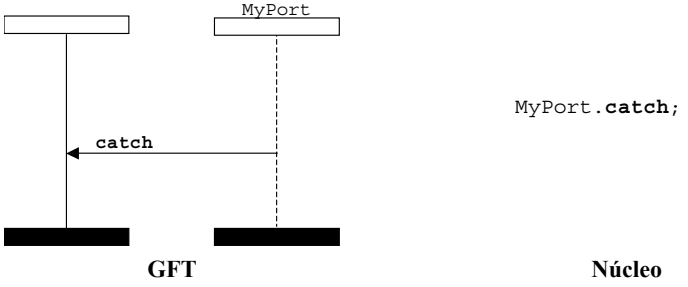


Figura 90/Z.142 – Operación catch any exception (fuera del símbolo de llamada)

11.8.4.6.3 Operación catch on any port

La operación adquirir desde cualquier puerto (catch on any port) se representa mediante un símbolo hallado (found) que simboliza cualquier puerto conectado al componente de prueba; la palabra clave **catch** deberá figurar encima de la flecha del mensaje. Dentro de un símbolo de llamada, la punta de la flecha del mensaje deberá estar conectada a la región de suspensión precedente sobre el componente de prueba (véase la figura 91). Fuera del símbolo de llamada, la punta de la flecha del mensaje no deberá estar conectada a la región de suspensión precedente sobre el componente de prueba (véase la figura 92). La plantilla, se la hubiere, se ubicará debajo de la flecha del mensaje.

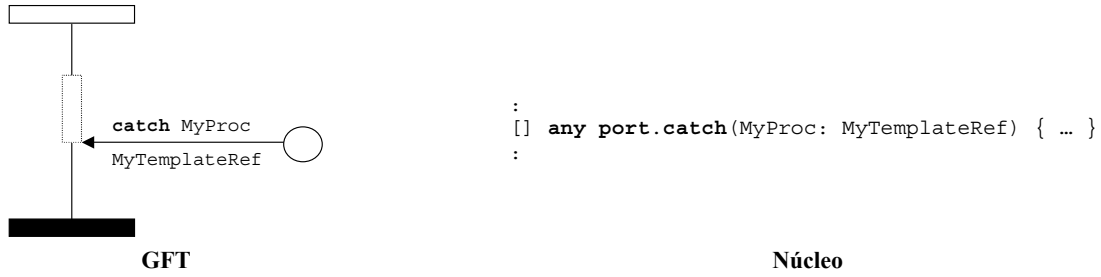


Figura 91/Z.142 – Operación catch on any port (dentro del símbolo de llamada)

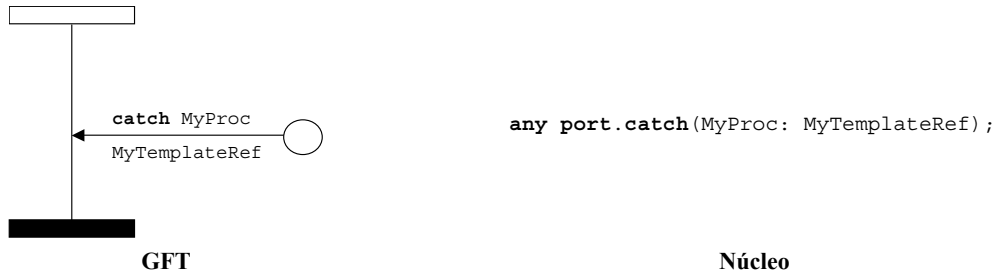
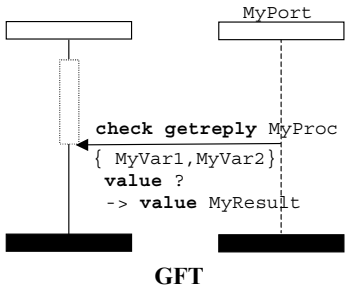


Figura 92/Z.142 – Operación catch on any port (fuera del símbolo de llamada)

11.8.5 La operación check

La operación verificar (check) se representará mediante un símbolo mensaje que sale del ejemplar del puerto y llega hasta el componente de prueba. La palabra clave **check** deberá situarse encima de la flecha del mensaje. La información relacionada con **receive** (véase la figura 93), **getcall**, **getreply** (véanse las figuras 94 y 95) y **catch** deben figurar después de la palabra check y de conformidad con las reglas que representan dichas operaciones.

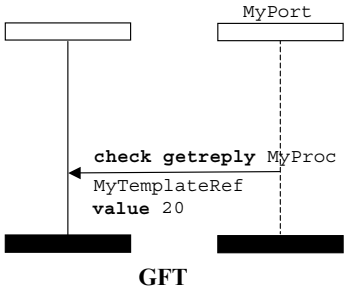
Figura 93/Z.142 – Operación check a receive (dentro de la plantilla en línea)




```
:
[] MyPort.check(getreply(MyProc:{MyVar1, MyVar2}
          value ?) -> value MyResult)
   { … }
:
```

Núcleo

Figura 94/Z.142 – Operación check a getreply (dentro del símbolo de llamada)



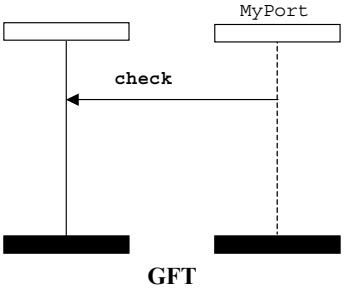

```
MyPort.check(getreply(MyProc: MyTemplateRef
value 20);
```

Núcleo

Figura 95/Z.142 – Operación check a getreply (fuera de un símbolo de llamada)

11.8.5.1 La operación check any

La operación verificar todo (check any) se representará mediante un símbolo mensaje que sale del ejemplar del puerto y llega hasta el componente de prueba; la palabra **check** deberá estar situada encima de la flecha del mensaje (véase la figura 96). No deberá aparecer ninguna palabra clave de operación de recepción, tipo y plantilla junto a ésta. Como opción, podrá añadirse la información de direcciones y de almacenamiento del emisor.




```
MyPort.check;
```

Núcleo

Figura 96/Z.142 – Operación check any

11.8.5.2 Operación check on any port

La operación verificar cualquier puerto (check on any port) se representa mediante un símbolo hallado (found) que simboliza cualquier puerto conectado al componente de prueba y la palabra clave **check** deberá figurar encima de la flecha del mensaje (véase la figura 97). La información relacionada con **receive**, **getcall**, **getreply** y **catch** deberá figurar a continuación de la palabra clave check y estar de conformidad con las reglas que representan dichas operaciones.

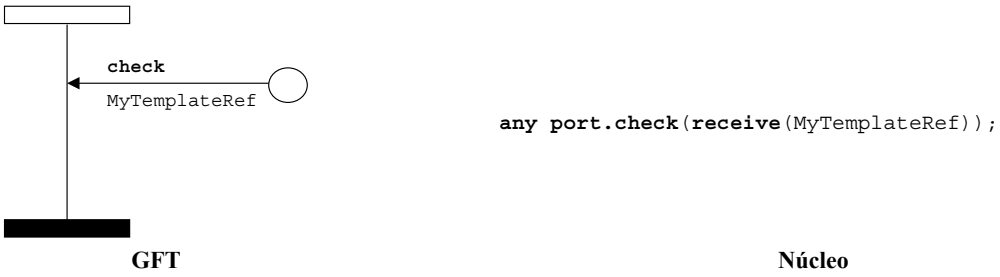


Figura 97/Z.142 – Operación check a receive on any port

11.8.6 Control de puertos de comunicación

11.8.6.1 La operación clear port

La operación borrar puerto (clear port) se representa mediante un símbolo de condición con la palabra clave **clear**. El símbolo estará conectado al ejemplar de componente que realiza la operación clear port y al puerto que se borra (véase la figura 98).

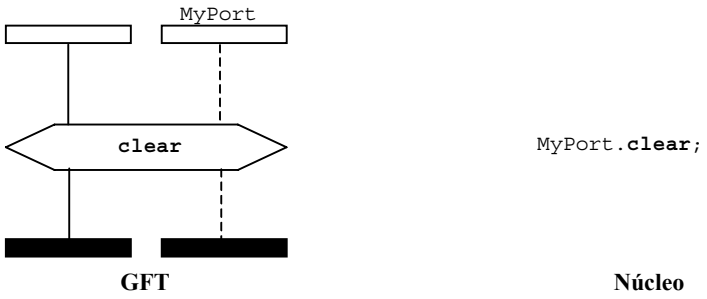


Figura 98/Z.142 – Operación clear port

11.8.6.2 La operación start port

La operación iniciar puerto (start port) se representa mediante un símbolo de condición con la palabra clave **start**. Este símbolo estará conectado al ejemplar del componente de prueba que realiza la operación inicia puerto y al puerto que es iniciado (véase la figura 99).

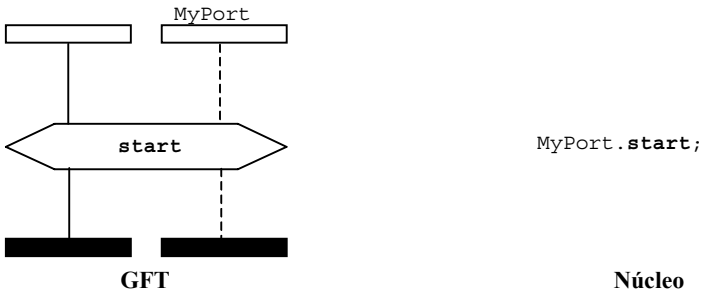


Figura 99/Z.142 – Operación start port

11.8.6.3 La operación stop port

La operación detener puerto (stop port) se representa mediante un símbolo de condición con la palabra clave **stop**. Este símbolo estará conectado al ejemplar del componente de prueba que realiza la operación stop port, y al puerto que se detiene (véase la figura 100).

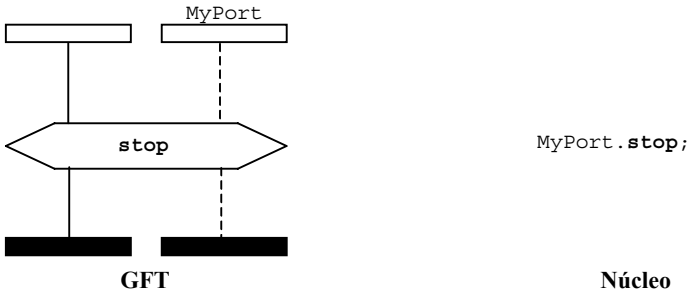


Figura 100/Z.142 – Operación stop port

11.8.6.4 Utilización de any y all en los puertos

La representación GFT de la palabra clave **any** de los puertos junto con las operaciones **receive**, **trigger**, **getcall**, **getreply**, **catch**, y **check** se describen en las correspondientes subcláusulas de 11.8.

La palabra clave **all** para los puertos junto con las operaciones **clear**, **start** o **stop** se representa mediante un símbolo de condición conectado al mismo que contiene la operación **clear**, **start** o **stop** para todas las distancias de puertos representadas en el diagrama GFT correspondiente a un caso de prueba, función o alternativa.

11.9 Operaciones de temporización

En el GFT, existen dos símbolos de temporizador diferentes: uno para temporizadores identificados y otro para identificadores de llamada (véase la figura 101). Se diferencian en que el símbolo de los temporizadores identificados se dibuja con líneas continuas y el de los temporizadores de llamada con líneas discontinuas. Un temporizador identificado tendrá su nombre al lado de su símbolo, mientras que un temporizador de llamada no tiene nombre. En esta cláusula se definen los temporizadores identificados. El temporizador de llamada se describe en 11.8.

Figura 101/Z.142 – Temporizador identificado y temporizadores de llamada

El GFT no proporciona una representación gráfica para la operación de temporizador **running** (tratándose ésta de una expresión booleana). Esta operación se indica mediante texto en los lugares donde va a utilizarse.

11.9.1 La operación start timer

Para la operación iniciar temporizador (start timer), el símbolo start timer se conectará al ejemplar del componente. Es posible escribir el nombre del temporizador y el valor opcional de su duración (entre paréntesis) (véase figura 102).

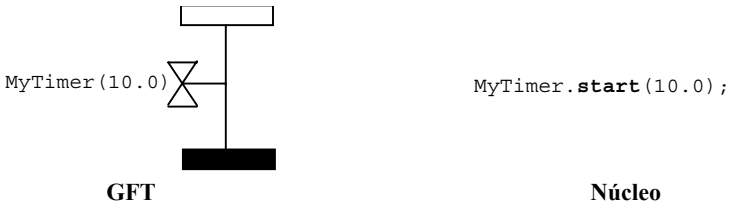


Figura 102/Z.142 – Operación start timer

11.9.2 La operación stop timer

Para la operación detener temporizador (stop timer), el símbolo stop timer se conectará al ejemplar del componente. Es posible asociar el nombre del temporizador opcional (véase la figura 103).

Figura 103/Z.142 – Operación stop timer

Los símbolos para las operaciones start timer y stop timer pueden conectarse con una línea vertical. En este caso, sólo es necesario especificar el identificador del temporizador al lado del símbolo del star timer (véase la figura 104).

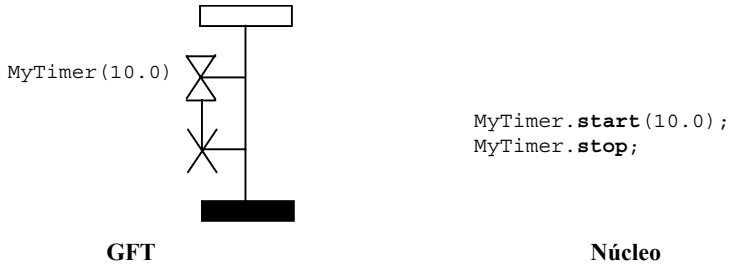


Figura 104/Z.142 – Símbolos start y stop timer conectados

11.9.3 La operación timeout

Para la operación temporización (timeout) se conectará el símbolo timeout al ejemplar del componente. Como opción, podrá escribirse el nombre del temporizador (véase la figura 105).

Figura 105/Z.142 – Operación timeout

Los símbolos para las operaciones start timer y timeout pueden conectarse con una línea vertical. En este caso sólo es necesario especificar el identificador del temporizador al lado del símbolo start timer (véase la figura 106).

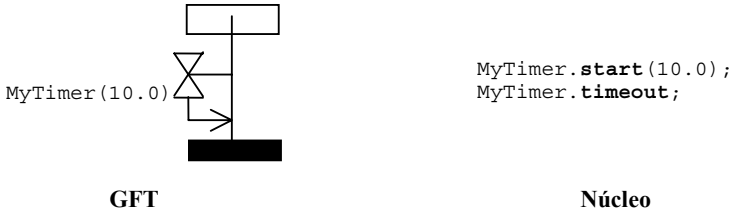


Figura 106/Z.142 – Símbolos start y timeout timer conectados

11.9.4 La operación read timer

La operación leer temporizador (read timer) se escribirá dentro de un recuadro de acción (véase la figura 107).

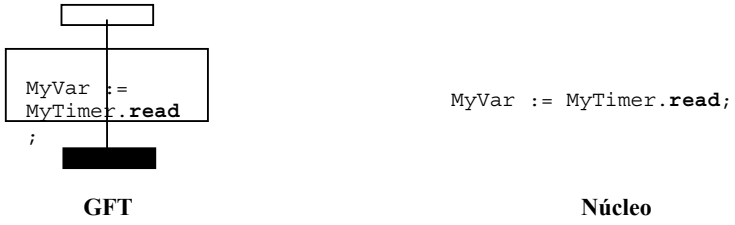


Figura 107/Z.142 – Operación read timer

11.9.5 Utilización de any y all en los temporizadores

La operación stop timer puede aplicarse a todos **(all)** los temporizadores (véase la figura 108).

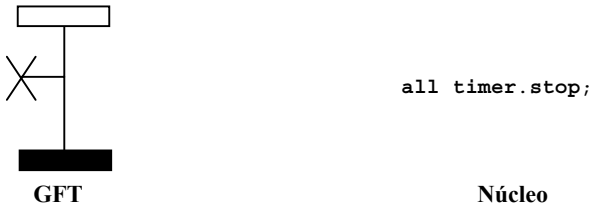
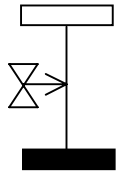


Figura 108/Z.142 – Paro de todos los temporizadores

La operación timeout puede aplicarse a cualquier **(any)** temporizador (véase la figura 109).



```
any timer.timeout;
```

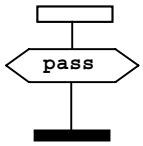
GFT **Núcleo**

Figura 109/Z.142 – Expiración de cualquier temporizador

11.10 Las operaciones veredicto de prueba

La operación establecer veredicto de prueba, **setverdict**, se representa en GFT mediante un símbolo de condición que contiene los valores **pass, fail, inconc** o **none** (véase la figura 110).

NOTA – Las reglas para establecer un nuevo veredicto son las mismas que las reglas de superposición TTCN-3 normales para veredictos de prueba.




```
setverdict(pass);
```

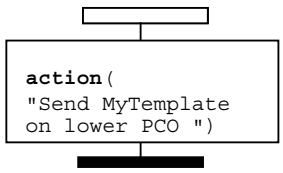
GFT **Núcleo**

Figura 110/Z.142 – Establecimiento de un veredicto local

El GFT no proporciona la representación gráfica para la operación **getverdict** (que se trata de una expresión). Esta operación se escribe mediante texto en los lugares donde se va a utilizar.

11.11 Acciones externas

Las acciones externas se representan dentro de símbolos del recuadro de acción (véase la figura 111). La sintaxis de la acción externa se escribe dentro del símbolo.




```
action("Send MyTemplate on lower PCO ");
```

GFT **Núcleo**

Figura 111/Z.142 – Acciones externas

11.12 Especificación de atributos

Los atributos definidos para la parte de control del módulo, casas de prueba, funciones y alternativas se representan dentro del símbolo de texto. La sintaxis de las preposiciones **with** se ubican dentro de ese símbolo. En la figura 112 se da un ejemplo.

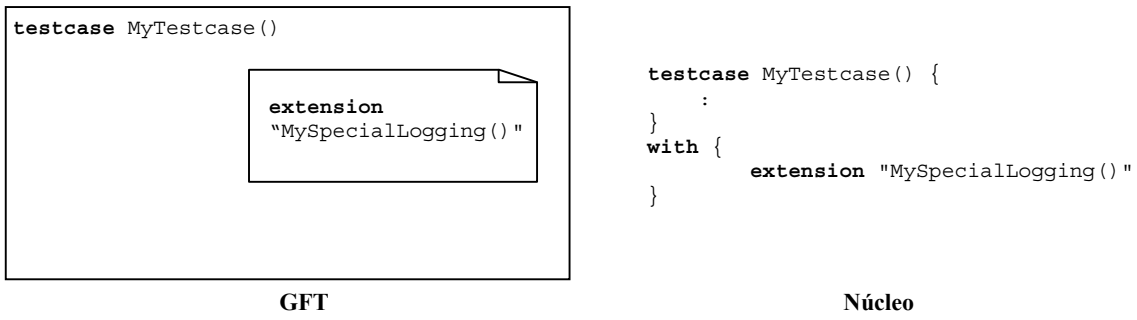


Figura 112/Z.142 – Especificación de atributos

Annex A (normative)

GFT BNF

A.1 Meta-language for GFT

The graphical syntax for GFT is defined on the basis of the graphical syntax of MSC [3]. The graphical syntax definition uses a meta-language, which is explained in 1.3.4/Z.120 [3]:

"The graphical syntax is not precise enough to describe the graphics such that there are no graphical variations. Small variations on the actual shapes of the graphical terminal symbols are allowed. These include, for instance, shading of the filled symbols, the shape of an arrow head and the relative size of graphical elements. Whenever necessary the graphical syntax will be supplemented with informal explanation of the appearance of the constructions.The meta-language consists of a BNF-like notation with the special meta-constructions: ***contains***, ***is followed by***, ***is associated with***, ***is attached to, above*** and ***set.*** These constructs behave like normal BNF production rules, but additionally they imply some logical or geometrical relation between the arguments. The ***is attached to*** construct behaves somewhat differently as explained below. The left-hand side of all constructs except ***above*** must be a symbol. A symbol is a non-terminal that produces in every production sequence exactly one graphical terminal. We will consider a symbol that ***is attached to*** other areas or that ***is associated with*** a text string as a symbol too. The explanation is informal and the meta-language does not precisely describe the geometrical dependencies".

See [3] for more details.

A.2 Conventions for the syntax description

Table A.1 defines the meta-notation used to specify the grammar for GFT. It is identical to the meta-notation used by TTCN-3, but different from the meta-notation used by MSC. In order to ease the readability, the correspondence to the MSC meta-notation is given in addition and differences are indicated.

Table A.1/Z.142 – The syntactic meta-notation

Meaning	TTCN-3	GFT	MSC	Differences
is defined to be	::=	::=	::=	
abc followed by xyz	abc xyz	abc xyz	abc xyz	
Alternative	|	|	|	
0 or 1 instances of abc	[abc]	[abc]	[abc]	
0 or more instances of abc	{abc}	{abc}	{abc}*	X
1 or more instances of abc	{abc} +	{abc} +	{abc} +	
Textual grouping	(...)	(...)	{...}	X
the non-terminal symbol abc	abc	abc (for a GFT non-terminal) or abc (for a TTCN non-terminal)	<abc>	X
a terminal symbol abc	**abc**	**abc**	**abc** or <name> or <character string>	X

A.3 The GFT grammar

A.3.1 Diagrams

A.3.1.1 Control Diagram

```
ControlDiagram ::=
        Frame contains ( ControlHeading ControlBodyArea )

ControlHeading ::=
        TTCN3ModuleKeyword TTCN3ModuleId
        { LocalDefinition [ SemiColon ] }

ControlBodyArea ::=
        { ControlInstanceArea TextLayer ControlEventLayer } set

TextLayer ::=
        { TextArea } set

ControlEventLayer::=
        ControlEventArea | ControlEventArea above ControlEventLayer

ControlEventArea ::=
    (
        InstanceTimerEventArea
    |   ControlActionArea
    |   InstanceInvocationArea
    |   ExecuteTestcaseArea
    |   ControlInlineExpressionArea )
    [ is associated with { CommentArea } set ]
```

A.3.1.2 Testcase Diagram

```
TestcaseDiagram ::=
        Frame contains ( TestcaseHeading TestcaseBodyArea )

TestcaseHeading ::=
        TestcaseKeyword TestcaseIdentifier
        "("[ TestcaseFormalParList ] ")"
        ConfigSpec
        { LocalDefinition [ SemiColon ] }

TestcaseBodyArea ::=
        { InstanceLayer TextLayer InstanceEventLayer PortEventLayer ConnectorLayer } set
```

```
InstanceLayer ::=
        { InstanceArea } set


InstanceEventLayer ::=
        InstanceEventArea | InstanceEventArea above InstanceEventLayer

InstanceEventArea ::=
    (
        InstanceSendEventArea
    |   InstanceReceiveEventArea
    |   InstanceCallEventArea
    |   InstanceGetcallEventArea
    |   InstanceReplyEventArea
    |   InstanceGetreplyWithinCallEventArea
    |   InstanceGetreplyOutsideCallEventArea
    |   InstanceRaiseEventArea
    |   InstanceCatchWithinCallEventArea
    |   InstanceCatchTimeoutWithinCallEventArea
    |   InstanceCatchOutsideCallEventArea
    |   InstanceTriggerEventArea
    |   InstanceCheckEventArea
    |   InstanceFoundEventArea
    |   InstanceTimerEventArea
    |   InstanceActionArea
    |   InstanceLabellingArea
    |   InstanceConditionArea
    |   InstanceInvocationArea
    |   InstanceDefaultHandlingArea
    |   InstanceComponentCreateArea
    |   InstanceComponentStartArea
    |   InstanceComponentStopArea
    |   InstanceInlineExpressionArea )
    [ is associated with { CommentArea } set ]
```

/* STATIC SEMANTICS – A condition area containing a boolean expression shall be used within alt inline expression, i.e. AltArea, and call inline expression, i.e. CallArea, only */

```
InstanceCallEventArea ::=
        InstanceBlockingCallEventArea
    |   InstanceNonBlockingCallEventArea


PortEventLayer ::=
        PortEventArea | PortEventArea above PortEventLayer

PortEventArea ::=
        PortOutEventArea
    |   PortOtherEventArea

PortOutEventArea ::=
        PortOutMsgEventArea
    |   PortGetcallOutEventArea
    |   PortGetreplyOutEventArea
    |   PortCatchOutEventArea
    |   PortTriggerOutEventArea
    |   PortCheckOutEventArea

PortOtherEventArea ::=
        PortInMsgEventArea
    |   PortCallInEventArea
    |   PortReplyInEventArea
    |   PortRaiseInEventArea
    |   PortConditionArea
    |   PortInvocationArea
    |   PortInlineExpressionArea

ConnectorLayer ::=
    {
        SendArea
    |   ReceiveArea
    |   NonBlockingCallArea
    |   GetcallArea
    |   ReplyArea
    |   GetreplyWithinCallArea
    |   GetreplyOutsideCallArea
    |   RaiseArea
    |   CatchWithinCallArea
    |   CatchOutsideCallArea
    |   TriggerArea
```

```
    |   CheckArea
    |   ConditionArea
    |   InvocationArea
    |   InlineExpressionArea
    } set
```

A.3.1.3 Function Diagram

```
FunctionDiagram ::=
        Frame contains ( FunctionHeading FunctionBodyArea )

FunctionHeading ::=
        FunctionKeyword FunctionIdentifier
        "("[ FunctionFormalParList ] ")"
        [ RunsOnSpec ] [ ReturnType ]
        { LocalDefinition [ SemiColon ] }

FunctionBodyArea ::=
        TestcaseBodyArea
```

A.3.1.4 Altstep Diagram

```
AltstepDiagram ::=
        Frame contains (AltstepHeading AltstepBodyArea )

AltstepHeading ::=
        AltstepKeyword AltstepIdentifier
        "("[AltstepFormalParList ] ")"
        [ RunsOnSpec ]
        { LocalDefinition [ SemiColon ] }

AltstepBodyArea ::=
        TestcaseBodyArea
```

/* STATIC SEMANTICS – A altstep body area shall contain a single alt inline expression only */

A.3.1.5 Comments

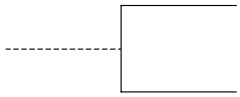
```
TextArea ::=
                TextSymbol
                contains ( { TTCN3Comments } [ MultiWithAttrib ] { TTCN3Comments } )
```

Note that there is no explicit rule for TTCN3 comments, they are explained in ES 201 873-1 [1], clause A.1.4.

/* STATIC SEMANTICS – Within a diagram there shall be at most one text symbol defining a with statement */

```
TextSymbol ::=
```

```
CommentArea ::=
                EventCommentSymbol contains TTCN3Comments

EventCommentSymbol ::=
```



/* STATIC SEMANTICS – A comment symbol can be attached to any graphical symbol in GFT */

A.3.1.6 Diagram

```
Frame ::=
```

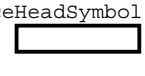
```
LocalDefinition ::=
            ConstDef
        |   VarInstance
        |   TimerInstance
```

/* STATIC SEMANTICS – Declarations of constants and variables with create, activate, and execute statements as well as with functions that include communication functions must not be made textually within LocalDefinition, but must be made graphically within create, default, execute, and reference symbols, respectively */

A.3.2 Instances

A.3.2.1 Component Instances

```
InstanceArea ::=
        ComponentInstanceArea
    |   PortInstanceArea

ComponentInstanceArea ::=
        ComponentHeadArea is followed by ComponentBodyArea

ComponentHeadArea::=
        ( MTCOp | SelfOp )
        is followed by ( InstanceHeadSymbol [ contains ComponentType ] )

InstanceHeadSymbol ::=
```



```
ComponentBodyArea ::=
        InstanceAxisSymbol
        is attached to { InstanceEventArea } set
        is followed by ComponentEndArea

InstanceAxisSymbol::=
```

```
ComponentEndArea ::=
        InstanceEndSymbol
    |   StopArea
    |   ReturnArea
    |   RepeatSymbol
    |   GotoArea
```

/* STATIC SEMANTICS – The return symbol shall be used within function diagrams only */
/* STATIC SEMANTICS – The repeat symbol shall end the component instance of a altstep diagram only */

A.3.2.2 Port Instances

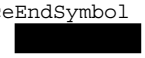
```
PortInstanceArea ::=
        PortHeadArea is followed by PortBodyArea

PortHeadArea::=
        Port
        is followed by ( InstanceHeadSymbol [ contains PortType ] )

PortBodyArea ::=
        PortAxisSymbol
        is attached to { PortEventArea } set
        is followed by InstanceEndSymbol

PortAxisSymbol::=
```

A.3.2.3 Control Instances

```
ControlInstanceArea ::=
        ControlInstanceHeadArea is followed by ControlInstanceBodyArea

ControlInstanceHeadArea ::=
        ControlKeyword
        is followed by InstanceHeadSymbol

ControlInstanceBodyArea::=
        InstanceAxisSymbol
        is attached to { ControlEventArea } set
        is followed by ControlInstanceEndArea

ControlInstanceEndArea ::=
        InstanceEndSymbol
```

A.3.2.4 Instance End

```
InstanceEndSymbol ::=
```

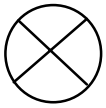


```
StopArea::=
        StopSymbol
        is associated with ( Expression )
```

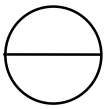
/* STATIC SEMANTICS – The expression shall refer to either the mtc or to self */

```
StopSymbol::=
```

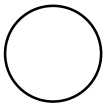
```
ReturnArea::=
        ReturnSymbol
        [ is associated with Expression ]

ReturnSymbol::=
```



```
RepeatSymbol::=
```



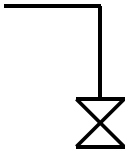
```
GotoArea::=
        GotoSymbol
        contains LabelIdentifier

GotoSymbol::=
```



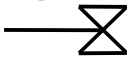
A.3.3 Timer

```
InstanceTimerEventArea::=
        InstanceTimerStartArea
    |   InstanceTimerStopArea
    |   InstanceTimeoutArea


InstanceTimerStartArea ::=
        TimerStartSymbol
        is associated with ( TimerRef ["(" TimerValue ")"] )
        is attached to InstanceAxisSymbol
        [is attached to { TimerStopSymbol2 | TimeoutSymbol3 } ]
```

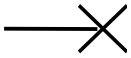
```
TimerStartSymbol ::=
        TimerStartSymbol1 | TimerStartSymbol2

TimerStartSymbol1 ::=
```



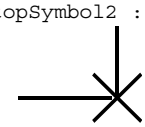
```
TimerStartSymbol2 ::=
```



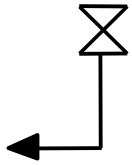
```
InstanceTimerStopArea ::=
        TimerStopArea1 | TimerStopArea2

TimerStopArea1 ::=
        TimerStopSymbol1
        is associated with TimerRef
        is attached to InstanceAxisSymbol

TimerStopArea2 ::=
        TimerStopSymbol2
        is attached to InstanceAxisSymbol
        is attached to TimerStartSymbol

TimerStopSymbol1 ::=
```



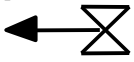
```
TimerStopSymbol2 ::=
```



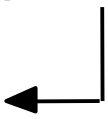
```
InstanceTimeoutArea ::=
        TimeoutArea1 | TimeoutArea2

TimeoutArea1 ::=
        TimeoutSymbol
        is associated with TimerRef
        is attached to InstanceAxisSymbol

TimeoutArea2 ::=
        TimeoutSymbol3
        is attached to InstanceAxisSymbol
        is attached to TimerStartSymbol

TimeoutSymbol ::=
        TimeoutSymbol1 | TimeoutSymbol2

TimeoutSymbol1 ::=
```



```
TimeoutSymbol2 ::=
```



```
TimeoutSymbol3 ::=
```



A.3.4 Action

```
InstanceActionArea ::=
        ActionSymbol
        contains { ActionStatement [SemiColon] }+
        is attached to InstanceAxisSymbol

ActionSymbol ::=
```

```
ActionStatement ::=
        SUTStatements
    |   ConnectStatement
    |   MapStatement
    |   DisconnectStatement
    |   UnmapStatement
    |   ConstDef
    |   VarInstance
    |   TimerInstance
    |   Assignment
    |   LogStatement
    |   LoopConstruct
    |   ConditionalConstruct
```

/* STATIC SEMANTICS – Declarations of constants and variables with create, activate, and execute statements as well as with function invocations of user-defined functions must not be made textually within an action box, but must be made graphically within create, default, execute, and reference symbols, respectively */
/* STATIC SEMANTICS – Assignments with create, activate, and execute statements as well as with function invocations of user-defined functions must not be made textually within an action box, but must be made graphically within create, default, execute, and reference symbols, respectively */
/* STATIC SEMANTICS – Only those loop and conditional constructs, which do not involve communication operations, i.e. those with "data functions" only, may be contained in action boxes */

```
ControlActionArea ::=
        ActionSymbol
        is attached to InstanceAxisSymbol
        contains { ControlActionStatement [SemiColon] }+

ControlActionStatement ::=
        SUTStatements
    |   ConstDef
    |   VarInstance
    |   TimerInstance
    |   Assignment
    |   LogStatement
```

/* STATIC SEMANTICS – Declarations of constants and variables with create, activate, and execute statements as well as with function invocations of user-defined functions must not be made textually within an action box, but must be made graphically within create, default, execute, and reference symbols, respectively */
/* STATIC SEMANTICS – Assignments with create, activate, and execute statements as well as with function invocations of user-defined functions must not be made textually within an action box, but must be made graphically within create, default, execute, and reference symbols, respectively */

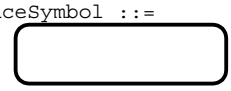
A.3.5 Invocation

```
InvocationArea ::=
        ReferenceSymbol
        contains Invocation
        is attached to InstanceAxisSymbol
        [ is attached to { PortAxisSymbol } set ]
```

/* STATIC SEMANTICS – All port instances have to be covered by the reference symbol for an invoked function if it has a runs on specification, as well as for an invoked altstep */
/* STATIC SEMANTICS – Only those port instances, which are passed into a function via port parameters, have to be covered by the reference symbol for an invoked function without a runs on specification. Note that the reference symbol may be attached to port instances which are not passed as port parameters into the function. */

```
Invocation ::=
        FunctionInstance
    |   AltstepInstance
    |   ConstDef
    |   VarInstance
    |   Assignment
```

```
ReferenceSymbol ::=
```



A.3.5.1 Function and Altstep Invocation on Component/Control Instances

```
InstanceInvocationArea ::=
        InstanceInvocationBeginSymbol
        is followed by InstanceInvocationEndSymbol
        is attached to InstanceAxisSymbol
        is attached to InvocationArea

InstanceInvocationBeginSymbol ::=
        VoidSymbol

InstanceInvocationEndSymbol ::=
        VoidSymbol
```

A.3.5.2 Function and Altstep Invocation on Ports

```
PortInvocationArea ::=
        PortInvocationBeginSymbol
        is followed by PortInvocationEndSymbol
        is attached to PortAxisSymbol
        is attached to InvocationArea
```

/* STATIC SEMANTICS – Only invocations with function instances and test step instances shall be attached to a port instance, in that case all port instances have to be covered by the reference symbol for an invoked function if it has a runs on specification, as well as for an invoked altstep */

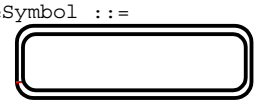
```
PortInvocationBeginSymbol ::=
        VoidSymbol

PortInvocationEndSymbol ::=
        VoidSymbol
```

A.3.5.3 Testcase Execution

```
ExecuteTestcaseArea ::=
        ExecuteSymbol
        contains TestCaseExecution
        is attached to InstanceAxisSymbol

TestCaseExecution::=
        TestcaseInstance
    |   ConstDef
    |   VarInstance
    |   Assignment
```

/* STATIC SEMANTICS – Declarations of constants and variables as well as assignments shall use as outermost right-hand expression an execute statement */

```
ExecuteSymbol ::=
```

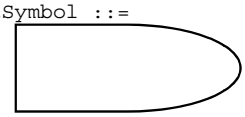


A.3.6 Activation/Deactivation of Defaults

```
InstanceDefaultHandlingArea ::=
        DefaultSymbol
        contains DefaultHandling
        is attached to InstanceAxisSymbol

DefaultHandling::=
        ActivateOp
    |   DeactivateStatement
    |   ConstDef
    |   VarInstance
    |   Assignment
```

/* STATIC SEMANTICS – Declarations of constants and variables as well as assignments shall use as outermost right-hand expression an activate statement */

```
DefaultSymbol ::=
```



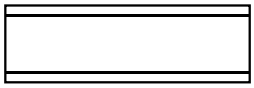
A.3.7 Test Components

A.3.7.1 Creation of Test Components

```
InstanceComponentCreateArea ::=
        CreateSymbol
        contains Creation
        is attached to InstanceAxisSymbol

Creation ::=
        CreateOp
    |   ConstDef
    |   VarInstance
    |   Assignment
```

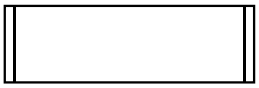
/* STATIC SEMANTICS – Declarations of constants and variables as well as assignments shall use as outermost right-hand expression a create statement */

```
CreateSymbol ::=
```



A.3.7.2 Starting Test Components

```
InstanceComponentStartArea ::=
        StartSymbol
        contains StartTCStatement
        is attached to InstanceAxisSymbol

StartSymbol ::=
```



A.3.7.3 Stopping Test Components

```
InstanceComponentStopArea ::=
        StopSymbol
        is associated with ( Expression | AllKeyword )
        is attached to InstanceAxisSymbol
```

/* STATIC SEMANTICS – The expression shall refer to a component identifier */
/* STATIC SEMANTICS – The instance component stop area shall be used as last event of an operand in an inline expression symbol, if the component stops itself (e.g., self.stop) or stops the test execution (e.g., mtc.stop). */

A.3.8 Inline Expressions

```
InlineExpressionArea ::=
        IfArea
    |   ForArea
    |   WhileArea
    |   DoWhileArea
    |   AltArea
    |   InterleaveArea
    |   CallArea

IfArea ::=
        IfInlineExpressionArea
        is attached to InstanceInlineExpressionBeginSymbol
        [ is attached to InstanceInlineExpressionSeparatorSymbol ]
        is attached to InstanceInlineExpressionEndSymbol
        [ is attached to { PortInlineExpressionBeginSymbol } set
          [ is attached to { PortInlineExpressionSeparatorSymbol } set ]
          is attached to { PortInlineExpressionEndSymbol } set ]
```

/* STATIC SEMANTICS – If a SeparatorSymbol is contained in the inline expression symbol, then InstanceInlineExpressionSeparatorSymbols on component and port instances are used to attach the SeparatorSymbol to the respective instances. */

```
InstanceInlineExpressionBeginSymbol ::=
        VoidSymbol

InstanceInlineExpressionSeparatorSymbol::=
        VoidSymbol

InstanceInlineExpressionEndSymbol ::=
        VoidSymbol

VoidSymbol ::=      .

IfInlineExpressionArea ::=
        InlineExpressionSymbol
        contains ( IfKeyword "(" BooleanExpression ")"
                   is followed by OperandArea
                   [ is followed by SeparatorSymbol
                     is followed by OperandArea ] )
OperandArea ::=
        ConnectorLayer
```

/* STATIC SEMANTICS – The event layer within an operand area shall not have a condition with a boolean expression */

```
ForArea ::=
        ForInlineExpressionArea
        is attached to InstanceInlineExpressionBeginSymbol
        is attached to InstanceInlineExpressionEndSymbol
        [ is attached to { PortInlineExpressionBeginSymbol } set
          is attached to { PortInlineExpressionEndSymbol } set ]

ForInlineExpressionArea ::=
        InlineExpressionSymbol
        contains ( ForKeyword "(" Initial [SemiColon] Final [SemiColon] Step ")"
                   is followed by OperandArea )

WhileArea ::=
        WhileInlineExpressionArea
        is attached to InstanceInlineExpressionBeginSymbol
        is attached to InstanceInlineExpressionEndSymbol
        [ is attached to { PortInlineExpressionBeginSymbol  } set
          is attached to { PortInlineExpressionEndSymbol } set ]

WhileInlineExpressionArea ::=
        InlineExpressionSymbol
        contains ( WhileKeyword "(" BooleanExpression ")"
                   is followed by OperandArea )

DoWhileArea ::=
        DoWhileInlineExpressionArea
        is attached to InstanceInlineExpressionBeginSymbol
        is attached to InstanceInlineExpressionEndSymbol
        [ is attached to { PortInlineExpressionBeginSymbol } set
          is attached to { PortInlineExpressionEndSymbol } set ]

DoWhileInlineExpressionArea ::=
        InlineExpressionSymbol
        contains ( DoKeyword WhileKeyword "(" BooleanExpression ")"
                   is followed by OperandArea )

AltArea ::=
        AltInlineExpressionArea
        is attached to InstanceInlineExpressionBeginSymbol
        { is attached to InstanceInlineExpressionSeparatorSymbol }
        is attached to InstanceInlineExpressionEndSymbol
        [ is attached to { PortInlineExpressionBeginSymbol } set
          [ is attached to { PortInlineExpressionSeparatorSymbol } set ]
          is attached to { PortInlineExpressionEndSymbol } set ]
```

/* STATIC SEMANTICS – The number of InstanceInlineExpressionSeparatorSymbol per component and port instances has to adhere to the number of SeparatorSymbols contained within the inline expression symbol: the InstanceInlineExpressionSeparatorSymbol on component and port instances are used to attach the SeparatorSymbols to the respective instances. */

```
AltInlineExpressionArea ::=
        InlineExpressionSymbol
        contains ( AltKeyword
                   is followed by GuardedOperandArea
                   { is followed by SeparatorSymbol
                     is followed by GuardedOperandArea }
                   [ is followed by SeparatorSymbol
                     is followed by ElseOperandArea ] )

GuardedOperandArea ::=
        GuardOpLayer is followed by
        ConnectorLayer
```

/* STATIC SEMANTICS – For the individual operands of an alt inline expression at first, either a InstanceTimeoutArea shall be given on the component instance, or a GuardOpLayer has to be given */

```
GuardOpLayer ::=
        DoneArea
    |   ReceiveArea
    |   TriggerArea
    |   GetcallArea
    |   CatchOutsideCallArea
    |   CheckArea
    |   GetreplyOutsideCallArea

ElseOperandArea ::=
        ElseConditionArea
        is followed by ConnectorLayer

InterleaveArea ::=
        InterleaveInlineExpressionArea
        is attached to InstanceInlineExpressionBeginSymbol
        { is attached to InstanceInlineExpressionSeparatorSymbol }
        is attached to InstanceInlineExpressionEndSymbol
        [ is attached to { PortInlineExpressionBeginSymbol } set
          [ is attached to { PortInlineExpressionSeparatorSymbol } set ]
          is attached to { PortInlineExpressionEndSymbol } set ]
```

/* STATIC SEMANTICS – The number of InstanceInlineExpressionSeparatorSymbol per component and port instances has to adhere to the number of SeparatorSymbols contained within the inline expression symbol: the InstanceInlineExpressionSeparatorSymbol on component and port instances are used to attach the SeparatorSymbols to the respective instances. */

```
InterleaveInlineExpressionArea ::=
        InlineExpressionSymbol
        contains ( InterleavedKeyword
                   is followed by UnguardedOperandArea
                   { is followed by SeparatorSymbol
                     is followed by UnguardedOperandArea } )

UnguardedOperandArea ::=
        UnguardedOpLayer is followed by
        ConnectorLayer
```

/* STATIC SEMANTICS – The connector layer within an interleave inline expression area may not contain loop statements, goto, activate, deactivate, stop, return or calls to functions */

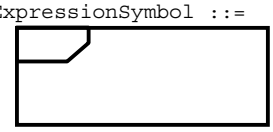
```
UnguardedOpLayer ::=
        ReceiveArea
    |   TriggerArea
    |   GetcallArea
    |   CatchOutsideCallArea
    |   CheckArea
    |   GetreplyOutsideCallArea

CallArea ::=
        CallInlineExpressionArea
        is attached to InstanceInlineExpressionBeginSymbol
        { is attached to InstanceInlineExpressionSeparatorSymbol }
        is attached to InstanceInlineExpressionEndSymbol
        [ is attached to { PortInlineExpressionBeginSymbol } set
          [ is attached to { PortInlineExpressionSeparatorSymbol } set ]
          is attached to { PortInlineExpressionEndSymbol } set ]
```

/* STATIC SEMANTICS – The number of InstanceInlineExpressionSeparatorSymbol per component and port instances has to adhere to the number of SeparatorSymbols contained within the inline expression symbol: the InstanceInlineExpressionSeparatorSymbol on component and port instances are used to attach the SeparatorSymbols to the respective instances. */

```
CallInlineExpressionArea ::=
        InlineExpressionSymbol
        contains ( CallOpKeyword "(" TemplateInstance ")" [ ToClause ]
                   is followed by InstanceCallEventArea
                   { is followed by SeparatorSymbol
                     is followed by GuardedCallOperandArea } )

GuardedCallOperandArea ::=
        [ GuardedConditionLayer is followed by ]
        CallBodyOpsLayer
        is attached to SuspensionRegionSymbol
        is followed by ConnectorLayer
```

/* STATIC SEMANTICS – For the individual operands in the GuardedCallOperandArea of a call inline expression at first, either a InstanceCatchTimeoutWithinCallEventArea shall be given on the component instance, or a CallBodyOpsLayer has to be given */

```
GuardedConditionLayer ::=
        BooleanExpressionConditionArea
    |   DoneArea

CallBodyOpsLayer ::=
        GetreplyWithinCallArea
    |   CatchWithinCallArea

InlineExpressionSymbol ::=
```



```
SeparatorSymbol::=
        - - - - - - - - - - - -
```

A.3.8.1 Inline Expressions on Component Instances

```
InstanceInlineExpressionArea ::=
        InstanceIfArea
        |InstanceForArea
        |InstanceWhileArea
        |InstanceDoWhileArea
        |InstanceAltArea
        |InstanceInterleaveArea
        |InstanceCallArea

InstanceIfArea ::=
        ( InstanceInlineExpressionBeginSymbol
          { is followed by InstanceEventArea }
          [ is followed by InstanceInlineExpressionSeparatorSymbol
            { is followed by InstanceEventArea } ]
          is followed by InstanceInlineExpressionEndSymbol )
        is attached to InstanceAxisSymbol
        is attached to IfInlineExpressionArea

InstanceForArea ::=
        ( InstanceInlineExpressionBeginSymbol
          { is followed by InstanceEventArea }
          is followed by InstanceInlineExpressionEndSymbol )
        is attached to InstanceAxisSymbol
        is attached to ForInlineExpressionArea

InstanceWhileArea ::=
        ( InstanceInlineExpressionBeginSymbol
          { is followed by InstanceEventArea }
          is followed by InstanceInlineExpressionEndSymbol )
        is attached to InstanceAxisSymbol
        is attached to WhileInlineExpressionArea

InstanceDoWhileArea ::=
        ( InstanceInlineExpressionBeginSymbol
          { is followed by InstanceEventArea }
          is followed by InstanceInlineExpressionEndSymbol )
        is attached to InstanceAxisSymbol
        is attached to DoWhileInlineExpressionArea

InstanceAltArea ::=
        ( InstanceInlineExpressionBeginSymbol
          [ is followed by InstanceBooleanExpressionConditionArea ]
          is followed by InstanceGuardArea
```

```
          { is followed by InstanceInlineExpressionSeparatorSymbol
            is followed by InstanceGuardArea }
          [ is followed by InstanceInlineExpressionSeparatorSymbol
            is followed by InstanceElseGuardArea ]
          is followed by InstanceInlineExpressionEndSymbol )
        is attached to InstanceAxisSymbol
        is attached to AltInlineExpressionArea

InstanceGuardArea ::=
               ( InstanceInvocationArea
               | InstanceGuardOpArea )
               { is followed by InstanceEventArea }
               is attached to InstanceAxisSymbol
```

/* STATIC SEMANTICS – The instance invocation area shall contain a altstep instance only */

```
InstanceGuardOpArea ::=
        (   InstanceTimeoutArea
        |   InstanceReceiveEventArea
        |   InstanceTriggerEventArea
        |   InstanceGetcallEventArea
        |   InstanceGetreplyOutsideCallEventArea
        |   InstanceCatchOutsideCallEventArea
        |   InstanceCheckEventArea
        |   InstanceDoneArea )
        is attached to InstanceAxisSymbol

InstanceElseGuardArea ::=
               ElseConditionArea
               { is followed by InstanceEventArea }
               is attached to InstanceAxisSymbol


InstanceInterleaveArea ::=
        ( InstanceInlineExpressionBeginSymbol
          is followed by InstanceInterleaveGuardArea
          { is followed by InstanceInlineExpressionSeparatorSymbol
            is followed by InstanceInterleaveGuardArea }
          is followed by InstanceInlineExpressionEndSymbol )
        is attached to InstanceAxisSymbol
        is attached to InterleaveInlineExpressionArea

InstanceInterleaveGuardArea ::=
        InstanceGuardOpArea
        { is followed by InstanceEventArea }
        is attached to InstanceAxisSymbol
```

/* STATIC SEMANTICS – The instance event area may not contain loop statements, goto, activate, deactivate, stop, return or calls to functions */

```
InstanceCallArea ::=
        ( InstanceInlineExpressionBeginSymbol
          [ is followed by InstanceBooleanExpressionConditionArea ]
          [ is followed by InstanceCallOpArea ]
          { is followed by InstanceInlineExpressionSeparatorSymbol
            is followed by InstanceCallGuardArea}
          is followed by InstanceInlineExpressionEndSymbol )
        is attached to InstanceAxisSymbol
        is attached to CallInlineExpressionArea

InstanceCallOpArea ::=
        InstanceCallEventArea
        is followed by SuspensionRegionSymbol
        [ is attached to InstanceCallTimerStartArea ]
        is attached to InstanceAxisSymbol
        is attached to CallInlineExpressionArea

SuspensionRegionSymbol ::=
        
InstanceCallGuardArea ::=
        SuspensionRegionSymbol
        [ is attached to InstanceGetreplyWithinCallEventArea
                       |    InstanceCatchWithinCallEventArea
                       |    InstanceCatchTimeoutWithinCallEventArea ]
        { is followed by InstanceEventArea }
        is attached to InstanceAxisSymbol
        is attached to CallInlineExpressionArea
```

A.3.8.2 Inline Expressions on Ports

```
PortInlineExpressionArea ::=
        PortIfArea
    |   PortForArea
    |   PortWhileArea
    |   PortDoWhileArea
    |   PortAltArea
    |   PortInterleaveArea
    |   PortCallArea

PortIfArea ::=
        (PortInlineExpressionBeginSymbol
          { is followed by PortEventArea }
          [ is followed by PortInlineExpressionSeparatorSymbol
            { is followed by PortEventArea }]
          is followed by PortInlineExpressionEndSymbol )
        is attached to PortAxisSymbol
        is attached to IfInlineExpressionArea

PortInlineExpressionBeginSymbol ::=
        VoidSymbol

PortInlineExpressionSeparatorSymbol::=
        VoidSymbol

PortInlineExpressionEndSymbol::=
        VoidSymbol

PortForArea ::=
        (PortInlineExpressionBeginSymbol
          { is followed by PortEventArea }
          is followed by PortInlineExpressionEndSymbol )
        is attached to PortAxisSymbol
        is attached to ForInlineExpressionArea

PortWhileArea ::=
        (PortInlineExpressionBeginSymbol
          { is followed by PortEventArea }
          is followed by PortInlineExpressionEndSymbol )
        is attached to PortAxisSymbol
        is attached to WhileInlineExpressionArea

PortDoWhileArea ::=
        ( PortInlineExpressionBeginSymbol
          { is followed by PortEventArea }
          is followed by PortInlineExpressionEndSymbol )
        is attached to PortAxisSymbol
        is attached to DoWhileInlineExpressionArea

PortAltArea ::=
        (PortInlineExpressionBeginSymbol
          [ is followed by PortOutEventArea ]
          { is followed by PortEventArea }
          { is followed by PortInlineExpressionSeparatorSymbol
            [ is followed by PortOutEventArea ]
            { is followed by PortEventArea } }
          is followed by PortInlineExpressionEndSymbol )
        is attached to PortAxisSymbol
        is attached to AltInlineExpressionArea

PortInterleaveArea ::=
        ( PortInlineExpressionBeginSymbol
          [ is followed by PortOutEventArea ]
          { is followed by PortEventArea }
          { is followed by PortInlineExpressionSeparatorSymbol
            [ is followed by PortOutEventArea ]
            { is followed by PortEventArea } }
          is followed by PortInlineExpressionEndSymbol )
        is attached to PortAxisSymbol
        is attached to InterleaveInlineExpressionArea

PortCallArea ::=
        (PortInlineExpressionBeginSymbol
          [ is followed by PortCallInEventArea]
          { is followed by PortEventArea }
          { is followed by PortInlineExpressionSeparatorSymbol
            [ is followed by PortOutEventArea ]
            { is followed by PortEventArea } }
```

```
          is followed by PortInlineExpressionEndSymbol )
        is attached to InstanceAxisSymbol
        is attached to CallInlineExpressionArea
```

A.3.8.3 Inline Expressions on Control Instances

```
ControlInlineExpressionArea ::=
        ControlIfArea
    |   ControlForArea
    |   ControlWhileArea
    |   ControlDoWhileArea
    |   ControlAltArea
    |   ControlInterleaveArea

ControlIfArea ::=
        ( InstanceInlineExpressionBeginSymbol
          [ is followed by ControlEventArea ]
          [ is followed by InstanceInlineExpressionSeparatorSymbol
            is followed by ControlEventArea ]
          is followed by InstanceInlineExpressionEndSymbol )
        is attached to InstanceAxisSymbol
        is attached to IfInlineExpressionArea

ControlForArea ::=
        ( InstanceInlineExpressionBeginSymbol
          [ is followed by ControlEventArea ]
          is followed by InstanceInlineExpressionEndSymbol )
        is attached to InstanceAxisSymbol
        is attached to ForInlineExpressionArea

ControlWhileArea ::=
        ( InstanceInlineExpressionBeginSymbol
          [ is followed by ControlEventArea ]
          is followed by InstanceInlineExpressionEndSymbol )
        is attached to InstanceAxisSymbol
        is attached to WhileInlineExpressionArea

ControlDoWhileArea ::=
        ( InstanceInlineExpressionBeginSymbol
          [ is followed by ControlEventArea ]
          is followed by InstanceInlineExpressionEndSymbol )
        is attached to InstanceAxisSymbol
        is attached to DoWhileInlineExpressionArea

ControlAltArea ::=
        ( InstanceInlineExpressionBeginSymbol
          [ is followed by ControlGuardArea ]
          { is followed by InstanceInlineExpressionSeparatorSymbol
            is followed by ControlGuardArea }
          [ is followed by InstanceInlineExpressionSeparatorSymbol
            is followed by ControlElseGuardArea ]
          is followed by InstanceInlineExpressionEndSymbol )
        is attached to InstanceAxisSymbol
        is attached to AltInlineExpressionArea

ControlGuardArea ::=
              ( InstanceInvocationArea
              | InstanceTimeoutArea)
              { is followed by ControlEventArea }
              is attached to InstanceAxisSymbol
```

/* STATIC SEMANTICS – The instance invocation area shall contain a altstep instance only */

```
ControlElseGuardArea ::=
              ElseConditionArea
              { is followed by ControlEventArea }
              is attached to InstanceAxisSymbol


ControlInterleaveArea ::=
        ( InstanceInlineExpressionBeginSymbol
          [ is followed by ControlInterleaveGuardArea ]
          { is followed by InstanceInlineExpressionSeparatorSymbol
            is followed by ControlInterleaveGuardArea }
          is followed by InstanceInlineExpressionEndSymbol )
        is attached to InstanceAxisSymbol
        is attached to InterleaveInlineExpressionArea
```

```
ControlInterleaveGuardArea ::=
        InstanceTimeoutArea
        { is followed by ControlEventArea }
        is attached to InstanceAxisSymbol
```

/* STATIC SEMANTICS – The instance event area may not contain loop statements, goto, activate, deactivate, stop, return or calls to functions */

A.3.9 Condition

```
ConditionArea ::=
        PortOperationArea


BooleanExpressionConditionArea ::=
        ConditionSymbol
        contains BooleanExpression
        is attached to InstanceConditionBeginSymbol
        is attached to InstanceConditionEndSymbol
```

/* STATIC SEMANTICS – Boolean expressions within conditions shall be used as guards within alt and call inline expressions only They shall be attached to a single test component or control instance only.*/

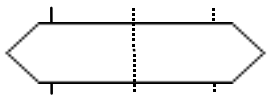
```
InstanceConditionBeginSymbol ::=
        VoidSymbol

InstanceConditionEndSymbol ::=
        VoidSymbol

DoneArea ::=
        ConditionSymbol
        contains DoneStatement
        is attached to InstanceConditionBeginSymbol
        is attached to InstanceConditionEndSymbol

SetVerdictArea ::=
        ConditionSymbol
        contains SetVerdictText
        is attached to InstanceConditionBeginSymbol
        is attached to InstanceConditionEndSymbol

SetVerdictText ::=
        ( SetVerdictKeyword "(" SingleExpression ")" )
        | pass
        | fail
        | inconc
        | none
```

/* STATIC SEMANTICS – SingleExpression must resolve to a value of type verdict */
/* STATIC SEMANTICS – The SetLocalVerdict shall not be used to assign the value error */
/* STATIC SEMANTICS – If the keywords pass, fail, inconc, and fail are used, the form with the setverdict keyword shall not be used */

```
PortOperationArea ::=
        ConditionSymbol
        contains PortOperationText
        is attached to InstanceConditionBeginSymbol
        is attached to InstanceConditionEndSymbol
        [   is attached to { PortInlineExpressionBeginSymbol }+ set
            is attached to { PortInlineExpressionEndSymbol }+ set ]
        is attached to InstancePortOperationArea
        is attached to PortConditionArea
```

/* STATIC SEMANTICS – The condition symbol shall be attached to either to all ports or to just one port */

If the condition symbol crosses a port axis symbol of a port which is not involved in this port operation, its the port axis symbol is drawn through:



```
PortOperationText ::=
            ClearOpKeyword
        |   StartKeyword
        |   StopKeyword
```

```
ElseConditionArea ::=
        ConditionSymbol
        contains ElseKeyword
        is attached to InstanceAxisSymbol

ConditionSymbol ::=
```

A.3.9.1 Condition on Component Instances

```
InstanceConditionArea ::=
            InstanceDoneArea
        |   InstanceSetVerdictArea
        |   InstancePortOperationArea

InstanceBooleanExpressionConditionArea ::=
        InstanceConditionBeginSymbol
        is followed by InstanceConditionEndSymbol
        is attached to InstanceAxisSymbol
        is attached to BooleanExpressionConditionArea

InstanceDoneArea ::=
        InstanceConditionBeginSymbol
        is followed by InstanceConditionEndSymbol
        is attached to InstanceAxisSymbol
        is attached to DoneArea

InstanceSetVerdictArea ::=
        InstanceConditionBeginSymbol
        is followed by InstanceConditionEndSymbol
        is attached to InstanceAxisSymbol
        is attached to SetVerdictArea

InstancePortOperationArea ::=
        InstanceConditionBeginSymbol
        is followed by InstanceConditionEndSymbol
        is attached to InstanceAxisSymbol
        is attached to PortOperationArea
```

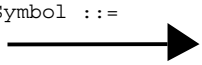
A.3.9.2 Condition on Ports

```
PortConditionArea ::=
        PortConditionBeginSymbol
        is followed by PortConditionEndSymbol
        is attached to PortAxisSymbol
        is attached to PortOperationArea

PortConditionBeginSymbol ::=
        VoidSymbol

PortConditionEndSymbol ::=
        VoidSymbol
```

A.3.10 Message-based Communication

```
SendArea ::=
        MessageSymbol
        [ is associated with Type ]
        is associated with ( [ DerivedDef AssignmentChar ] TemplateBody
                             [ ToClause ] )
        is attached to InstanceSendEventArea
        is attached to PortInMsgEventArea

/* STATIC SEMANTICS – A type, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – A derived definition, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A template shall be put underneath the message symbol */
/* STATIC SEMANTICS – A to clause, if existent, shall be put underneath the message symbol */

ReceiveArea ::=
        MessageSymbol
        [ is associated with Type ]
        is associated with ( [ [ DerivedDef AssignmentChar ] TemplateBody ]
                             [ FromClause ] [ PortRedirect ] )
        is attached to InstanceReceiveEventArea
        is attached to PortOutMsgEventArea
```

```
/* STATIC SEMANTICS – A type, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – A derived definition, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A template, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A from clause, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A port redirect, if existent, shall be put underneath the message symbol */

MessageSymbol ::=
```



A.3.10.1 Message-based Communication on Component Instances

```
InstanceSendEventArea ::=
        MessageOutSymbol
        is attached to InstanceAxisSymbol
        is attached to MessageSymbol

MessageOutSymbol ::=
        VoidSymbol
```

The VoidSymbol is a geometric point without spatial extension.

```
InstanceReceiveEventArea ::=
        MessageInSymbol
        is attached to InstanceAxisSymbol
        is attached to MessageSymbol

MessageInSymbol ::=
        VoidSymbol
```

A.3.10.2 Message-based Communication on Port Instances

```
PortInMsgEventArea ::=
        MessageInSymbol
        is attached to PortAxisSymbol
        is attached to MessageSymbol

PortOutMsgEventArea ::=
        MessageOutSymbol
        is attached to PortAxisSymbol
        is attached to MessageSymbol
```

A.3.11 Signature-based Communication

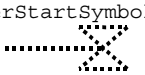
```
NonBlockingCallArea ::=
        MessageSymbol
        is associated with CallKeyword [ Signature ]
        is associated with ( [ DerivedDef AssignmentChar ] TemplateBody
                             [ ToClause ] )
        is attached to InstanceCallEventArea
        is attached to PortCallInEventArea

/* STATIC SEMANTICS – A signature, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – A derived definition, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A template shall be put underneath the message symbol */
/* STATIC SEMANTICS – A to clause, if existent, shall be put underneath the message symbol */

GetcallArea ::=
        MessageSymbol
        is associated with GetcallKeyword [ Signature ]
        is associated with ( [ [ DerivedDef AssignmentChar ] TemplateBody ]
                             [ FromClause ] [ PortRedirectWithParam ] )
        is attached to InstanceGetcallEventArea
        is attached to PortGetcallOutEventArea

/* STATIC SEMANTICS – A signature, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – A derived definition, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A template, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A from clause, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A port redirect, if existent, shall be put underneath the message symbol */

ReplyArea ::=
        MessageSymbol
        is associated with ReplyKeyword [ Signature ]
        is associated with ( [ DerivedDef AssignmentChar ] TemplateBody
                             [ ReplyValue ] [ ToClause ] )
        is attached to InstanceReplyEventArea
        is attached to PortReplyInEventArea
```

```
/* STATIC SEMANTICS – A signature, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – A derived definition, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A template shall be put underneath the message symbol */
/* STATIC SEMANTICS – A reply value, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A to clause, if existent, shall be put underneath the message symbol */

GetreplyWithinCallArea ::=
        MessageSymbol
        is attached to SuspensionRegionSymbol
        is associated with GetreplyKeyword [ Signature ]
        is associated with ( [ [ DerivedDef AssignmentChar ] TemplateBody ]
                             [ ValueMatchSpec ]
                             [ FromClause ] [ PortRedirectWithParam ] )
        is attached to InstanceGetreplyEventArea
        is attached to PortGetreplyOutEventArea

/* STATIC SEMANTICS – A signature, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – A derived definition, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A template, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A value match specification, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A from clause, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A port redirect, if existent, shall be put underneath the message symbol */

GetreplyOutsideCallArea ::=
        MessageSymbol
        is associated with GetreplyKeyword [ Signature ]
        is associated with ( [ [ DerivedDef AssignmentChar ] TemplateBody ]
                             [ ValueMatchSpec ]
                             [ FromClause ] [ PortRedirectWithParam ] )
        is attached to InstanceGetreplyEventArea
        is attached to PortGetreplyOutEventArea

/* STATIC SEMANTICS – A signature, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – A derived definition, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A template, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A value match specification, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A from clause, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A port redirect, if existent, shall be put underneath the message symbol */

RaiseArea ::=
        MessageSymbol
        is associated with RaiseKeyword Signature [ "," Type ]
        is associated with ( [ DerivedDef AssignmentChar ] TemplateBody
                             [ ToClause ] )
        is attached to InstanceRaiseEventArea
        is attached to PortRaiseInEventArea

/* STATIC SEMANTICS – A signature shall be put on top of the message symbol */
/* STATIC SEMANTICS – A n exception type, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – A derived definition, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A template shall be put underneath the message symbol */
/* STATIC SEMANTICS – A to clause, if existent, shall be put underneath the message symbol */

CatchWithinCallArea ::=
        MessageSymbol
        is attached to SuspensionRegionSymbol
        is associated with CatchKeyword Signature [ "," Type ]
        is associated with ( [ [ DerivedDef AssignmentChar ] TemplateBody ]
                             [ FromClause ] [ PortRedirect ] )
        is attached to InstanceCatchEventArea
        is attached to PortCatchOutEventArea

/* STATIC SEMANTICS – A signature shall be put on top of the message symbol */
/* STATIC SEMANTICS – A n exception type, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – A derived definition, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A template, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A from clause, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A port redirect, if existent, shall be put underneath the message symbol */

CatchOutsideCallArea ::=
        MessageSymbol
        is associated with CatchKeyword Signature [ "," Type ]
        is associated with ( [ [ DerivedDef AssignmentChar ] TemplateBody ]
                             [ FromClause ] [ PortRedirect ] )
        is attached to InstanceCatchEventArea
        is attached to PortCatchOutEventArea
```

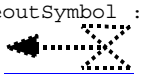
/* STATIC SEMANTICS – A signature shall be put on top of the message symbol */
/* STATIC SEMANTICS – A n exception type, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – A derived definition, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A template, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A from clause, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A port redirect, if existent, shall be put underneath the message symbol */

A.3.11.1 Signature-based Communication on Component Instances

```
InstanceBlockingCallEventArea ::=
        InstanceSendEventArea
        [ is attached to InstanceCallTimerStartArea ]
        is attached to SuspensionRegionSymbol

InstanceCallTimerStartArea ::=
        CallTimerStartSymbol
        is associated with TimerValue
        is attached to InstanceAxisSymbol
        is attached to SuspensionRegionSymbol
        [   is attached to CallTimeoutSymbol3 ]

CallTimerStartSymbol ::=
```



```
InstanceNonBlockingCallEventArea ::=
        InstanceSendEventArea

InstanceGetcallEventArea ::=
        InstanceReceiveEventArea

InstanceReplyEventArea ::=
        InstanceSendEventArea

InstanceGetreplyWithinCallEventArea ::=
        InstanceReceiveEventArea
        is attached to SuspensionRegionSymbol

InstanceGetreplyOutsideCallEventArea ::=
        InstanceReceiveEventArea

InstanceRaiseEventArea ::=
        InstanceSendEventArea

InstanceCatchWithinCallEventArea ::=
        InstanceReceiveEventArea
        is attached to SuspensionRegionSymbol

InstanceCatchTimeoutWithinCallEventArea ::=
        CallTimeoutSymbol
        is attached to SuspensionRegionSymbol
        is attached to InstanceAxisSymbol

CallTimeoutSymbol ::=
```



```
InstanceCatchOutsideCallEventArea ::=
        InstanceReceiveEventArea
```

A.3.11.2 Signature-based Communication on Ports

```
PortGetcallOutEventArea::=
        PortOutMsgEventArea

PortGetreplyOutEventArea::=
        PortOutMsgEventArea

PortCatchOutEventArea::=
        PortOutMsgEventArea

PortCallInEventArea::=
        PortInMsgEventArea

PortReplyInEventArea::=
        PortInMsgEventArea

PortRaiseInEventArea::=
        PortInMsgEventArea
```

A.3.12 Trigger and Check

A.3.12.1 Trigger and Check on Component Instances

```
TriggerArea ::=
        MessageSymbol
        is associated with ( TriggerOpKeyword [ Type ] )
        is associated with ( [ [ DerivedDef AssignmentChar ] TemplateBody ]
                             [ FromClause ] [ PortRedirect ] )
        is attached to ReceiveEventArea
        is attached to PortOutMsgEventArea

/* STATIC SEMANTICS – The trigger keyword shall be put on top of the message symbol */
/* STATIC SEMANTICS – A type, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – A derived definition, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A template, if existent,  shall be put underneath the message symbol */
/* STATIC SEMANTICS – A from clause, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A port redirect, if existent, shall be put underneath the message symbol */

CheckArea ::=
        MessageSymbol
        is associated with ( CheckOpKeyword [ CheckOpInformation ] )
        is associated with CheckData
        is attached to ReceiveEventArea
        is attached to PortOutMsgEventArea

/* STATIC SEMANTICS – The check keyword shall be put on top of the message symbol */
/* STATIC SEMANTICS – The check op information, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – The check data, if existent, shall be put underneath the message symbol */

CheckOpInformation ::=
        Type
        | ( GetCallOpKeyword [ Signature ] )
        | ( GetReplyOpKeyword [ Signature ] )
        | ( CatchOpKeyword Signature [ Type ] )

CheckData ::=
        ( [ [ DerivedDef AssignmentChar ] TemplateBody [ ValueMatchSpec ] ]
          [ FromClause ] [ PortRedirect | PortRedirectWithParam ] )
        | ( [ FromClause ] [ PortRedirectSymbol SenderSpec ] )

/* STATIC SEMANTICS – A value matching specification shall be used in combination with getreply only */
/* STATIC SEMANTICS – A port redirect with parameters shall be used in combination with getcall and getreply only */

InstanceTriggerEventArea ::=
        InstanceReceiveEventArea

InstanceCheckEventArea ::=
        InstanceReceiveEventArea
```

A.3.12.2 Trigger and Check on Port Instances

```
PortTriggerOutEventArea ::=
        PortOutMsgEventArea

PortCheckOutEventArea ::=
        PortOutMsgEventArea
```

A.3.13 Handling of Communication from Any Port

```
InstanceFoundEventArea ::=
        FoundSymbol
        contains FoundEvent
        is attached to InstanceAxisSymbol

/* STATIC SEMANTICS – The label identifier shall be placed inside the circle of the labelling symbol */

FoundEvent ::=
        FoundMessage
    |   FoundTrigger
    |   FoundGetCall
    |   FoundGetReply
    |   FoundCatch
    |   FoundCheck
```

```
FoundMessage ::=
        FoundSymbol
        [ is associated with Type ]
        is associated with ( [ [ DerivedDef AssignmentChar ] TemplateBody ]
                             [ FromClause ] [ PortRedirect ] )
        is attached to InstanceAxisSymbol

/* STATIC SEMANTICS – A type, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – A derived definition, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A template, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A from clause, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A port redirect, if existent, shall be put underneath the message symbol */

FoundTrigger ::=
        FoundSymbol
        is associated with ( TriggerOpKeyword [ Type ] )
        is associated with ( [ [ DerivedDef AssignmentChar ] TemplateBody ]
                             [ FromClause ] [ PortRedirect ] )
        is attached to InstanceAxisSymbol

/* STATIC SEMANTICS – The trigger keyword shall be put on top of the message symbol */
/* STATIC SEMANTICS – A type, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – A derived definition, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A template, if existent,  shall be put underneath the message symbol */
/* STATIC SEMANTICS – A from clause, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A port redirect, if existent, shall be put underneath the message symbol */


FoundGetCall ::=
        FoundSymbol
        is associated with GetcallKeyword [ Signature ]
        is associated with ( [ [ DerivedDef AssignmentChar ] TemplateBody ]
                             [ FromClause ] [ PortRedirectWithParam ] )
        is attached to InstanceAxisSymbol

/* STATIC SEMANTICS – A signature, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – A derived definition, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A template, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A from clause, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A port redirect, if existent, shall be put underneath the message symbol */

FoundGetReply ::=
        FoundSymbol
        is associated with GetreplyKeyword [ Signature ]
        is associated with ( [ [ DerivedDef AssignmentChar ] TemplateBody ]
                             [ ValueMatchSpec ]
                             [ FromClause ] [ PortRedirectWithParam ] )
        is attached to InstanceAxisSymbol

/* STATIC SEMANTICS – A signature, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – A derived definition, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A template, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A value match specification, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A from clause, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A port redirect, if existent, shall be put underneath the message symbol */

FoundCatch ::=
        FoundSymbol
        is associated with CatchKeyword Signature [ "," Type ]
        is associated with ( [ [ DerivedDef AssignmentChar ] TemplateBody ]
                             [ FromClause ] [ PortRedirect ] )
        is attached to InstanceAxisSymbol

/* STATIC SEMANTICS – A signature shall be put on top of the message symbol */
/* STATIC SEMANTICS – An exception type, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – A derived definition, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A template, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A from clause, if existent, shall be put underneath the message symbol */
/* STATIC SEMANTICS – A port redirect, if existent, shall be put underneath the message symbol */

FoundCheck ::=
        FoundSymbol
        is associated with ( CheckOpKeyword [ CheckOpInformation ] )
        is associated with CheckData
        is attached to ReceiveEventArea
        is attached to InstanceAxisSymbol
```

/* STATIC SEMANTICS – The check keyword shall be put on top of the message symbol */
/* STATIC SEMANTICS – The check op information, if existent, shall be put on top of the message symbol */
/* STATIC SEMANTICS – The check data, if existent, shall be put underneath the message symbol */

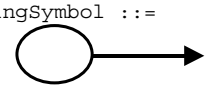
```
FoundSymbol ::=
```

A.3.14 Labelling

```
InstanceLabellingArea ::=
        LabellingSymbol
        contains LabelIdentifier
        is attached to InstanceAxisSymbol
```

/* STATIC SEMANTICS – The label identifier shall be placed inside the circle of the labelling symbol */

```
LabellingSymbol ::=
```

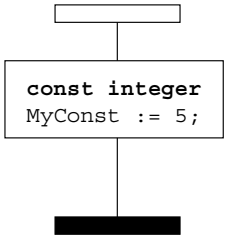
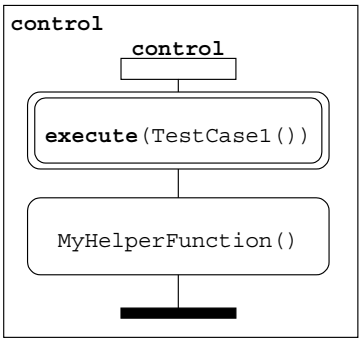
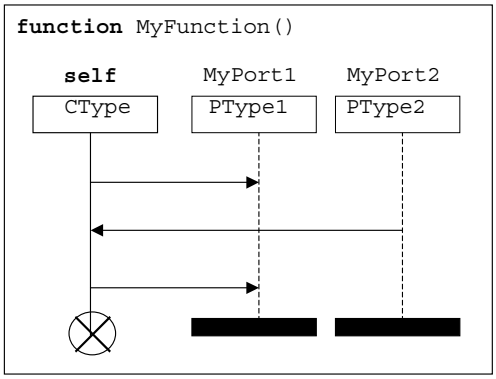
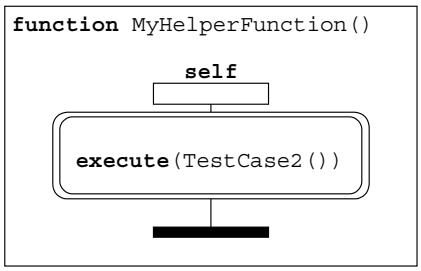


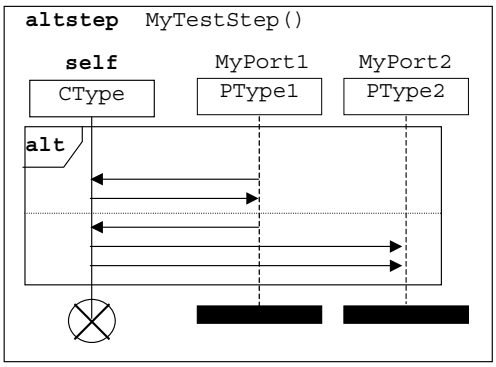
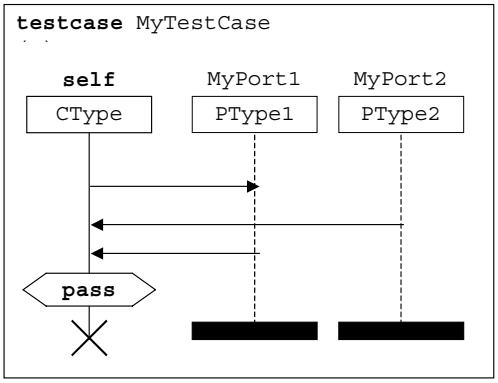
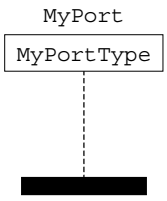
Annex B (informative)

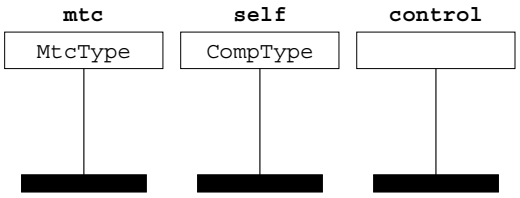
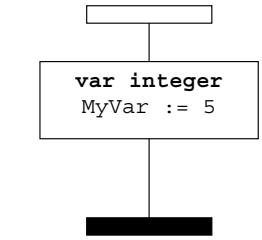
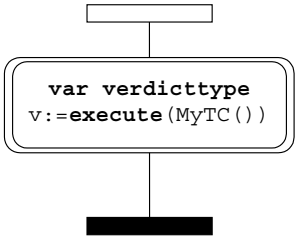
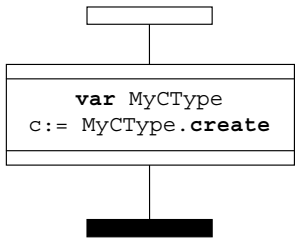
Reference guide for GFT

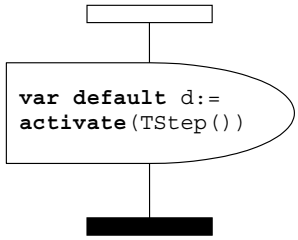
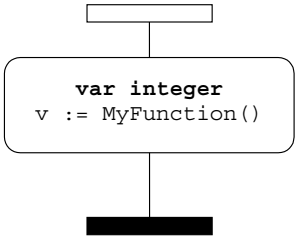
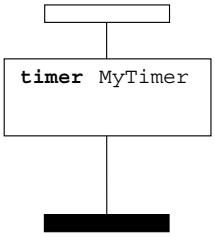
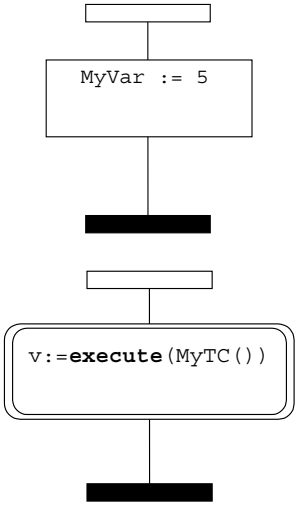
This annex lists the main TTCN-3 language elements and their representation in GFT. For a complete description of the GFT symbols and their use, please refer to the main text.

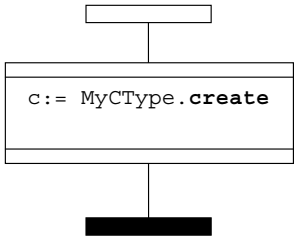
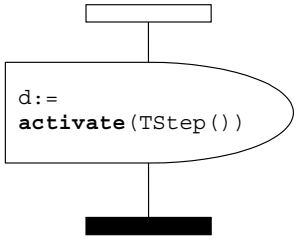
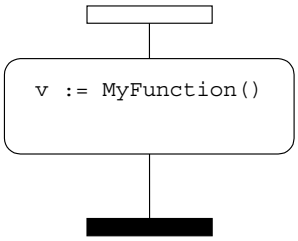
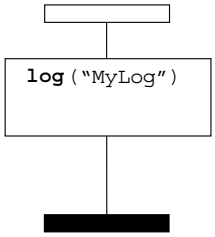
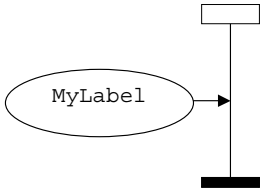
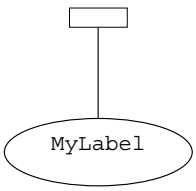
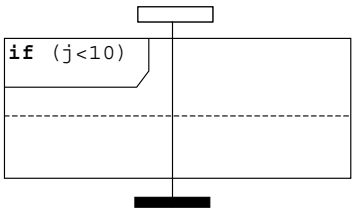
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
Module Definitions			
TTCN-3 module definition	`module`		No special GFT symbol, i.e., the core language or another presentation format may be used.
Import of definitions from other module	`import`		No special GFT symbol, i.e., the core language or another presentation format may be used.
Grouping of definitions	`group`		No special GFT symbol, i.e., the core language or another presentation format may be used.
Data type definitions	`type`		No special GFT symbol, i.e., the core language or another presentation format may be used.
Communication port definitions	`port`		No special GFT symbol, i.e., the core language or another presentation format may be used.
Test component definitions	`component`		No special GFT symbol, i.e., the core language or another presentation format may be used.
Signature definitions	`signature`		No special GFT symbol, i.e., the core language or another presentation format may be used.
External function/constant definitions	`external`		No special GFT symbol, i.e., the core language or another presentation format may be used.

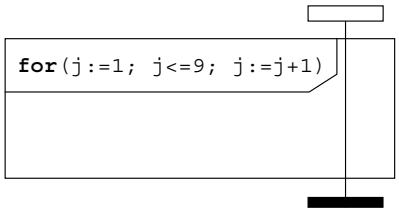
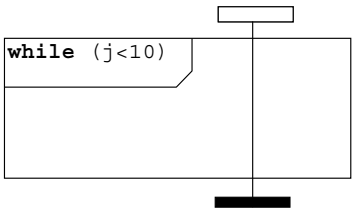
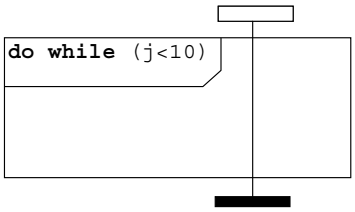
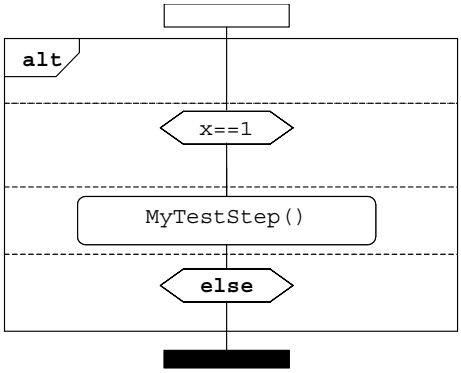
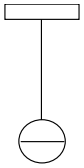
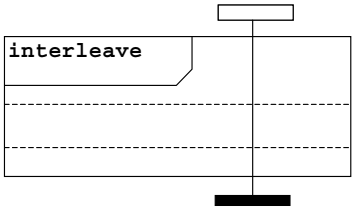
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
Constant definitions	**const**	**const** integer MyConst := 5; 	Textual constant declaration in the header of a control, test case, test step or function diagram. Local constant declaration in an action box.
Data/signature template definitions	**template**		No special GFT symbol, i.e., the core language or another presentation format may be used.
Control definitions	**control**		GFT control diagram represents the control part of a TTCN-3 module.
Function definitions	**function**	 	GFT function diagrams are used to represent functions. GFT function diagrams may be defined to structure the behavior of the control part of a TTCN-3 module.

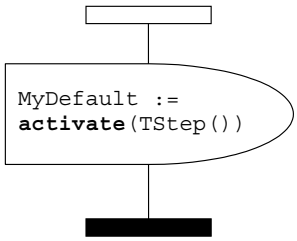
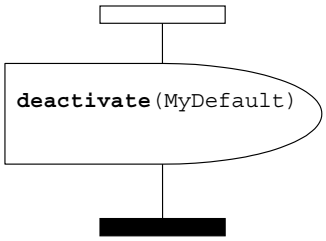
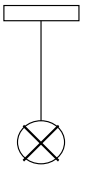
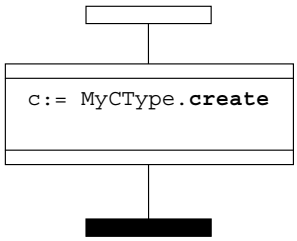
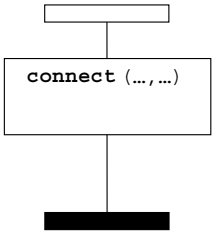
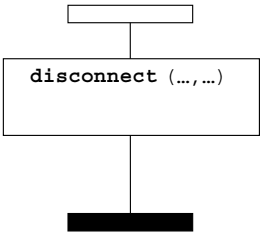
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
Altstep definitions	**altstep**		GFT altstep diagrams are used to represent altsteps.
Test case definitions	**testcase**		GFT test case diagrams are used to represent test cases.
Usage of Component Instances and Ports			
Port instance			A Port in a test case, test step and function diagram is represented by and instance with a dashed instance line. The port name is specified above and the (optional) port type is described within the instance header.

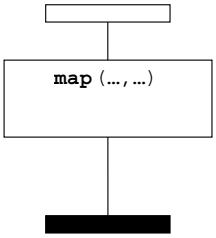
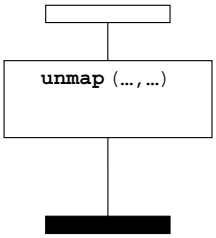
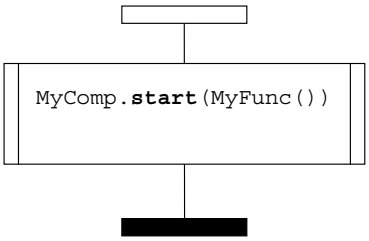
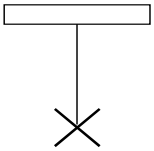
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
Test component instance			An **mtc** instance represents the main test component in a test case diagram. A **self** instance represents a test component in a test step or function diagam. A **control** instance represents the instance that executes the module control part in a control diagram.
Declarations			
Variable declarations	**var**	**var integer** MyVar := 5   	Textual variable declaration in the header of a control, test case, test step or function diagram. Variable declaration in an action box. Variable declaration within a test case execution symbol. Variable declaration within a test component creation symbol.

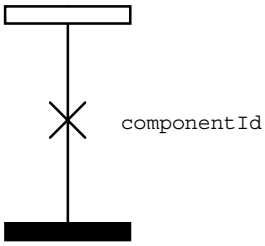
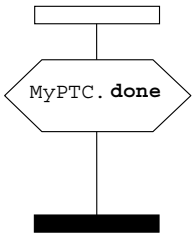
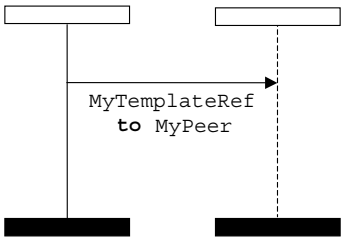
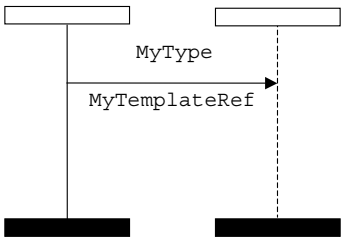
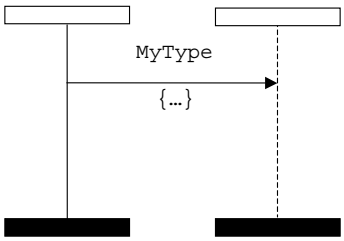
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
		 	Variable declaration within a default activation symbol. Variable declaration within a reference symbol.
Timer declarations	**timer**	**timer** MyTimer 	Textual timer declaration in the header of a control, test case, test step or function diagram. Timer declaration in an action box.
Basic program statements			
Expressions	(…)		No special GFT symbol, i.e., the core language or another presentation format may be used.
Assignments	:=		Assignment in an action box. Assignment within a test case execution symbol.

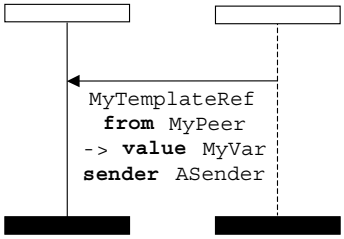
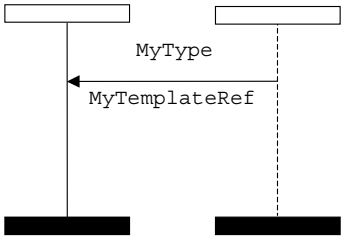
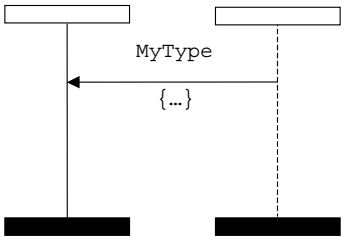
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
		  	Assignment within a test component creation symbol. Assignment within a default activation symbol. Assignment within a reference symbol.
Logging	**log**		The log statement is put into an action box.
Label and Goto	**label**		Definition of a label.
	goto		Go to label.
If-else	**if** (…) {…} **else** {…}		

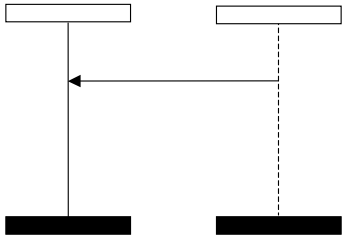
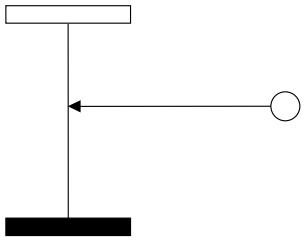
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
For loop	`for (…) {…}`		
While loop	`while (…) {…}`		
Do while loop	`do {…} while (…)`		
Behavioural program statements			
Alternative behaviour	`alt {…}`		
Repeat	`repeat`		To be used within alternative behaviour and test steps.
Interleaved behaviour	`interleave {…}`		

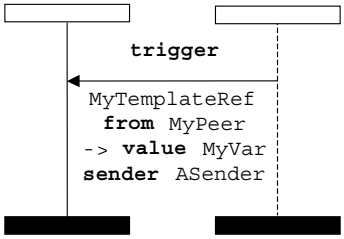
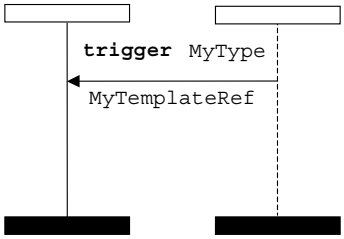
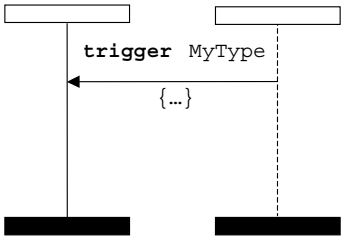
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
Activate a default	**activate**		The activate statement is put into a default symbol.
Deactivate a default	**deactivate**		The deactivate statement is put into a default symbol.
Returning control	**return**		The optional return value is attached to the return symbol.
Configuration operations			
Create parallel test component	**create**		The create statement is put into a test component creation symbol.
Connect component to component	**connect**		The connect statement is put into an action box.
Disconnect two components	**disconnect**		The disconnect statement is put into an action box.

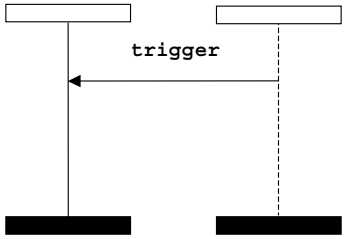
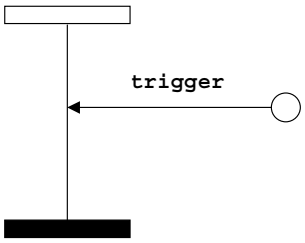
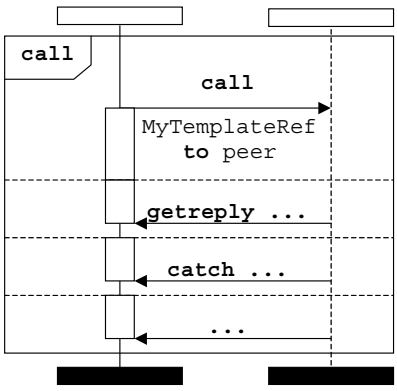
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
Map port to test system interface	**map**		The map statement is put into an action box.
Unmap port from test system interface	**unmap**		The unmap statement is put into an action box.
Get MTC address	**mtc**		No special GFT symbol, used within statements, expressions or as test component identifier.
Get test system interface address	**system**		No special GFT symbol, used within statements or expressions.
Get own address	**self**		No special GFT symbol, used within statements, expressions or as test component identifier.
Start execution of test component	**start**		The start statement is put into a start symbol.
Stop execution of a test component by itself	**stop**		The termination of mtc terminates also all the other test components. Port instances cannot be stopped.

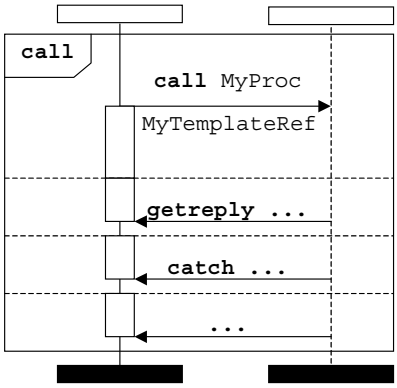
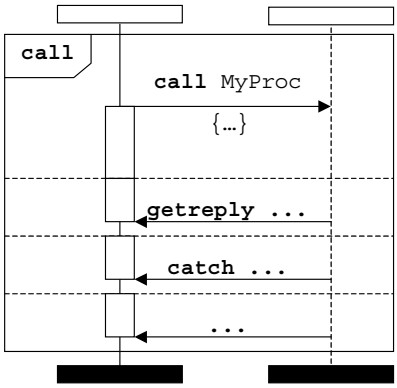
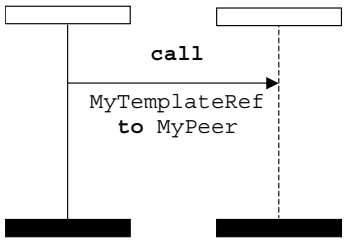
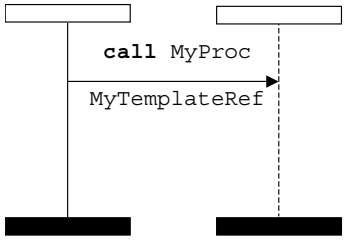
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
of another test component			The component identifier is put near to the stop symbol.
Check termination of a PTC	**running**		No special GFT symbol, used within expressions.
Wait for termination of a PTC	**done**		The done statement is put into a condition symbol.
Communication operations			
Send message	**send**		Send a message defined by a template reference but without type information. The receiver is identified uniquely by the (optional) **to**-directive.
			Send a message defined by a template reference and with type information. An (optional) **to**-directive may be present to identify the peer entity uniquely.
			Send a message defined by an inline template definition. An (optional) **to**-directive may be present to identify the peer entity uniquely.

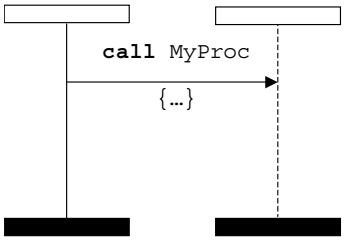
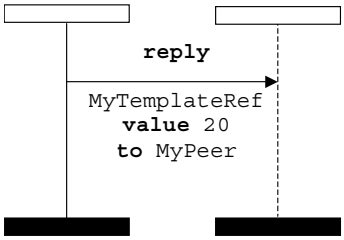
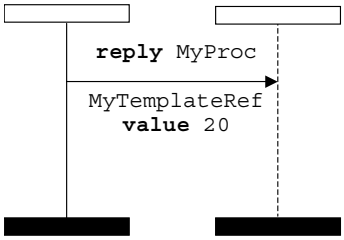
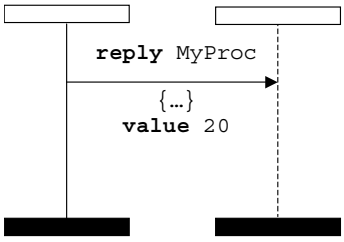
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
Receive message	**receive**		Receive a message with a value defined by a template reference but without type information. The (optional) **from**-directive denotes that the sender of the message shall be identified by variable `MyPeer`. The (optional) **value**-directive assigns received message to variable `MyVar`. The (optional) **sender**-directive retrieves the identifier of the sender and stores it in variable `ASender`.
			Receive a message with a value defined by a template reference and with type information. Optional **from**-, **value**- and **sender**-directives may be present to identify the sender of the message, to assign the message to a variable or to retrieve the identifier of the peer entity.
			Receive a message with a value defined by an inline template definition. Optional **from**-, **value**- and sender-directives may be present to identify the sender of the message, to assign the message to a variable or to retrieve the identifier of the peer entity.

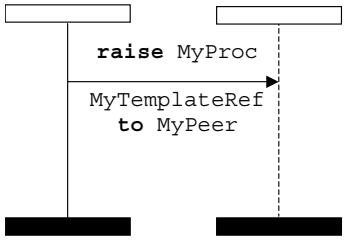
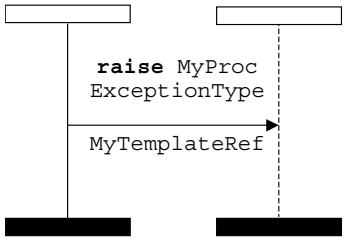
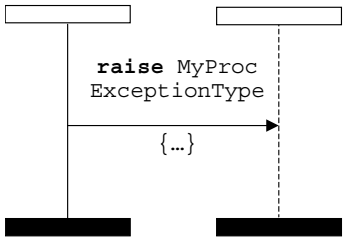
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
			Receive any message (no value and no type is specified). Optional **from**-, **value**- and sender-directives may be present to identify the sender of the message, to assign the message to a variable or to retrieve the identifier of the peer entity.
			Receive any message (no value and no type is specified) from any port. The message value to be received from any port may be restricted by means referring to templates or by using inline templates. Optional **from**-, **value**- and **sender**-directives may be present to identify the sender of the message, to assign the message to a variable or to retrieve the identifier of the peer entity.

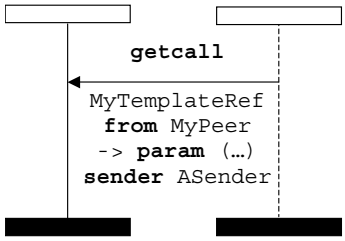
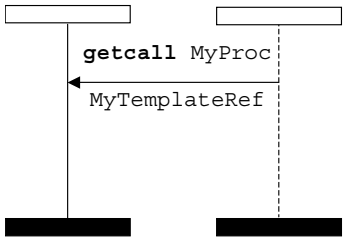
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
Trigger message	**trigger**		Trigger on a message with a value defined by a template reference but without type information. The (optional) **from**-directive denotes that the sender of the message shall be identified by variable `MyPeer`. The (optional) **value**-directive assigns received message to variable `MyVar`. The (optional) **sender**-directive retrieves the identifier of the sender and stores it in variable `ASender`.
			Trigger on a message with a value defined by a template reference and with type information. Optional **from**-, **value**- and **sender**-directives may be present to identify the sender of the message, to assign the message to a variable or to retrieve the identifier of the peer entity.
			Trigger on a message with a value defined by an inline template definition. Optional **from**-, **value**- and **sender**-directives may be present to identify the sender of the message, to assign the message to a variable or to retrieve the identifier of the peer entity.

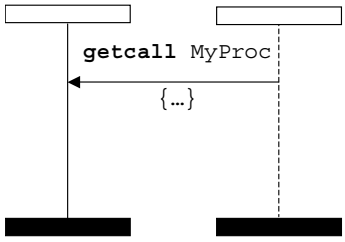
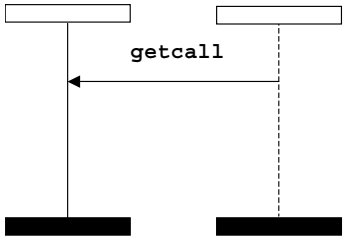
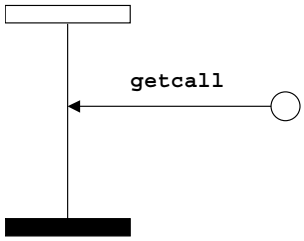
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
			Trigger on any message (no value and no type is specified). Optional **from-**, **value-** and **sender-**directives may be present to identify the sender of the message, to assign the message to a variable (of type **anytype**) and to retrieve the identifier of the peer entity.
			Trigger on any message (no value and no type is specified) from any port. The value of the message that shall cause the trigger from any port may be restricted by means referring to templates or by using inline templates. Optional **from-**, **value-** and **sender-**directives may be present to identify the sender of the message, to assign the message to a variable (of type **anytype**) and to retrieve the identifier of the peer entity.
Invoke blocking procedure call	`call`		Invoking a blocking procedure by using a signature template. The receiver is identified uniquely by the (optional) **to-**directive. The call body, i.e., possible **getreply** and **catch** operations, is shown schematically only.

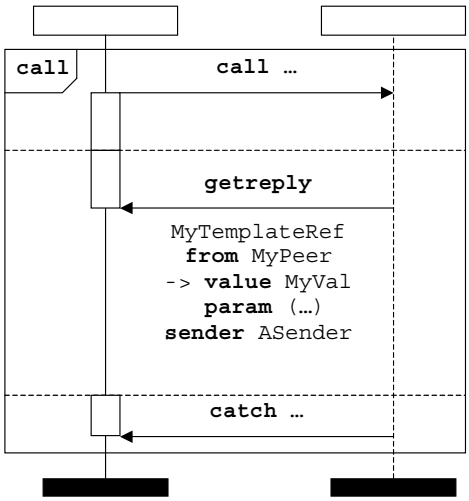
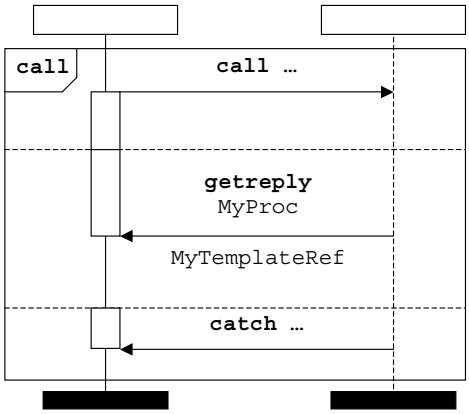
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
		 	Invoking a blocking procedure by using a signature template and signature information. An (optional) **to**-directive may be present to identify the peer entity uniquely. The call body, i.e., possible **getreply** and **catch** operations, is shown schematically only. Invoking a blocking procedure by using an inline template. An (optional) **to**-directive may be present to identify the peer entity uniquely. The call body, i.e., possible **getreply** and **catch** operations, is shown schematically only.
Invoke non-blocking procedure call	`call`	 	Call a remote procedure, the call is defined by a template reference but without signature information. The receiver is identified uniquely by the (optional) **to**-directive. Call the remote procedure `MyProc`. The call is defined by a template reference. An (optional) **to**-directive may be present to identify the peer entity uniquely.

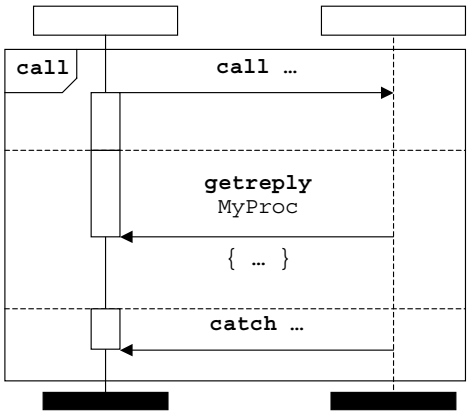
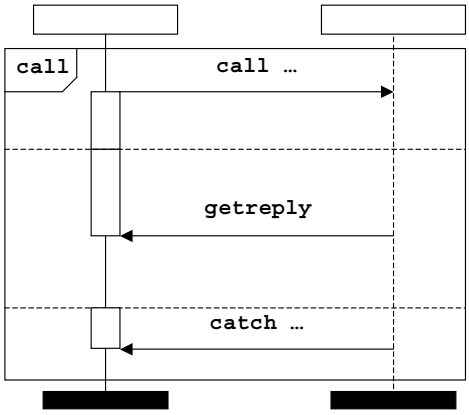
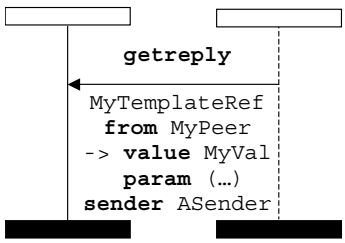
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
			Call the remote procedure MyProc. The call is defined by an inline template. An (optional) **to-** directive may be present to identify the peer entity uniquely.
Reply to procedure call from remote entity	**reply**		Reply to a remote procedure call. The reply is defined by a template reference and the possible return value (**value-** directive). NOTE – The signature information is part of the template definition. The receiver is identified uniquely by the (optional) **to-** directive.
			Reply to a remote procedure call of MyProc. The reply is defined by a template reference and the possible return value (**value-** directive). An (optional) **to-** directive may be present to identify the peer entity uniquely.
			Reply to a remote procedure call of MyProc. The reply is defined by an inline template and the possible return value (**value-** directive). An (optional) **to-** directive may be present to identify the peer entity

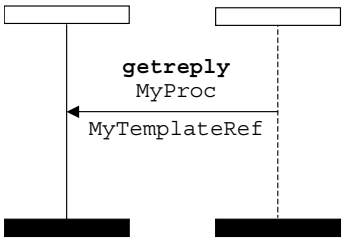
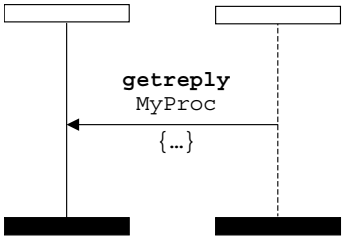
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
			uniquely.
Raise exception (to an accepted call)	**raise**		Raise an exception to an accepted call of MyProc. The exception is defined by a template reference. NOTE – The type of the exception is defined within the template definition. The receiver is identified uniquely by the (optional) **to**-directive.
			Raise an exception to an accepted call of MyProc. The exception is defined by its (optional) type and a template reference. An (optional) **to**-directive may be present to identify the peer entity uniquely.
			Raise an exception to an accepted call of MyProc. The exception is defined by its type and an inline template. An (optional) **to**-directive may be present to identify the peer entity uniquely.

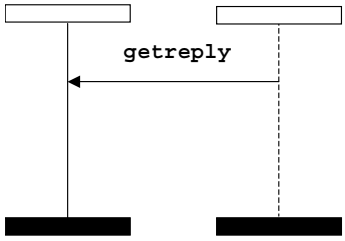
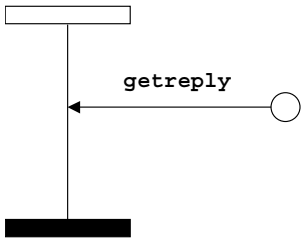
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
Accept procedure call from remote entity	**getcall**		Accept a procedure call from a remote entity. The call signature has to match the conditions defined by the template reference. NOTE – The signature information is part of the template definition. The (optional) **from**-directive denotes that the sender of the call shall be identified by variable `MyPeer`. The (optional) **param**-directive assigns in-parameter values to Variables. The (optional) **sender**-directive retrieves the identifier of the sender and stores it in variable `ASender`.
			Accept a procedure call from a remote entity. The call signature has to match the conditions defined by signature reference and the template reference. Optional **from**-, **param**-and **sender**-directives may be present to identify the sender of the call, to assign the in-parameters to variables or to retrieve the identifier of the peer entity.

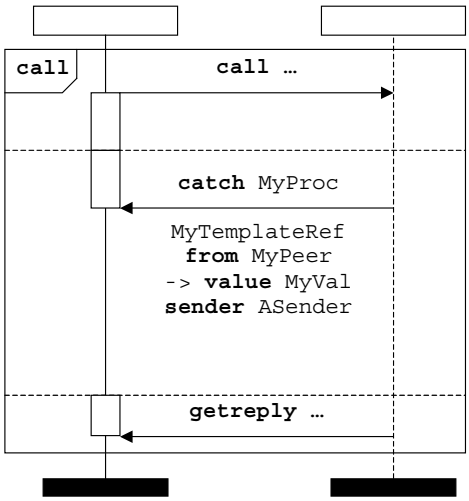
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
			Accept a procedure call from a remote entity. The call signature has to match the conditions defined by signature reference and the inline template definition. Optional **from-**, **param-**and **sender-** directives may be present to identify the sender of the call, to assign the in-parameters to variables or to retrieve the identifier of the peer entity.
			Accept any procedure call from any remote entity. Optional **from-** and **sender-**directives may be present to identify the sender of the call or to retrieve the identifier of the peer entity.
			Accept any procedure call from any remote entity at any port. The call to be received from any port may be restricted by means referring to templates or by using inline templates. Optional **from-**, **param-**and **sender-** directives may be present to identify the sender of the call, to assign the in-parameters to variables or to retrieve the identifier of the peer entity.

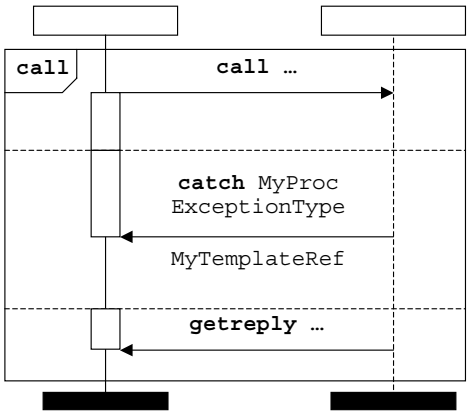
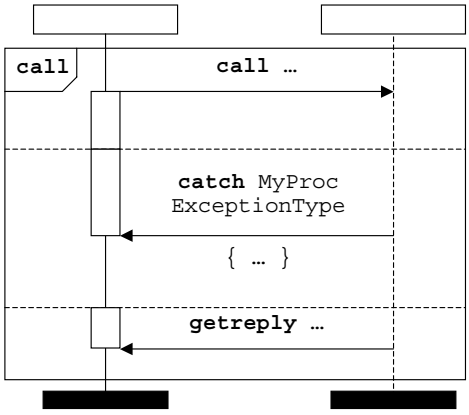
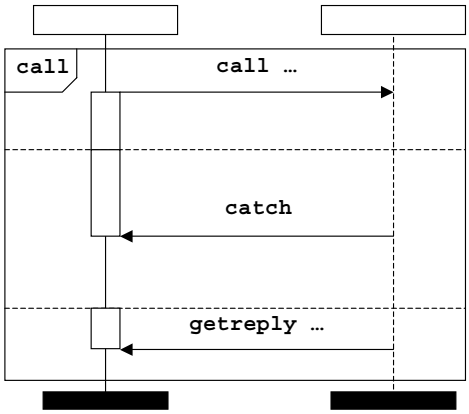
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
Handle response from a previous blocking call	**getreply**	 	Receive a response from a blocking call. The reply has to match the conditions defined by the template reference. NOTE – The signature information is part of the template definition. The (optional) **from**-directive denotes that the sender of the call shall be identified by variable `MyPeer`. The (optional) value-directive assigns the possible return value of the procedure to variable `MyVal`. The (optional) **param**-directive assigns out-parameter values to Variables. The (optional) **sender**-directive retrieves the identifier of the sender and stores it in variable `ASender`. Receive a response from a blocking call. The reply has to match the conditions defined by signature reference and the template reference. Optional **from**-, **value**-, **param**-and **sender**-directives may be present to identify the sender of the reply, to retrieve the return value of the procedure, to assign the in-parameters to variables or to retrieve the identifier of the peer entity.

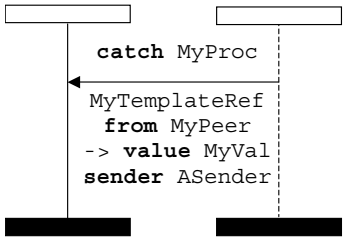
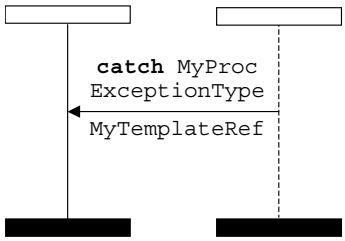
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
		 	Receive a response from a blocking call. The reply has to match the conditions defined by signature reference and the inline template definition. Optional **from-**, **value-**, **param-**and **sender-**directives may be present to identify the sender of the reply, to retrieve the return value of the procedure, to assign the in-parameters to variables or to retrieve the identifier of the peer entity. Accept any response from a blocking call.
Handle response from a previous non-blocking call or independent from a call	`getreply`		Receive a response from a previous call. The reply has to match the conditions defined by the template reference. NOTE – The signature information is part of the template definition. The (optional) **from-** directive denotes that the sender of the call shall be identified by variable `MyPeer`.

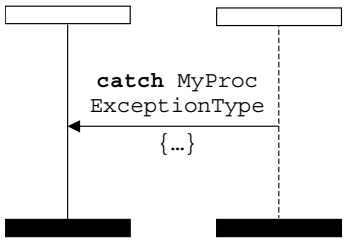
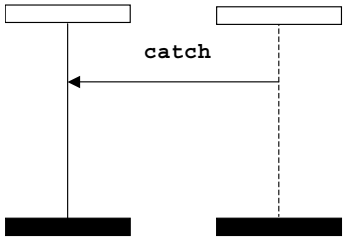
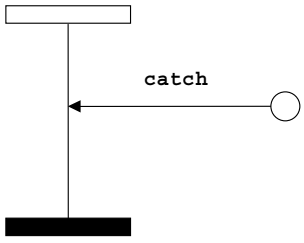
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
			The (optional) value- directive assigns the possible return value of the procedure to variable `MyVal`. The (optional) **param-** directive assigns out-parameter values to Variables. The (optional) **sender-**directive retrieves the identifier of the sender and stores it in variable `ASender`.
			Receive a response from a previous call. The reply has to match the conditions defined by signature reference and the template reference. Optional **from-**, **value-**, **param-**and **sender-** directives may be present to identify the sender of the reply, to retrieve the return value of the procedure, to assign the in-parameters to variables or to retrieve the identifier of the peer entity.
			Receive a response from a previous call. The reply has to match the conditions defined by signature reference and the inline template definition.

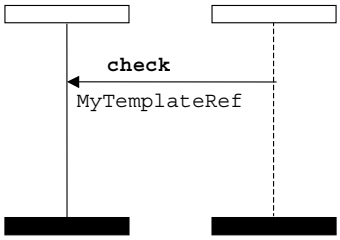
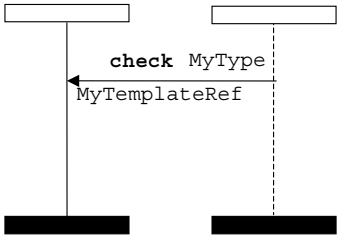
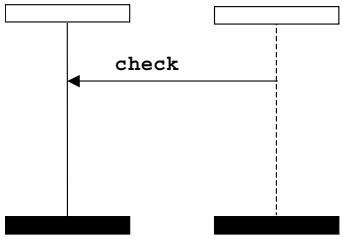
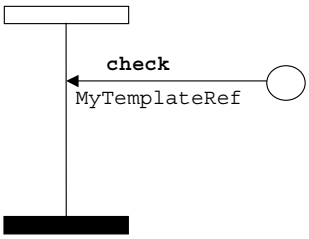
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
		 	Optional **from**-, **value**-, **param**-and **sender**-directives may be present to identify the sender of the reply, to retrieve the return value of the procedure, to assign the in-parameters to variables or to retrieve the identifier of the peer entity. Accept any response from any previous call. Optional **from**- and **sender**-directives may be present to identify the sender of the reply or to retrieve the identifier of the peer entity. Accept any response from any previous call at any port. The reply to be received from any port may be restricted by means referring to templates or by using inline templates. Optional **from**-, **value**-, **param**-and **sender**-directives may be present to identify the sender of the reply, to retrieve the return value of the procedure, to assign the in-parameters to variables or to retrieve the identifier of the peer entity.

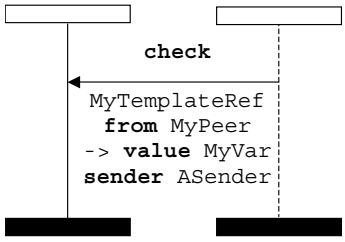
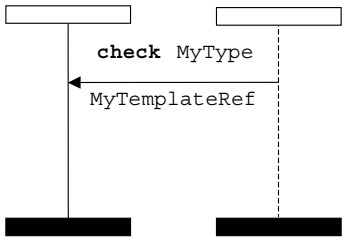
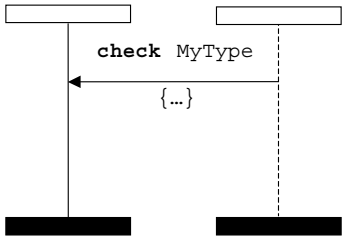
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
Catch exception from a previous blocking call	**catch**		Catch an exception from a previous call. The exception has to match the conditions defined by the template reference. NOTE – The type information is part of the template definition. The (optional) **from**-directive denotes that the sender of the exception shall be identified by variable `MyPeer`. The (optional) value-directive assigns the value of the exception to variable `MyVal`. The (optional) **sender**-directive retrieves the identifier of the sender and stores it in variable `ASender`.

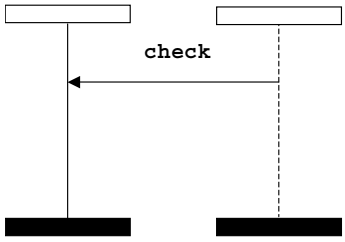
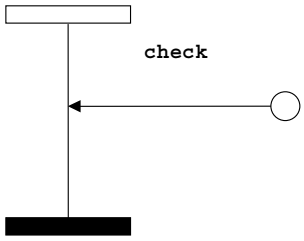
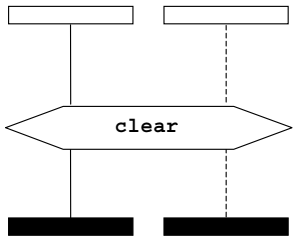
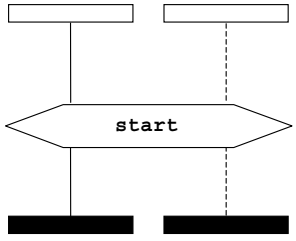
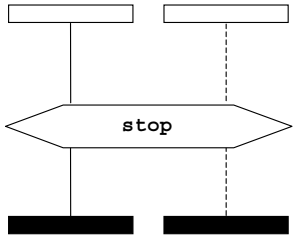
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
			Catch an exception from a previous call. The exception has to match the conditions defined by the exception type and the template reference. Optional **from-**, **value-**, and **sender-** directives may be present to identify the sender of the exception, to retrieve the exception value or to retrieve the identifier of the peer entity.
			Catch an exception from a previous call. The exception has to match the conditions defined by the exception type and the inline template definition. Optional **from-**, **value-**, and **sender-** directives may be present to identify the sender of the exception, to retrieve the exception value or to retrieve the identifier of the peer entity.
			Accept any exception from a blocking call. Optional **from-**, **value-** and **sender-**directives may be present to identify the sender of the exception, to retrieve the exception value (and assign it to a variable of type **anytype**) or to retrieve the identifier of the peer entity.

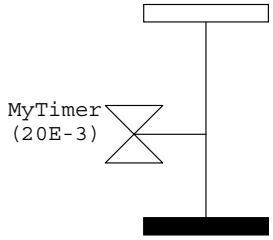
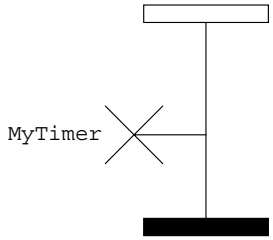
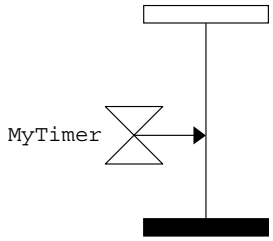
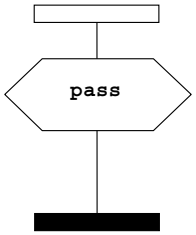
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
Catch exception from a previous non-blocking call or independent from a call	**catch**	 	Catch an exception from a previous call. The exception has to match the conditions defined by the template reference. NOTE – The type information is part of the template definition. The (optional) **from-** directive denotes that the sender of the exception shall be identified by variable `MyPeer`. The (optional) value- directive assigns the value of the exception to variable `MyVal`. The (optional) **sender-**directive retrieves the identifier of the sender and stores it in variable `ASender`. Catch an exception from a previous call. The exception has to match the conditions defined by the exception type and the template reference. Optional **from-**, **value-**, and **sender-** directives may be present to identify the sender of the exception, to retrieve the exception value or to retrieve the identifier of the peer entity.

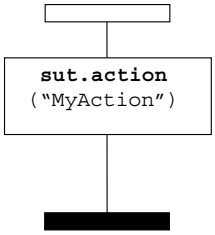
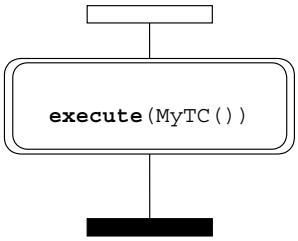
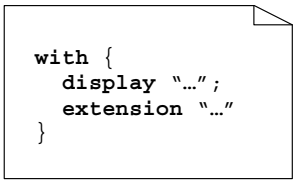
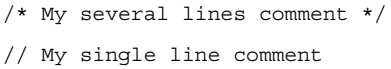
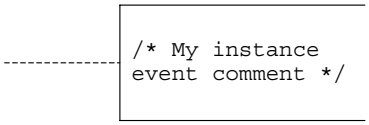
Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
			Catch an exception from a previous call. The exception has to match the conditions defined by the exception type and the inline template definition. Optional **from-**, **value-**, and **sender-** directives may be present to identify the sender of the exception, to retrieve the exception value or to retrieve the identifier of the peer entity.
			Catch any exception from any previous call. Optional **from-**, **value-** and **sender-**directives may be present to identify the sender of the exception, to retrieve the exception value (and assign it to a variable of type **anytype**) or to retrieve the identifier of the peer entity.
			Catch any exception from any previous call at any port. The exception to be received from any port may be restricted by means referring to templates or by using inline templates. Optional **from-**, **value-**, and **sender-** directives may be present to identify the sender of the exception, to retrieve the exception value or to retrieve the identifier of the peer entity.

Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
Check (current) message/call received	**check**		with template, without type.
			with template, with type.
			without template, without type (any message from that port).
		 Can be used also in combination with getcall, getreply, and catch	with template, without type, without port (this message from that port).

Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
Check current message, call, reply or exception	**check**		Check if a message with a value defined by a template reference has been received. The syntax follows the syntax for the reception of messages. NOTE – Check may also be used in combination with **getcall**, **getreply** and **catch**.
			Check if a message with a value defined by a template reference has been received. The syntax follows the syntax for the reception of messages. NOTE – Check may also be used in combination with **getcall**, **getreply** and **catch**.
			Check if a message with a value defined by an inline template definition has been received. The syntax follows the syntax for the reception of messages. NOTE – Check may also be used in combination with **getcall**, **getreply** and **catch**.

Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
		 	Check if any message (no value and no type is specified) has been received. The syntax follows the syntax for the reception of messages. NOTE – Check may also be used in combination with **getcall**, **getreply** and **catch**. Check if any message (no value and no type is specified) has been received at any port. The syntax follows the syntax for the reception of messages. NOTE – Check may also be used in combination with **getcall**, **getreply** and **catch**.
Clear port	`clear`		The clear port statement is put into a condition symbol. The condition shall cover the instance of the port to be cleared only.
Clear and give access to port	`start`		The start port statement is put into a condition symbol. The condition shall cover the instance of the port to be started only.
Stop access (receiving & sending) at port	`stop`		The stop statement is put into a condition symbol. The condition shall cover the instance of the port to be stopped only.

Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
Timer operations			
Start timer	`start`		
Stop timer	`stop`		
Read elapsed time	`read`		No special GFT symbol, used within statements or expressions.
Check if timer running	`running`		No special GFT symbol, used within statements or expressions.
Timeout operation	`timeout`		
Set local verdict	`verdict.set`		The verdict is put into a condition symbol.
Get local verdict	`verdict.get`		No special GFT symbol, used within statements or expressions.

Language element	Associated keyword	GFT symbols, if existent, and typical usage	Note
SUT operations			
Remote action to be done by the SUT	`sut.action`		The action statement is put into an action box.
Execution of test cases			
Execute test case	`execute`		The execute statement is put into a testcase execution symbol.
Attributes			
Definition of attributes for control, testcases, teststeps and functions	`with`		The with statement is put into a text symbol.
Comments			
Comments within text			Can be used wherever text can be placed.
Comments for instance events			Shall be attached to events on a control, test component or port instance.
Comments control, test case, function or test step diagrams			Shall be attached to events on a control, test component or port instance.

Annex C (informative)

Mapping GFT to TTCN-3 Core Language

This annex defines an executable mapping from GFT/gr to the TTCN-3 core language [1]. The purpose behind this activity has been to aid the validation of the graphical grammar and language concepts. It can also be used to verify that the GFT to core and core to GFT mappings are bidirectional.

In order to provide both an abstract and readable implementation we have chosen to use the functional programming language, Standard ML of New Jersey (SML/NJ). SML/NJ is open source and is freely available from Bell Laboratories [http://cm.bell-labs.com/cm/cs/what/smlnj/].

C.1 Approach

The approach for the executable model has been to firstly represent both the core language and GFT grammars as separate SML data types (structures) `cn_type` and `gft_type` respectively. We then define a set of mapping functions that map the GFT data type onto the core data type for each GFT object (i.e., test case diagram, test step diagram, control diagram, and function diagram). The SML signatures for these mapping functions are as follows:

```
gft_testcase_to_cn : gft_type.TestcaseDiagram -> cn_type.TestcaseDef
gft_altstep_to_cn : gft_type.AltstepDiagram -> cn_type.AltstepDef
gft_function_to_cn : gft_type.FunctionDiagram -> cn_type.FunctionDef
gft_control_to_cn  : gft_type.ControlDiagram  -> cn_type.TTCN3Module
```

C.1.2 Overview of SML/NJ

This clause introduces some of the key SML concepts that have been used within the executable mappings.

C.1.2.1 Types and datatypes

SML is a strongly typed language and supports a number of base types, such as integers, strings, lists, etc. It also allows users to define pseudo types using the `type` keyword. For example, `type MyType = string` defines a new type called `MyType`. It also allows the definition of datatypes in which the user may define a set of constructors that give the user the option of defining a choice of sub types or datatypes. For example, `datatype MyDataType = C1 of string | C2 of int` represents the definition of a datatype called `MyDataType`, where C1 represents a constructor function that takes a string argument and returns `MyDataType`, and C2 is a constructor function that takes a string argument and returns `MyDataType`.

C.1.2.2 Functions

Within SML we use functions to define algorithms. The general structure of a function is of the following form: `fun name arguments = expression`, where the arguments may be contained within parentheses, e.g., `fun f (x,y)`, or in a curried fashions e.g., `fun f x y`. For this mapping we tend towards the use of arguments contained within parentheses.

In some cases, the body of the function (the `expression` part) can be defined in the following manner: `fun f = let … in … end`. This form is useful for simplifying the definition of the function by splitting it into sub expressions that are evaluated as needed. The `let…in` clause contains sub expressions, and `in…end` contains the expression representing the function body.

C.1.2.3 Pattern matching

SML has the ability to define patterns representing function arguments. For example, you will find that most of the function definitions contained within the mapping are of the following form.

```
fun f (C1 x) = x_toInt x
 |  f (C2 y) = y_toInt 2
 |  f (_)    = 3
```

where the function `f` takes a single datatype as an argument and returns an integer. In this example, the function defines three separate patterns `(C1 x)`, `(C2 y)` and `(_)`. If the first pattern is matched `(C1 x)` then the value of the variable `x` is passed to the function `x_toInt`, which in turn returns an integer. If the first argument isn't matched then the second is tested and so on. The last pattern `(_)` represents 'any value', meaning that if none of the previous patterns are matched then the expression associated with this pattern is evaluated. In this case an integer with the value 3 is returned.

C.1.2.4 Recursion

Another useful aspect of SML is its ability to represent recursive functions (e.g., functions that operate over a list of elements). For example, below we show a function `f` that takes a list and recursively applies the function `g` to each element within a list, returning an updated list.

```
fun f []     = []
 |  f (h::t) = (g h)::f t
```

In this example you will notice that we use `[]` to denote an empty list, and `(h::t)` to denote the head and tail of a list. Where, '`::`' is a function that prepends a list element to a list of elements. In this case the function defines two patterns. The first pattern matches an empty list and returns an empty list. The second pattern matches a non-empty list. In doing so, it firstly binds the head of the list to the variable `h` and the tail to the variable `t`. Secondly it applies the function `g` to the head of the list and prepends to the result of recursively applying `f` on the tail (the remainer of the list).

C.2 Modelling GFT graphical grammar in SML

We have used the following rules to represent the GFT graphical grammar as an SML type:

- Each non-symbol production within GFT is represented as a SML type or datatype.
- Graphical attachments, as defined by the '*is attach to*' meta operator are modelled using the SML datatype `is_attached_to`. This type allows the two ends of graphical attachment to be represented by a pair of string labels. Note that the mapping functions do not need to know about all attachments, therefore the use of `is_attached_to` is not exhaustive.
- The SML `set` type is use to model the meta type '*set'*, where appropriate.
- We only define GFT types to a level where a core type is referenced.
- The SML `option` type is used to denote optional productions. For example, a production '`[Type]`' would model as `Type option`, where the SML constructors `SOME` and `NONE` are used to represent the value of an optional type.

Below is an example of how we model the GFT production PortOperationArea as a SML type.

```
type PortOperationArea  =   (* Condition contains *)
                            PortOperationText *
                            (* is_attached_to InstanceConditionBeginSymbol *)
                            (* is attached to InstanceConditionEndSymbol *)
                            (* is attached to PortConditionBeginSymbol set *)
                            (* is attached to PortConditionEndSymbol set *)
                            is_attached_to *  (* InstancePortOperationArea *)
                            is_attached_to    (* PortConditionArea *)
```

C.2.1 SML modules

The types and functions needed for the executable mapping are grouped into SML modules, where each module defines a signature and a structure. The signature part defines what types and functions (including signatures) are visible outside the structure in which they are defined. The structure part

contains the types and function definitions. For example, below we show an example of a module containing a signature and structure:

```
signature MyStructSig =
sig
type MyStructType
val f : int -> int (* A function f that takes an integer argument and returns an integer *)
end
structure MyStruct : MyStructSig =
struct
    type MyStructType = int
    fun f x = x+1
end
```

Within the executable GFT to core mapping we define the following SML modules: `gft_type.sml,CN_type.sml, gfttocn.sml,given_functions.sml`.

C.2.2 Function naming and references

Each mapping function follows the name of the GFT type which it is mapping. For example, `p_TimeoutArea` is the mapping function for mapping the TimeoutArea production.

References to types and functions outside of the current scope are prefixed with the name of the structure containing their definition. For example, a reference to the SML type within the cn_type structure is prefixed with `cn_type`, e.g., `cn_type.ConstDef`.

C.2.3 Given functions

To simplify the definition of mapping functions we assume that the following functions are given:

```
val extract_TextLayer_from_testcase       : gft_type.TestcaseBodyArea -> gft_type.TextLayer
val extract_comments_from_TextLayer       : gft_type.TextLayer -> cn_type.TTCN3Comments list
val extract_with_statement_from_TextLayer : gft_type.TextLayer -> cn_type.WithStatement option
val get_attached_SendArea                 : gft_type.is_attached_to * gft_type.ConnectorLayer
                                                    -> gft_type.SendArea
val get_attached_ReceiveArea              : gft_type.is_attached_to * gft_type.ConnectorLayer
                                                    -> gft_type.ReceiveArea
val get_attached_NonBlockingCallArea      : gft_type.is_attached_to * gft_type.ConnectorLayer
                                                    -> gft_type.NonBlockingCallArea
val get_attached_GetcallArea              : gft_type.is_attached_to * gft_type.ConnectorLayer
                                                    -> gft_type.GetcallArea
val get_attached_ReplyArea                : gft_type.is_attached_to * gft_type.ConnectorLayer
                                                    -> gft_type.ReplyArea
val get_attached_GetreplyOutsideCallArea  : gft_type.is_attached_to * gft_type.ConnectorLayer
                                                    -> gft_type.GetreplyOutsideCallArea
val get_attached_RaiseArea                : gft_type.is_attached_to * gft_type.ConnectorLayer
                                                    -> gft_type.RaiseArea
val get_attached_CatchOutsideCallArea     : gft_type.is_attached_to * gft_type.ConnectorLayer
                                            -> gft_type.CatchOutsideCallArea
val get_attached_TriggerArea              : gft_type.is_attached_to * gft_type.ConnectorLayer
                                            -> gft_type.TriggerArea
val get_attached_CheckArea                : gft_type.is_attached_to * gft_type.ConnectorLayer
                                            -> gft_type.CheckArea
val get_attached_starttimer               : gft_type.is_attached_to * gft_type.InstanceEventLayer
                                            -> cn_type.TimerRef
val get_attached_PortOperationArea        : gft_type.is_attached_to * gft_type.ConnectorLayer
                                            -> gft_type.PortOperationArea
val get_attached_InvocationArea           : gft_type.is_attached_to * gft_type.ConnectorLayer
                                            -> gft_type.InvocationArea
val get_attached_IfArea                   : gft_type.InstanceInlineExpressionBeginSymbol *
                                            gft_type.ConnectorLayer
                                            -> gft_type.IfArea
val get_attached_ForArea                  : gft_type.InstanceInlineExpressionBeginSymbol *
                                            gft_type.ConnectorLayer
                                            -> gft_type.ForArea
val get_attached_WhileArea                : gft_type.InstanceInlineExpressionBeginSymbol *
                                            gft_type.ConnectorLayer
                                            -> gft_type.WhileArea
val get_attached_DoWhileArea              : gft_type.InstanceInlineExpressionBeginSymbol *
                                            gft_type.ConnectorLayer
                                            -> gft_type.DoWhileArea
val get_attached_AltArea                  : gft_type.InstanceInlineExpressionBeginSymbol *
                                            gft_type.ConnectorLayer
                                            -> gft_type.AltArea
```

```
val get_attached_InterleaveArea          : gft_type.InstanceInlineExpressionBeginSymbol *
                                           gft_type.ConnectorLayer
                                           -> gft_type.InterleaveArea
val get_number_of_port_instances         : gft_type.InstanceLayer -> int
val get_attached_portcondition_set       : gft_type.is_attached_to * gft_type.PortEventLayer
                                           -> gft_type.is_attached_to list
val get_attached_port                    : gft_type.is_attached_to * gft_type.InstanceLayer
                                           -> cn_type.Port
val GuardOpLayer_to_ConnectorLayer       : gft_type.GuardOpLayer -> gft_type.ConnectorLayer
val UnguardOpLayer_to_ConnectorLayer     : gft_type.UnguardedOpLayer -> gft_type.ConnectorLayer
```

C.2.4 GFT and core SML types

The types are contained in gft_type.sml and CNType.sml. They are not given here, but can be provided on request from ETSI MTS.

C.2.5 GFT to CN mapping functions

```
signature GFTTOCN =
sig
    val gft_testcase_to_cn : gft_type.TestcaseDiagram -> cn_type.TestcaseDef
end (* end of signature *)

structure gfttocn : GFTTOCN =
struct

open gft_type

(*************************************************************************
    p_TextLayer : gft_type.TextLayer -> (cn_type.TTCN3Comment list,cn_type.WithStatement)
*************************************************************************)
fun p_TextLayer TextLayer = (given_functions.extract_comments_from_TextLayer TextLayer,
                             case (given_functions.extract_with_statement_from_TextLayer TextLayer)
                                of NONE   => []
                                |  SOME x => x)

(*************************************************************************
    p_SendArea : SendArea * InstanceLayer
                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_SendArea  ((Type,
                  (DerivedDef, TemplateBody, ToClause),
                  InstanceSendEventArea,
                  PortInMsgEventArea),p)
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.SENDSTATEMENT(
                                        given_functions.get_attached_port(PortInMsgEventArea,p),
                                        ((Type,DerivedDef,TemplateBody),ToClause)
                                )
                        )
                )
(* end of p_SendArea *)

(*************************************************************************
    p_ReceiveArea : ReceiveArea * InstanceLayer
                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_ReceiveArea  ((Type,
                     (SOME (DerivedDef,TemplateBody),FromClause,PortRedirect),
                     InstanceReceiveEventArea,
                     PortOutMsgEventArea),p)
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.RECEIVESTATEMENT(
                                        given_functions.get_attached_port(PortOutMsgEventArea,p),
                                        (SOME(Type,DerivedDef,TemplateBody),FromClause,PortRedirect)
                                )
                        )
                )
 | p_ReceiveArea   ((Type,
                     (NONE,FromClause,PortRedirect),
                     InstanceReceiveEventArea,
                     PortOutMsgEventArea),p)
        =
                cn_type.FUNCTIONSTATEMENT(
```

```
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.RECEIVESTATEMENT(
                                        given_functions.get_attached_port(PortOutMsgEventArea,p),
                                        (NONE,FromClause,PortRedirect)
                                )
                        )
                )
(* end of p_ReceiveArea *)

(*************************************************************************
    p_NonBlockingCallArea : NonBlockingCallArea * Timervalue option * InstanceLayer
                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_NonBlockingCallArea
                  ((Signature,
                  (DerivedDef, TemplateBody, ToClause),
                  InstanceCallEventArea,
                  PortCallInEventArea),TimerValue,p)
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.CALLSTATEMENT(
                                        given_functions.get_attached_port(PortCallInEventArea,p),
                                        (((Signature, DerivedDef, TemplateBody),NONE),ToClause),
                                        NONE (* No statement list *)
                                )
                        )
                )
(* end of p_NonBlockingCallArea *)

(*************************************************************************
    p_GetCallArea : GetCallArea * InstanceLayer
                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_GetCallArea ((Signature,
                  (SOME (DerivedDef,TemplateBody),
                   FromClause,
                   PortRedirectWithParam),
                   InstanceGetCallEventArea,
                   PortGetCallOutEventArea),p)
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                               cn_type.GETCALLSTATEMENT(
                                    given_functions.get_attached_port(PortGetCallOutEventArea,p),
                                        (
                                        SOME (Signature,DerivedDef,TemplateBody),
                                        FromClause,
                                        PortRedirectWithParam
                                        )
                                )
                        )
                )
 |  p_GetCallArea ((Signature,
                  (NONE,
                   FromClause,
                   PortRedirectWithParam),
                   InstanceGetCallEventArea,
                   PortGetCallOutEventArea),p)
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.GETCALLSTATEMENT(
                                       given_functions.get_attached_port(PortGetCallOutEventArea,p),
                                        (
                                        NONE,
                                        FromClause,
                                        PortRedirectWithParam
                                        )
                                )
                        )
                )
(* end of p_GetCallArea *)

(*************************************************************************
    p_ReplyArea : ReplyArea * InstanceLayer
                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_ReplyArea  ((Signature,
                   (DerivedDef, TemplateBody, ReplyValue,ToClause),
```

```
                  InstanceReplyEventArea,
                  PortReplyInEventArea),p)
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.REPLYSTATEMENT(
                                        given_functions.get_attached_port(PortReplyInEventArea,p),
                                        ((Signature,DerivedDef,TemplateBody),ReplyValue,ToClause)
                                )
                        )
                )
(* end of p_ReplyArea *)

(*************************************************************************
    p_GetreplyOutsideCallArea : GetreplyOutsideCallArea * InstanceLayer
                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_GetreplyOutsideCallArea ((Signature,
                                (SOME (DerivedDef,TemplateBody,ValueMatchSpec),
                                 FromClause,
                                 PortRedirectWithParam),
                                 InstanceGetreplyEventArea,
                                 PortGetreplyOutEventArea),p)
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.GETREPLYSTATEMENT(
                                      given_functions.get_attached_port(PortGetreplyOutEventArea,p),
                                        (
                                        SOME ((Signature,DerivedDef,TemplateBody),ValueMatchSpec),
                                        FromClause,
                                        PortRedirectWithParam
                                        )
                                )
                        )
                )
 |  p_GetreplyOutsideCallArea ((Signature,
                                (NONE,
                                 FromClause,
                                 PortRedirectWithParam),
                                 InstanceGetreplyEventArea,
                                 PortGetreplyOutEventArea),p)
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.GETREPLYSTATEMENT(
                                      given_functions.get_attached_port(PortGetreplyOutEventArea,p),
                                        (
                                        NONE,
                                        FromClause,
                                        PortRedirectWithParam
                                        )
                                )
                        )
                )
(* end of p_GetreplyOutsideCallArea *)

(*************************************************************************
    p_RaiseArea : RaiseArea * InstanceLayer
                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_RaiseArea ((Signature,
                  (TemplateInstance,ToClause),
                   InstanceRaiseEventArea,
                   PortGetreplyoutEventArea),p)
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.RAISESTATEMENT(
                                      given_functions.get_attached_port(PortGetreplyoutEventArea,p),
                                        (
                                        Signature,
                                        TemplateInstance,
                                        ToClause
                                        )
                                )
                        )
                )
(* end of p_RaiseArea *)
```

```
(*****************************************************************************
    p_CatchOutsideCallArea : CatchOutsideCallArea * InstanceLayer
                        -> cn_type.FunctionStatementOrDef
*****************************************************************************)
fun p_CatchOutsideCallArea ((SOME Signature,
                        (SOME TemplateInstance,FromClause,PortRedirect),
                         InstanceRaiseEventArea,
                         PortCatchoutEventArea),p)
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.CATCHSTATEMENT(
                                        given_functions.get_attached_port(PortCatchoutEventArea,p),
                                        (
                                        SOME (cn_type.CATCHTEMPLATE (Signature,TemplateInstance)),
                                        FromClause,
                                        PortRedirect
                                        )
                                )
                        )
                )
    | p_CatchOutsideCallArea ((NONE,
                          (NONE,FromClause,PortRedirect),
                         InstanceRaiseEventArea,
                         PortCatchoutEventArea),p)
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.CATCHSTATEMENT(
                                        given_functions.get_attached_port(PortCatchoutEventArea,p),
                                        (
                                        NONE,
                                        FromClause,
                                        PortRedirect
                                        )
                                )
                        )
                )
    | p_CatchOutsideCallArea ((_,
                          (_,FromClause,PortRedirect),
                         InstanceRaiseEventArea,
                         PortCatchoutEventArea),p)
        =
        (print ("Error: Catch must have both a signature and type declared. \n");
                OS.Process.exit OS.Process.failure)
(* end of p_CatchOutsideCallArea *)

(*****************************************************************************
    p_TriggerArea : TriggerArea * InstanceLayer
                        -> cn_type.FunctionStatementOrDef
*****************************************************************************)
fun p_TriggerArea ((Type,
                   (SOME (DerivedDef,TemplateBody),
                   FromClause,
                   PortRedirect),
                   InstanceTriggerEventArea,
                   PortOutMsgEventArea),p)
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.TRIGGERSTATEMENT(
                                        given_functions.get_attached_port(PortOutMsgEventArea,p),
                                        (
                                        SOME (Type,DerivedDef,TemplateBody),
                                        FromClause,
                                        PortRedirect
                                        )
                                )
                        )
                )
 |  p_TriggerArea ((Signature,
                  (NONE,
                   FromClause,
                   PortRedirect),
                   InstanceTriggerEventArea,
                   PortOutMsgEventArea),p)
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.TRIGGERSTATEMENT(
```

```
                                        given_functions.get_attached_port(PortOutMsgEventArea,p),
                                        (
                                        NONE,
                                        FromClause,
                                        PortRedirect
                                        )
                                    )
                                )
                            )
(* End of p_TriggerArea *)

(*************************************************************************
    p_CheckOpInformation : CheckOpInformation * CheckData -> cn_type.PortCheckOp
*************************************************************************)
fun p_CheckOpInformation (NONE,FROMCLAUSEONLY(FromClause,SenderSpec))   =
        SOME (cn_type.CHECKPARAMETER1 (FromClause,SenderSpec))

 |  p_CheckOpInformation (SOME (TYPE Type),
             DERIVEDEF (SOME (DerivedDef,TemplateBody,ValueMatchSpec),
                        FromClause,PortRedirect))      =
               SOME (cn_type.PORTRECEIVEOP
               (
                   SOME (SOME Type,DerivedDef,TemplateBody),
                   FromClause,PortRedirect
               )
           )
 |  p_CheckOpInformation (SOME (TYPE Type),
             DERIVEDEF (NONE,FromClause,PortRedirect))     =
               SOME (cn_type.PORTRECEIVEOP
               (
                   NONE,
                   FromClause,PortRedirect
               )
             )
 |  p_CheckOpInformation (SOME (GETCALLOPKEYWORD Signature),
             DERIVEDEFWPARAM (SOME (DerivedDef,TemplateBody,ValueMatchSpec),
                              FromClause,PortRedirectWithParam)) =
               SOME (cn_type.PORTGETCALLOP
               (
                   SOME (Signature,DerivedDef,TemplateBody),
                   FromClause,PortRedirectWithParam
               )
             )
 |  p_CheckOpInformation (SOME (GETCALLOPKEYWORD Signature),
             DERIVEDEFWPARAM (NONE,
                              FromClause,PortRedirectWithParam)) =
               SOME (cn_type.PORTGETCALLOP
               (
               NONE,FromClause,PortRedirectWithParam
               )
             )
 |  p_CheckOpInformation (SOME (GETREPLYOPKEYWORD Signature),
             DERIVEDEFWPARAM (SOME (DerivedDef,TemplateBody,ValueMatchSpec),
                              FromClause,PortRedirectWithParam)) =
               SOME (cn_type.PORTGETREPLYOP
               (
               SOME ((Signature,DerivedDef,TemplateBody),ValueMatchSpec),
               FromClause,PortRedirectWithParam
               )
             )
 |  p_CheckOpInformation (SOME (GETREPLYOPKEYWORD Signature),
             DERIVEDEFWPARAM (NONE,
                              FromClause,PortRedirectWithParam)) =
               SOME (cn_type.PORTGETREPLYOP
               (
               NONE,FromClause,PortRedirectWithParam
               )
             )
 |  p_CheckOpInformation (SOME (CATCHOPKEYWORD (Signature,Type)),
             DERIVEDEF (SOME (DerivedDef,TemplateBody,ValueMatchSpec),
                        FromClause,PortRedirect)) =
               SOME (cn_type.PORTCATCHOP
               (
               SOME (cn_type.CATCHTEMPLATE (Signature,(Type,DerivedDef,TemplateBody))),
               FromClause,PortRedirect
               )
             )
 |  p_CheckOpInformation (SOME (CATCHOPKEYWORD (Signature,Type)),
             DERIVEDEF (NONE,
                        FromClause,PortRedirect)) =
```

```
                SOME (cn_type.PORTCATCHOP
                (
                NONE,FromClause,PortRedirect
                )
             )
(* end of p_CheckOpInformation *)

(*************************************************************************
    p_CheckArea : CheckArea * InstanceLayer
                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_CheckArea ((CheckOpInformation,
                  CheckData,
                  InstanceReceiveEventArea,
                  PortOutMsgEventArea),p)
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.CHECKSTATEMENT(
                                        given_functions.get_attached_port(PortOutMsgEventArea,p),
                                        (
                                        p_CheckOpInformation (CheckOpInformation,CheckData)
                                        )
                                )
                        )
                )
(* end of p_CheckArea *)

(*************************************************************************
    p_InstanceSendEventArea : InstanceSendEventArea *
                  (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceSendEventArea (InstanceSendEventArea, (_,InstanceLayer,ConnectorLayer,_)) =
        let
                val SendArea =
            given_functions.get_attached_SendArea (InstanceSendEventArea, ConnectorLayer)
        in
                p_SendArea (SendArea,InstanceLayer)
        end
(* end of p_InstanceSendEventArea *)

(*************************************************************************
    p_InstanceReceiveEventArea : InstanceReceiveEventArea *
                    (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceReceiveEventArea (InstanceReceiveEventArea, (_,InstanceLayer,ConnectorLayer,_)) =
        let
                val ReceiveArea =
        given_functions.get_attached_ReceiveArea (InstanceReceiveEventArea, ConnectorLayer)
        in
                p_ReceiveArea (ReceiveArea,InstanceLayer)
        end
(* end of p_InstanceReceiveEventArea *)

(*************************************************************************
    p_InstanceCallEventArea : InstanceCallEventArea*
                  (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                        -> cn_type.FunctionStatementOrDef
*************************************************************************)

fun p_InstanceCallEventArea (INSTANCENONBLOCKINGCALLEVENTAREA InstanceNonBlockingCallEventArea,
                             (_,InstanceLayer,ConnectorLayer,_)) =
        let
                val NonBlockingCallArea =
 given_functions.get_attached_NonBlockingCallArea (InstanceNonBlockingCallEventArea, ConnectorLayer)
        in
                p_NonBlockingCallArea (NonBlockingCallArea,NONE,InstanceLayer)
        end

(* end of p_InstanceCallEventArea *)

(*************************************************************************
    p_InstanceGetCallEventArea : InstanceGetCallEventArea *
                    (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceGetCallEventArea (InstanceReceiveEventArea, (_,InstanceLayer,ConnectorLayer,_)) =
        let
```

```
                val GetCallArea =
        given_functions.get_attached_GetcallArea (InstanceReceiveEventArea, ConnectorLayer)
        in
                p_GetCallArea (GetCallArea,InstanceLayer)
        end

(* end of p_InstanceGetCallEventArea *)

(*************************************************************************
    p_InstanceReplyEventArea : InstanceReplyEventArea*
                        (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceReplyEventArea (InstanceReplyEventArea, (_,InstanceLayer,ConnectorLayer,_)) =
        let
                val ReplyArea =
        given_functions.get_attached_ReplyArea (InstanceReplyEventArea, ConnectorLayer)
        in
                p_ReplyArea (ReplyArea,InstanceLayer)
        end
(* end of p_InstanceReplyEventArea *)

(*************************************************************************
    p_InstanceGetreplyOutsideCallEventArea :
                InstanceGetreplyOutsideCallEventArea *
                    (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                    -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceGetreplyOutsideCallEventArea (InstanceGetreplyOutsideCallEventArea,
(_,InstanceLayer,ConnectorLayer,_)) =
        let
                val GetreplyOutsideCallArea =
                    given_functions.get_attached_GetreplyOutsideCallArea
                    (InstanceGetreplyOutsideCallEventArea, ConnectorLayer)
        in
                p_GetreplyOutsideCallArea (GetreplyOutsideCallArea,InstanceLayer)
        end
(* end of p_InstanceGetCallEventArea *)

(*************************************************************************
    p_InstanceRaiseEventArea : InstanceRaiseEventArea *
                        (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceRaiseEventArea (InstanceRaiseEventArea, (_,InstanceLayer,ConnectorLayer,_)) =
        let
                val RaiseArea =
            given_functions.get_attached_RaiseArea (InstanceRaiseEventArea, ConnectorLayer)
        in
                p_RaiseArea (RaiseArea,InstanceLayer)
        end
(* end of p_InstanceRaiseEventArea *)

(*************************************************************************
    p_InstanceCatchOutsideCallEventArea : InstanceCatchOutsideCallEventArea *
                        (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceCatchOutsideCallEventArea (InstanceCatchOutsideCallEventArea,
(_,InstanceLayer,ConnectorLayer,_)) =
        let
                val CatchOutsideCallArea =
            given_functions.get_attached_CatchOutsideCallArea (InstanceCatchOutsideCallEventArea,
ConnectorLayer)
        in
                p_CatchOutsideCallArea (CatchOutsideCallArea,InstanceLayer)
        end
(* end of p_InstanceCatchOutsideCallEventArea *)

(*************************************************************************
    p_InstanceCatchTimeoutWithinCallEventArea :
        InstanceCatchTimeoutWithinCallEventArea *
                        (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceCatchTimeoutWithinCallEventArea (InstanceCatchTimeoutWithinCallEventArea,
(_,InstanceLayer,ConnectorLayer,_)) =
        let
        in
        cn_type.FUNCTIONSTATEMENT(
```

```
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.CATCHSTATEMENT(
                        given_functions.get_attached_port(PortCatchoutEventArea,p),
                                        (
                                        SOME (cn_type.CATCHTIMEOUT),
                                        NONE,
                                        NONE
                                        )
                                )
                        )
                )
        end
(* end of p_InstanceCatchTimeoutWithinCallEventArea *)

(*************************************************************************
    p_InstanceTriggerEventArea : (InstanceTriggerEventArea *
                        (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceTriggerEventArea (InstanceTriggerEventArea, (_,InstanceLayer,ConnectorLayer,_)) =
        let
                val TriggerArea =
given_functions.get_attached_TriggerArea (InstanceTriggerEventArea, ConnectorLayer)
        in
                p_TriggerArea (TriggerArea,InstanceLayer)
        end

(* end of p_InstanceTriggerEventArea*)

(*************************************************************************
    p_InstanceCheckEventArea : (InstanceCheckEventArea *
                        (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceCheckEventArea (InstanceCheckEventArea, (_,InstanceLayer,ConnectorLayer,_)) =
        let
                val CheckArea =
given_functions.get_attached_CheckArea (InstanceCheckEventArea, ConnectorLayer)
        in
                p_CheckArea (CheckArea,InstanceLayer)
        end

(* end of p_InstanceCheckEventArea*)

(*************************************************************************
    p_TimeoutArea1 : TimeoutArea1
                -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_TimeoutArea1 (NONE)                               = cn_type.FUNCTIONSTATEMENT (
                                cn_type.TIMERSTATEMENTS (
                                    cn_type.TIMEOUTSTATEMENT (cn_type.ANYTIMER)
                                    )
                                )
 |  p_TimeoutArea1 (SOME TimerRef)                      = cn_type.FUNCTIONSTATEMENT (
                                cn_type.TIMERSTATEMENTS (
                                    cn_type.TIMEOUTSTATEMENT (cn_type.TIMEOUTREF TimerRef)
                                    )
                                )
(* end of p_TimeoutArea1 *)

(*************************************************************************
    p_TimeoutArea2 : is_attached_to * InstanceEventLayer
                -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_TimeoutArea2 (is_attached_to,InstanceEventLayer) = cn_type.FUNCTIONSTATEMENT (
                                cn_type.TIMERSTATEMENTS (
                                    cn_type.TIMEOUTSTATEMENT (
                                        cn_type.TIMEOUTREF
                                        (given_functions.get_attached_starttimer
                                            (is_attached_to,InstanceEventLayer))
                                        )
                                    )
                                )
(* end of p_TimeoutArea2 *)

(*************************************************************************
    p_InstanceTimeoutArea : InstanceTimeoutArea
                -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceTimeoutArea (TIMEOUTAREA1 t,_)                            = p_TimeoutArea1 t
```

```
 |  p_InstanceTimeoutArea (TIMEOUTAREA2 t,(InstanceEventLayer,_,_,_))     = p_TimeoutArea2
(t,InstanceEventLayer)
(* end of p_InstanceTimeoutArea *)

(*************************************************************************
    p_TimerStopArea1 : TimerStopArea1
                -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_TimerStopArea1 (NONE)                             =
                        cn_type.FUNCTIONSTATEMENT (
                                cn_type.TIMERSTATEMENTS (
                                    cn_type.STOPTIMERSTATEMENT (cn_type.ALLTIMERS)
                                    )
                                )
 |  p_TimerStopArea1 (SOME TimerRef)                =
                        cn_type.FUNCTIONSTATEMENT (
                                cn_type.TIMERSTATEMENTS (
                                    cn_type.STOPTIMERSTATEMENT (cn_type.STOPTIMERREF TimerRef)
                                    )
                                )
(* end of p_TimerStopArea1 *)

(*************************************************************************
    p_TimerStopArea2 : TimerStopArea2 * InstanceEventLayer
                -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_TimerStopArea2 (is_attached_to,InstanceEventLayer) =
                        cn_type.FUNCTIONSTATEMENT (
                              cn_type.TIMERSTATEMENTS (
                                    cn_type.STOPTIMERSTATEMENT (
                                      cn_type.STOPTIMERREF
                                      (given_functions.get_attached_starttimer
                                        (is_attached_to,InstanceEventLayer))
                                    )
                                )
                        )
(* end of p_TimerStopArea2 *)

(*************************************************************************
    p_InstanceTimerStopArea : InstanceTimerStopArea *
                      (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceTimerStopArea (TIMERSTOPAREA1 t,_)                              = p_TimerStopArea1 t
 |  p_InstanceTimerStopArea (TIMERSTOPAREA2 t,(InstanceEventLayer,_,_,_))  = p_TimerStopArea2
(t,InstanceEventLayer)
(* end of p_InstanceTimerStopArea *)

(*************************************************************************
    p_InstanceTimerStartArea : InstanceTimerStartArea
                -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceTimerStartArea (TimerRef,TimerValue,_) = cn_type.FUNCTIONSTATEMENT (
                                cn_type.TIMERSTATEMENTS (
                                    cn_type.STARTTIMERSTATEMENT (TimerRef,TimerValue)
                                    )
                                )
(* end of p_InstanceTimerStartArea *)

(*************************************************************************
    p_InstanceTimerEventArea : InstanceTimerEventArea *
                      (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceTimerEventArea (INSTANCETIMERSTARTAREA t,_) = p_InstanceTimerStartArea t
 |  p_InstanceTimerEventArea (INSTANCETIMERSTOPAREA t,p)  = p_InstanceTimerStopArea (t,p)
 |  p_InstanceTimerEventArea (INSTANCETIMEOUTAREA t,p)    = p_InstanceTimeoutArea (t,p)
(* end of p_InstanceTimerEventArea *)

(*************************************************************************
    p_FoundMessage : FoundMessage -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_FoundMessage (Type,(SOME (DerivedDef,TemplateBody),FromClause,PortRedirect))
      =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.RECEIVESTATEMENT(

"any",(SOME(Type,DerivedDef,TemplateBody),FromClause,PortRedirect)
                                )
```

```
                        )
                )
 | p_FoundMessage (Type,(NONE,FromClause,PortRedirect))
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.RECEIVESTATEMENT(
                                        "any",(NONE,FromClause,PortRedirect)
                                )
                        )
                )

(*************************************************************************
    p_FoundTrigger : FoundTrigger -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_FoundTrigger (Type,(SOME (DerivedDef,TemplateBody),FromClause,PortRedirect))
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.TRIGGERSTATEMENT(
                                        "any",
                                        (
                                        SOME (Type,DerivedDef,TemplateBody),
                                        FromClause,
                                        PortRedirect
                                        )
                                )
                        )
                )
 |  p_FoundTrigger (Signature,(NONE,FromClause,PortRedirect))
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.TRIGGERSTATEMENT(
                                        "any",(NONE,FromClause,PortRedirect)
                                )
                        )
                )
(* end of p_FoundTrigger *)

(*************************************************************************
    p_FoundGetCall : FoundGetCall -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_FoundGetCall (Signature,(SOME (DerivedDef,TemplateBody),FromClause,PortRedirectWithParam))
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.GETCALLSTATEMENT(
                                        "any",
                                        (
                                        SOME (Signature,DerivedDef,TemplateBody),
                                        FromClause,
                                        PortRedirectWithParam
                                        )
                                )
                        )
                )
 |  p_FoundGetCall (Signature,(NONE,FromClause,PortRedirectWithParam))
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.GETCALLSTATEMENT(
                                        "any",(NONE,FromClause,PortRedirectWithParam)
                                )
                        )
                )
(* end of p_FoundGetCall *)

(*************************************************************************
    p_FoundGetReply : FoundGetReply -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_FoundGetReply (Signature,
                     (SOME (DerivedDef,TemplateBody,ValueMatchSpec),
                     FromClause,
                     PortRedirectWithParam))
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.GETREPLYSTATEMENT(
                                        "any",
```

```
                                        (
                                        SOME ((Signature,DerivedDef,TemplateBody),ValueMatchSpec),
                                        FromClause,
                                        PortRedirectWithParam
                                        )
                                )
                        )
                )
 |  p_FoundGetReply (Signature,(NONE,FromClause,PortRedirectWithParam))
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.GETREPLYSTATEMENT(
                                        "any",(NONE,FromClause,PortRedirectWithParam)
                                )
                        )
                )
(* end of p_FoundGetReply *)
(*************************************************************************
    p_FoundCatch : FoundCatch -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_FoundCatch (SOME Signature,(SOME TemplateInstance,FromClause,PortRedirect))
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.CATCHSTATEMENT(
                                        "any",
                                        (
                                        SOME (cn_type.CATCHTEMPLATE (Signature,TemplateInstance)),
                                        FromClause,
                                        PortRedirect
                                        )
                                )
                        )
                )
    | p_FoundCatch (NONE,(NONE,FromClause,PortRedirect))
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.CATCHSTATEMENT(
                                        "any",(NONE,FromClause,PortRedirect)
                                )
                        )
                )
    | p_FoundCatch (_,(_,FromClause,PortRedirect))
        =
        (print ("Error: Catch must have both a signature and type declared. \n");
                OS.Process.exit OS.Process.failure)
(* end of p_FoundCatch *)

(*************************************************************************
    p_FoundCheck : FoundCheck -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_FoundCheck ((CheckOpInformation,CheckData):FoundCheck)
        =
                cn_type.FUNCTIONSTATEMENT(
                        cn_type.COMMUNICATIONSTATEMENTS (
                                cn_type.CHECKSTATEMENT(
                                        "any",(p_CheckOpInformation (CheckOpInformation,CheckData))
                                )
                        )
                )
(* end of p_FoundCheck *)

(*************************************************************************
    p_FoundEvent : FoundEvent -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_FoundEvent (FOUNDMESSAGE FoundMessage)    = p_FoundMessage FoundMessage
 |  p_FoundEvent (FOUNDTRIGGER FoundTrigger)    = p_FoundTrigger FoundTrigger
 |  p_FoundEvent (FOUNDGETCALL FoundGetCall)    = p_FoundGetCall FoundGetCall
 |  p_FoundEvent (FOUNDGETREPLY FoundGetReply)  = p_FoundGetReply FoundGetReply
 |  p_FoundEvent (FOUNDCATCH FoundCatch)        = p_FoundCatch FoundCatch
 |  p_FoundEvent (FOUNDCHECK FoundCheck)        = p_FoundCheck FoundCheck
(* end of p_FoundEvent *)

(*************************************************************************
    p_InstanceFoundEventArea : InstanceFoundEventArea -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceFoundEventArea FoundEvent = p_FoundEvent FoundEvent
(* end of p_InstanceFoundEventArea *)
```

```
(*************************************************************************
    p_ActionStatement : ActionStatement -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_ActionStatement (SUTSTATEMENTS SUTStatements)         = cn_type.FUNCTIONSTATEMENT (
                                    cn_type.SUTSTATEMENTS SUTStatements)
 |  p_ActionStatement (CONNECTSTATEMENT ConnectStatement)   = cn_type.FUNCTIONSTATEMENT (
                                    cn_type.CONFIGURATIONSTATEMENTS (
                                        cn_type.CONNECTSTATEMENT ConnectStatement))
 |  p_ActionStatement (MAPSTATEMENT MapStatement)       = cn_type.FUNCTIONSTATEMENT (
                                    cn_type.CONFIGURATIONSTATEMENTS (
                                        cn_type.MAPSTATEMENT MapStatement))
 |  p_ActionStatement (DISCONNECTSTATEMENT DisconnectStatement) = cn_type.FUNCTIONSTATEMENT (
                                    cn_type.CONFIGURATIONSTATEMENTS (
                                        cn_type.DISCONNECTSTATEMENT DisconnectStatement))
 |  p_ActionStatement (UNMAPSTATEMENT UnmapStatement)       = cn_type.FUNCTIONSTATEMENT (
                                    cn_type.CONFIGURATIONSTATEMENTS (
                                        cn_type.UNMAPSTATEMENT UnmapStatement))
 |  p_ActionStatement (CONSTDEF ConstDef)           = cn_type.FUNCTIONLOCALDEF ConstDef
 |  p_ActionStatement (VARINSTANCE VarInstance)         = cn_type.FUNCTIONLOCALINST (
                                    cn_type.VARINSTANCE VarInstance)
 |  p_ActionStatement (TIMERINSTANCE TimerInstance)     = cn_type.FUNCTIONLOCALINST (
                                    cn_type.TIMERINSTANCE TimerInstance)
 |  p_ActionStatement (ASSIGNMENT Assignment)           = cn_type.FUNCTIONSTATEMENT (
                                    cn_type.BASICSTATEMENTS (
                                        cn_type.BASICASSIGNMENT Assignment))
 |  p_ActionStatement (LOGSTATEMENT LogStatement)       = cn_type.FUNCTIONSTATEMENT (
                                    cn_type.BASICSTATEMENTS (
                                        cn_type.LOGSTATEMENT LogStatement))
 |  p_ActionStatement (LOOPCONSTRUCT LoopConstruct)     = cn_type.FUNCTIONSTATEMENT (
                                    cn_type.BASICSTATEMENTS (
                                        cn_type.LOOPCONSTRUCT LoopConstruct))
 |  p_ActionStatement (CONDITIONALCONSTRUCT ConditionalConstruct)=cn_type.FUNCTIONSTATEMENT (
                                    cn_type.BASICSTATEMENTS (
                                        cn_type.CONDITIONALCONSTRUCT ConditionalConstruct))
(* end of p_ActionStatement *)

(*************************************************************************
    p_InstanceActionArea : InstanceActionArea -> cn_type.FunctionStatementOrDef list
*************************************************************************)
fun p_InstanceActionArea InstanceActionArea = map p_ActionStatement InstanceActionArea
(* end of p_InstanceActionArea *)

(*************************************************************************
    p_InstanceLabellingArea : InstanceLabellingArea -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceLabellingArea LabelIdentifier = cn_type.FUNCTIONSTATEMENT (
                            cn_type.BEHAVIOURSTATEMENTS (
                                cn_type.LABELSTATEMENT LabelIdentifier
                                )
                            )
(* end of p_InstanceLabellingArea *)

(*************************************************************************
    p_InstanceDoneArea : InstanceDoneArea -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceDoneArea DoneStatement =
        cn_type.FUNCTIONSTATEMENT (
            cn_type.CONFIGURATIONSTATEMENTS (
                cn_type.DONESTATEMENT DoneStatement))

(* end of p_InstanceDoneArea *)

(*************************************************************************
    p_SetVerdictArea : SetVerdictArea -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_SetVerdictArea (SETVERDICTKEYWORD SingleExpression) =
                cn_type.FUNCTIONSTATEMENT (
            cn_type.VERDICTSTATEMENTS SingleExpression)
 |  p_SetVerdictArea (PASSKEYWORD)      =
                cn_type.FUNCTIONSTATEMENT (
            cn_type.VERDICTSTATEMENTS "pass")
 |  p_SetVerdictArea (FAILKEYWORD)      =
                cn_type.FUNCTIONSTATEMENT (
            cn_type.VERDICTSTATEMENTS "fail")
 |  p_SetVerdictArea (INCONCKEYWORD)    =
                cn_type.FUNCTIONSTATEMENT (
            cn_type.VERDICTSTATEMENTS "inconc")
 |  p_SetVerdictArea (NONEKEYWORD)      =
                cn_type.FUNCTIONSTATEMENT (
```

```
            cn_type.VERDICTSTATEMENTS "none")
(* end of p_SetVerdictArea *)

(*****************************************************************************
    p_InstanceSetVerdictArea : InstanceSetVerdictArea -> cn_type.FunctionStatementOrDef
*****************************************************************************)
fun p_InstanceSetVerdictArea SetVerdictArea = p_SetVerdictArea SetVerdictArea
(* end of p_InstanceSetVerdictArea *)

(*****************************************************************************
    p_PortOperationArea : PortOperationArea -> cn_type.FunctionStatementOrDef
*****************************************************************************)
fun p_PortOperationArea ((PortOperationText,InstancePortOperationArea,PortConditionArea),
                         (PortEventLayer,InstanceLayer)) =
        let
                val number_of_ports   = given_functions.get_number_of_port_instances InstanceLayer
                val attached_port_list= given_functions.get_attached_portcondition_set
(PortConditionArea,PortEventLayer)

                fun map_operation (PortOrAll,STARTKEYWORD)   =
                        cn_type.STARTSTATEMENT (PortOrAll,cn_type.STARTKEYWORD)
                 |  map_operation (PortOrAll,STOPKEYWORD)    =
                        cn_type.STOPSTATEMENT (PortOrAll,cn_type.STOPKEYWORD)
                 |  map_operation (PortOrAll,CLEAROPKEYWORD) =
                        cn_type.CLEARSTATEMENT (PortOrAll,cn_type.CLEAROPKEYWORD)
        in
                (* If condition symbol covers all ports then perform operation on all ports *)
                if number_of_ports = (List.length attached_port_list) then
                        cn_type.FUNCTIONSTATEMENT (
                                cn_type.COMMUNICATIONSTATEMENTS (
                                        map_operation ("all",PortOperationText)
                                )
                        )

                (* A port operation can either be connected to one or all ports *)
                else if (List.length attached_port_list) > 1 then
                        (print ("Error: Port operation is attached to incorrect number of
ports.\n");
                         OS.Process.exit OS.Process.failure)

                (* Operation to be performed on a single port *)
                else if (List.length attached_port_list) = 1 then
                        cn_type.FUNCTIONSTATEMENT (
                                cn_type.COMMUNICATIONSTATEMENTS (
                                       map_operation (hd attached_port_list,PortOperationText)
                                )
                        )

                (* Condition symbol must be attached to at least on port instance *)
                else
                        (print ("Error: Port operation is not attached to any port instance(s).\n");
                         OS.Process.exit OS.Process.failure)
        end
(* end of p_PortOperationArea *)

(*****************************************************************************
    p_InstancePortOperationArea : InstancePortOperationArea *

(InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                        -> cn_type.FunctionStatementOrDef
*****************************************************************************)
fun p_InstancePortOperationArea
(InstancePortOperationArea,(_,InstanceLayer,ConnectorLayer,PortEventLayer)) =
        let
                val PortOperationArea =
            given_functions.get_attached_PortOperationArea (InstancePortOperationArea,
ConnectorLayer)
        in
                p_PortOperationArea (PortOperationArea,(PortEventLayer,InstanceLayer))
        end
(* end of p_InstancePortOperationArea *)

(*****************************************************************************
    p_InstanceConditionArea : InstanceConditionArea *
                                (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                        -> cn_type.FunctionStatementOrDef
*****************************************************************************)
fun p_InstanceConditionArea ((INSTANCEDONEAREA InstanceDoneArea),_) =
                         p_InstanceDoneArea InstanceDoneArea
 |  p_InstanceConditionArea ((INSTANCESETVERDICTAREA InstanceSetVerdictArea),_) =
```

```
                        p_InstanceSetVerdictArea InstanceSetVerdictArea
 |  p_InstanceConditionArea ((INSTANCEPORTOPERATIONAREA InstancePortOperationArea),p) =
                        p_InstancePortOperationArea  (InstancePortOperationArea,p)
(* end of p_InstanceConditionArea *)

(*************************************************************************
    p_InvocationArea : InvocationArea -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InvocationArea (INVOCATIONFUNCTIONINSTANCE FunctionInstance,_) = cn_type.FUNCTIONSTATEMENT (
                                    cn_type.BEHAVIOURSTATEMENTS (
                                        cn_type.FUNCTIONINSTANCE FunctionInstance))
 |  p_InvocationArea (INVOCATIONALTSTEPINSTANCE AltstepInstance,_) = cn_type.FUNCTIONSTATEMENT (
                                    cn_type.BEHAVIOURSTATEMENTS (
                                        cn_type.ALTSTEPINSTANCE AltstepInstance))
 |  p_InvocationArea (INVOCATIONCONSTDEF ConstDef,_)             = cn_type.FUNCTIONLOCALDEF ConstDef
 |  p_InvocationArea (INVOCATIONVARINSTANCE VarInstance,_)       = cn_type.FUNCTIONLOCALINST (
                                    cn_type.VARINSTANCE VarInstance)
 |  p_InvocationArea (INVOCATIONASSIGNMENT Assignment,_)             = cn_type.FUNCTIONSTATEMENT (
                                    cn_type.BASICSTATEMENTS (
                                        cn_type.BASICASSIGNMENT Assignment))
(* end of p_InvocationArea *)

(*************************************************************************
    p_InstanceInvocationArea : InstanceInvocationArea *
                                  (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceInvocationArea (InstanceInvocationArea,(_,_,ConnectorLayer,_)) =
        let
                val InvocationArea =
            given_functions.get_attached_InvocationArea(InstanceInvocationArea, ConnectorLayer)
        in
                p_InvocationArea InvocationArea
        end
(* end of p_InstanceInvocationArea *)

(*************************************************************************
    p_InstanceDefaultHandlingArea : InstanceInvocationArea
                                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceDefaultHandlingArea (DEACTIVATESTATEMENT DeactivateStatement)     =
cn_type.FUNCTIONSTATEMENT (
                    cn_type.BEHAVIOURSTATEMENTS (
                        cn_type.DEACTIVATESTATEMENT DeactivateStatement))
 |  p_InstanceDefaultHandlingArea (DEFAULTCONSTDEF ConstDef)             =
                    cn_type.FUNCTIONLOCALDEF ConstDef
 |  p_InstanceDefaultHandlingArea (DEFAULTVARINSTANCE VarInstance)      =
                    cn_type.FUNCTIONLOCALINST (
                        cn_type.VARINSTANCE VarInstance)
 |  p_InstanceDefaultHandlingArea (DEFAULTASSIGNMENT Assignment)        =
                    cn_type.FUNCTIONSTATEMENT (
                        cn_type.BASICSTATEMENTS (
                            cn_type.BASICASSIGNMENT Assignment))
(* end of p_InstanceDefaultHandlingArea *)

(*************************************************************************
    p_InstanceComponentCreateArea : InstanceComponentCreateArea
                                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceComponentCreateArea (CREATIONCONSTDEF ConstDef)           =
                cn_type.FUNCTIONLOCALDEF ConstDef
 |  p_InstanceComponentCreateArea (CREATIONVARINSTANCE VarInstance)     =
                cn_type.FUNCTIONLOCALINST (
                        cn_type.VARINSTANCE VarInstance)
 |  p_InstanceComponentCreateArea (CREATIONASSIGNMENT Assignment)       =
                cn_type.FUNCTIONSTATEMENT (
                        cn_type.BASICSTATEMENTS (
                                cn_type.BASICASSIGNMENT Assignment))
(* end of p_InstanceComponentCreateArea *)

(*************************************************************************
    p_InstanceComponentStartArea : InstanceComponentStartArea
                                        -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceComponentStartArea StartTCStatement =
                    cn_type.FUNCTIONSTATEMENT (
                            cn_type.CONFIGURATIONSTATEMENTS (
                                cn_type.STARTTCSTATEMENT StartTCStatement))
(* end of p_InstanceComponentStartArea *)
```

```
(*************************************************************************
    p_InstanceEventArea : InstanceEventArea *
(InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                              -> cn_type.FunctionStatementOrDef list
*************************************************************************)
fun p_InstanceEventArea (INSTANCESENDEVENTAREA (InstanceSendEventArea, comments),p) =
                [p_InstanceSendEventArea (InstanceSendEventArea,p)]
 |  p_InstanceEventArea (INSTANCERECEIVEEVENTAREA (InstanceReceiveEventArea, comments),p) =
                [p_InstanceReceiveEventArea (InstanceReceiveEventArea,p)]
 |  p_InstanceEventArea (INSTANCECALLEVENTAREA (InstanceCallEventArea, comments),p) =
                [p_InstanceCallEventArea (InstanceCallEventArea,p)]
 |  p_InstanceEventArea (INSTANCEGETCALLEVENTAREA (InstanceGetCallEventArea, comments),p) =
                [p_InstanceGetCallEventArea (InstanceGetCallEventArea,p)]
 |  p_InstanceEventArea (INSTANCEREPLYEVENTAREA (InstanceReplyEventArea, comments),p) =
                [p_InstanceReplyEventArea (InstanceReplyEventArea,p)]
 |  p_InstanceEventArea (INSTANCEGETREPLYOUTSIDECALLEVENTAREA (InstanceGetreplyOutsideCallEventArea,
comments),p) =
                [p_InstanceGetreplyOutsideCallEventArea (InstanceGetreplyOutsideCallEventArea,p)]
 |  p_InstanceEventArea (INSTANCERAISEEVENTAREA (InstanceRaiseEventArea, comments),p) =
                [p_InstanceRaiseEventArea (InstanceRaiseEventArea,p)]
 |  p_InstanceEventArea (INSTANCECATCHTIMEOUTWITHINCALLEVENTAREA
(InstanceCatchTimeoutWithinCallEventArea, comments),p) =
                [p_InstanceCatchTimeoutWithinCallEventArea
(InstanceCatchTimeoutWithinCallEventArea,p)]
 |  p_InstanceEventArea (INSTANCECATCHOUTSIDECALLEVENTAREA (InstanceCatchOutsideCallEventArea,
comments),p) =
                [p_InstanceCatchOutsideCallEventArea (InstanceCatchOutsideCallEventArea,p)]
 |  p_InstanceEventArea (INSTANCETRIGGEREVENTAREA (InstanceTriggerEventArea, comments),p) =
                [p_InstanceTriggerEventArea (InstanceTriggerEventArea,p)]
 |  p_InstanceEventArea (INSTANCECHECKEVENTAREA (InstanceCheckEventArea, comments),p) =
                [p_InstanceCheckEventArea (InstanceCheckEventArea,p)]
 |  p_InstanceEventArea (INSTANCEFOUNDEVENTAREA (InstanceFoundEventArea, comments),p) =
                [p_InstanceFoundEventArea InstanceFoundEventArea]
 |  p_InstanceEventArea (INSTANCETIMEREVENTAREA (InstanceTimerEventArea, comments),p) =
                [p_InstanceTimerEventArea (InstanceTimerEventArea,p)]
 |  p_InstanceEventArea (INSTANCEACTIONEVENTAREA (InstanceActionArea, comments),p) =
                 p_InstanceActionArea InstanceActionArea
 |  p_InstanceEventArea (INSTANCELABELLINGEVENTAREA (InstanceLabellingArea, comments),p) =
                [p_InstanceLabellingArea InstanceLabellingArea]
 |  p_InstanceEventArea (INSTANCECONDITIONEVENTAREA (InstanceConditionArea, comments),p) =
                [p_InstanceConditionArea (InstanceConditionArea,p)]
 |  p_InstanceEventArea (INSTANCEINVOCATIONEVENTAREA (InstanceInvocationArea, comments),p) =
                [p_InstanceInvocationArea (InstanceInvocationArea,p)]
 |  p_InstanceEventArea (INSTANCEDEFAULTHANDLINGAREA (InstanceDefaultHandlingArea, comments),p) =
                [p_InstanceDefaultHandlingArea InstanceDefaultHandlingArea]
 |  p_InstanceEventArea (INSTANCECOMPONENTCREATEAREA (InstanceComponentCreateArea, comments),p) =
                [p_InstanceComponentCreateArea InstanceComponentCreateArea]
 |  p_InstanceEventArea (INSTANCECOMPONENTSTARTAREA (InstanceComponentStartArea, comments),p) =
                [p_InstanceComponentStartArea InstanceComponentStartArea]
 |  p_InstanceEventArea (INSTANCEINLINEEXPRESSIONEVENTAREA (InstanceInlineExpressionEventArea,
comments),p) =
      let

        (*************************************************************************
    p_InstanceEventAreaList : InstanceEventArea list *
            (InstanceEventLayer, InstanceLayer,ConnectorLayer,PortEventLayer)
                                    -> cn_type.FunctionStatementOrDef list
    *************************************************************************)
    fun p_InstanceEventAreaList []  p                       = []
     |  p_InstanceEventAreaList (InstanceEventArea::t) p    =
p_InstanceEventArea (InstanceEventArea,p)@p_InstanceEventAreaList t p
    (* end of p_InstanceEventAreaList *)

        (*************************************************************************
    p_IfArea : IfArea * InstanceEventArea list * InstanceEventArea list option *
            (InstanceEventLayer, InstanceLayer,ConnectorLayer,PortEventLayer)
                                    -> cn_type.FunctionStatementOrDef
    *************************************************************************)
    fun p_IfArea (((BooleanExpression,OperandaArea1,NONE),_,SOME PortInlineExpressionBeginSymbol),
            IfInstanceEventAreaList,                                    (* If Event list *)
            NONE,                                                       (* Else Event list *)
           (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer))
        =
            cn_type.FUNCTIONSTATEMENT (
                cn_type.BASICSTATEMENTS (
                    cn_type.CONDITIONALCONSTRUCT
                        (BooleanExpression,
                         p_InstanceEventAreaList IfInstanceEventAreaList
                         (InstanceEventLayer,InstanceLayer,OperandaArea1,PortEventLayer),
                         [],     (* No if else clauses - represented by nested inline expressions *)
```

```
                    NONE))) (* No else clause in this case *)
|  p_IfArea (((BooleanExpression,OperandaArea1,SOME OperandaArea2),_,SOME
            PortInlineExpressionBeginSymbol),                    (* IfArea *)
            IfInstanceEventAreaList,                             (* If Event list *)
            SOME ElseInstanceEventAreaList,                      (* Else Event list *)
           (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer))
        cn_type.FUNCTIONSTATEMENT (
            cn_type.BASICSTATEMENTS (
                cn_type.CONDITIONALCONSTRUCT
                    (BooleanExpression,
                     p_InstanceEventAreaList IfInstanceEventAreaList
                    (InstanceEventLayer,InstanceLayer,OperandaArea1,PortEventLayer),
                     [],     (* No if else clauses - represented by nested inline expressions *)
                     SOME (p_InstanceEventAreaList ElseInstanceEventAreaList
                    (InstanceEventLayer,InstanceLayer,OperandaArea2,PortEventLayer))
                     )
                )
            )
(* end of p_IfArea *)

(*************************************************************************
p_InstanceIfArea : InstanceIfArea *
            (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                    -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceIfArea
            ((InstanceInlineExpressionBeginSymbol,IfInstanceEventArea,ElseInstanceEventArea),
              p as (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)) =
    let
        val IfArea = given_functions.get_attached_IfArea
                    (InstanceInlineExpressionBeginSymbol,ConnectorLayer)
    in
        p_IfArea (IfArea,IfInstanceEventArea,ElseInstanceEventArea,p)
    end
(* end of p_InstanceIfArea *)

(*************************************************************************
p_ForArea : ForArea * InstanceEventArea list  *
        (InstanceEventLayer, InstanceLayer,ConnectorLayer,PortEventLayer)
                                    -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_ForArea (((Initial, Final, Step, OperandaArea),_,_),  (* ForArea *)
         InstanceEventAreaList,                              (* Event list *)
        (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer))
    =
        cn_type.FUNCTIONSTATEMENT (
            cn_type.BASICSTATEMENTS (
                cn_type.LOOPCONSTRUCT (
                    cn_type.FORSTATEMENT (Initial, Final, Step,
                    p_InstanceEventAreaList InstanceEventAreaList
                    (InstanceEventLayer,InstanceLayer,OperandaArea,PortEventLayer)
                    )
                )
            )
        )
(* end of p_ForArea *)

(*************************************************************************
p_InstanceForArea : InstanceForArea *
            (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                    -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceForArea ((InstanceInlineExpressionBeginSymbol,InstanceEventArealist),
              p as (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)) =
    let
        val ForArea = given_functions.get_attached_ForArea
                     (InstanceInlineExpressionBeginSymbol,ConnectorLayer)
    in
        p_ForArea (ForArea,InstanceEventArealist,p)
    end
(* end of p_InstanceForArea *)

(*************************************************************************
p_WhileArea : WhileArea * InstanceEventArea list  *
        (InstanceEventLayer, InstanceLayer,ConnectorLayer,PortEventLayer)
                                    -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_WhileArea (((BooleanExpression, OperandaArea),_,_),   (* WhileArea *)
         InstanceEventAreaList,                              (* Event list *)
        (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer))
```

```
    =
        cn_type.FUNCTIONSTATEMENT (
            cn_type.BASICSTATEMENTS (
                cn_type.LOOPCONSTRUCT (
                    cn_type.WHILESTATEMENT (BooleanExpression,
                    p_InstanceEventAreaList InstanceEventAreaList
                    (InstanceEventLayer,InstanceLayer,OperandaArea,PortEventLayer)
                    )
                )
            )
        )
(* end of p_WhileArea *)

(*************************************************************************
p_InstanceWhileArea : InstanceWhileArea *
            (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                    -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceWhileArea ((InstanceInlineExpressionBeginSymbol,InstanceEventArealist),
              p as (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)) =
    let
        val WhileArea = given_functions.get_attached_WhileArea
                        (InstanceInlineExpressionBeginSymbol,ConnectorLayer)
    in
        p_WhileArea (WhileArea,InstanceEventArealist,p)
    end
(* end of p_InstanceWhileArea *)


    (*************************************************************************
p_DoWhileArea : DoWhileArea * InstanceEventArea list  *
        (InstanceEventLayer, InstanceLayer,ConnectorLayer,PortEventLayer)
                                    -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_DoWhileArea (((BooleanExpression, OperandaArea),_,_),
         InstanceEventAreaList,                              (* Event list *)
        (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer))
    =
        cn_type.FUNCTIONSTATEMENT (
            cn_type.BASICSTATEMENTS (
                cn_type.LOOPCONSTRUCT (
                    cn_type.WHILESTATEMENT (BooleanExpression,
                    p_InstanceEventAreaList InstanceEventAreaList
                    (InstanceEventLayer,InstanceLayer,OperandaArea,PortEventLayer)
                    )
                )
            )
        )
(* end of p_DoWhileArea *)

(*************************************************************************
p_InstanceDoWhileArea : InstanceDoWhileArea *
            (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                    -> cn_type.FunctionStatementOrDef
*************************************************************************)
fun p_InstanceDoWhileArea ((InstanceInlineExpressionBeginSymbol,InstanceEventArealist),
              p as (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)) =
    let
        val DoWhileArea = given_functions.get_attached_DoWhileArea
                        (InstanceInlineExpressionBeginSymbol,ConnectorLayer)
    in
        p_DoWhileArea (DoWhileArea,InstanceEventArealist,p)
    end
(* end of p_InstanceDoWhileArea *)

(*************************************************************************
p_BooleanExpressionConditionArea : BooleanExpressionConditionArea
                    -> cn_type.AltGuardChar
*************************************************************************)
fun p_BooleanExpressionConditionArea BooleanExpression = BooleanExpression
(* end of p_BooleanExpressionConditionArea *)

(*************************************************************************
p_InstanceBooleanExpressionConditionArea :InstanceBooleanExpressionConditionArea
                    -> cn_type.AltGuardChar
*************************************************************************)
fun p_InstanceBooleanExpressionConditionArea x = p_BooleanExpressionConditionArea x
(* end of p_InstanceBooleanExpressionConditionArea *)
```

```
    (*************************************************************************
    p_InstanceGuardOpArea : InstanceGuardOpArea -> cn_type.GuardOp
    *************************************************************************)
    fun p_InstanceGuardOpArea ((INSTANCEGUARDRECEIVEEVENTAREA x),p) =
        let
           val (cn_type.FUNCTIONSTATEMENT(
                cn_type.COMMUNICATIONSTATEMENTS(
                    cn_type.RECEIVESTATEMENT ReceiveStatement
                    )
                )
            ) = p_InstanceReceiveEventArea (x,p)
        in
            cn_type.GUARDRECEIVESTATEMENT ReceiveStatement
        end
    (* end of p_InstanceGuardOpArea *)

    (*************************************************************************
    p_GuardArea : GuardArea  *
            (InstanceEventLayer * InstanceLayer* (GuardOpLayer * ConnectorLayer) * PortEventLayer)
                -> cn_type.GuardStatement
    *************************************************************************)
    fun p_GuardArea (INSTANCEGUARDOPAREA (InstanceGuardOpArea,
                          InstanceEventAreaList),
                 (InstanceEventLayer,InstanceLayer,(GuardOpLayer,ConnectorLayer),PortEventLayer))
    =
       cn_type.GUARDOPSTATEMENT (
               p_InstanceGuardOpArea (InstanceGuardOpArea,(InstanceEventLayer,InstanceLayer,
                           given_functions.GuardOpLayer_to_ConnectorLayer
                            GuardOpLayer,PortEventLayer)
                          ),
               p_InstanceEventAreaList InstanceEventAreaList
                        (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
           )
    (* end of p_GuardArea *)

    (*************************************************************************
    p_InstanceGuardArea : InstanceGuardArea *
        (InstanceEventLayer * InstanceLayer* (GuardOpLayer * ConnectorLayer) * PortEventLayer)
                -> cn_type.GuardStatement
    *************************************************************************)
    fun p_InstanceGuardArea  ((SOME InstanceBooleanExpressionConditionArea,GuardArea),p) =
          (SOME (p_InstanceBooleanExpressionConditionArea InstanceBooleanExpressionConditionArea),
                   p_GuardArea (GuardArea,p))
          | p_InstanceGuardArea  ((NONE,GuardArea),p) =
          (NONE,
           p_GuardArea (GuardArea,p))
    (* end of p_InstanceGuardArea *)

    (*************************************************************************
    p_InstanceElseGuardArea : InstanceElseGuardArea *
        (InstanceEventLayer * InstanceLayer* (GuardOpLayer * ConnectorLayer) * PortEventLayer)
                -> cn_type.GuardStatement
    *************************************************************************)
    fun p_InstanceElseGuardArea (SOME (ElseConditionArea,InstanceEventAreaList),
                    (InstanceEventLayer,InstanceLayer,SOME (_,ConnectorLayer),PortEventLayer))
                =
                     SOME (p_InstanceEventAreaList InstanceEventAreaList
                    (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer))
     |  p_InstanceElseGuardArea (NONE,_)
                =    NONE
    (* end of p_InstanceElseGuardArea *)

    (*************************************************************************
    p_InstanceGuardAreaList : InstanceGuardArea list ->
    (InstanceEventLayer * InstanceLayer * (GuardOpLayer * ConnectorLayer) list * PortEventLayer)
                        -> cn_type.GuardStatement list
    *************************************************************************)
    fun p_InstanceGuardAreaList ([],_)= []
     |  p_InstanceGuardAreaList
(InstanceGuardArea::t,(InstanceEventLayer,InstanceLayer,h::t',PortEventLayer)) =
           p_InstanceGuardArea
(InstanceGuardArea,(InstanceEventLayer,InstanceLayer,h,PortEventLayer))::
           p_InstanceGuardAreaList (t,(InstanceEventLayer,InstanceLayer,t',PortEventLayer))
     (* end of p_InstanceGuardAreaList *)

    (*************************************************************************
    p_AltArea : AltArea * InstanceGuardArea * InstanceGuardArea list * InstanceElseGuardArea option*
           (InstanceEventLayer, InstanceLayer,ConnectorLayer,PortEventLayer)
                                      -> cn_type.FunctionStatementOrDef
    *************************************************************************)
```

```
    fun p_AltArea (((GuardedOperandArea,GuardedOperandAreaList,ElseOperandArea),_,_),
              InstanceGuardArea,InstanceGuardAreaList,InstanceElseGuardArea,  (* Event lists *)
            (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer))
        =
            cn_type.FUNCTIONSTATEMENT (
                cn_type.BEHAVIOURSTATEMENTS (
                    cn_type.ALTCONSTRUCT (
                        p_InstanceGuardArea
(InstanceGuardArea,(InstanceEventLayer,InstanceLayer,GuardedOperandArea,PortEventLayer))
                            ::p_InstanceGuardAreaList
(InstanceGuardAreaList,(InstanceEventLayer,InstanceLayer,GuardedOperandAreaList,PortEventLayer)),
                        p_InstanceElseGuardArea
(InstanceElseGuardArea,(InstanceEventLayer,InstanceLayer,ElseOperandArea,PortEventLayer))
                        )
                    )
            )
    (* end of p_AltArea *)

    (*************************************************************************
    p_InstanceAltArea : InstanceAltArea *
                (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                        -> cn_type.FunctionStatementOrDef
    *************************************************************************)
    fun p_InstanceAltArea
((InstanceInlineExpressionBeginSymbol,InstanceGuardArea,InstanceGuardArealist,InstanceElseGuardArea)
,
                  p as (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)) =
        let
            val AltArea = given_functions.get_attached_AltArea
                          (InstanceInlineExpressionBeginSymbol,ConnectorLayer)
        in
            p_AltArea (AltArea,InstanceGuardArea,InstanceGuardArealist,InstanceElseGuardArea,p)
        end
    (* end of p_InstanceAltArea *)

    (*************************************************************************
    p_InstanceGuardOpArea : InstanceGuardOpArea *
            (InstanceEventLayer * InstanceLayer* (GuardOpLayer * ConnectorLayer) * PortEventLayer)
                -> cn_type.InterleavedGuard
    *************************************************************************)
    fun p_InstanceGuardOpArea ((INSTANCEGUARDRECEIVEEVENTAREA x),p) =
        let
           val (cn_type.FUNCTIONSTATEMENT(
                cn_type.COMMUNICATIONSTATEMENTS(
                    cn_type.RECEIVESTATEMENT ReceiveStatement
                    )
                )
            ) = p_InstanceReceiveEventArea (x,p)
        in
            cn_type.GUARDRECEIVESTATEMENT ReceiveStatement
        end
    (* p_InstanceGuardOpArea *)

    (*************************************************************************
    p_InstanceInterleaveGuardArea : InstanceInterleaveGuardArea *
                (InstanceEventLayer * InstanceLayer* (GuardOpLayer * ConnectorLayer) *
PortEventLayer)
                -> cn_type.InterleavedGuardElement
    *************************************************************************)
    fun p_InstanceInterleaveGuardArea  ((InstanceGuardOpArea,InstanceEventAreaList),
        (InstanceEventLayer,InstanceLayer,(UnguardOpLayer,ConnectorLayer),PortEventLayer)) =
            (p_InstanceGuardOpArea (InstanceGuardOpArea,(InstanceEventLayer,InstanceLayer,
                            given_functions.UnguardOpLayer_to_ConnectorLayer
                                        UnguardOpLayer,PortEventLayer)),
             p_InstanceEventAreaList InstanceEventAreaList
                            (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer))
    (* end of p_InstanceInterleaveGuardArea *)

     (*************************************************************************
    p_InstanceInterleaveGuardAreaList : InstanceInterleaveGuardArea list ->
        (InstanceEventLayer * InstanceLayer * (GuardOpLayer * ConnectorLayer) list * PortEventLayer)
                        -> cn_type.InterleavedGuardElement list
    *************************************************************************)
    fun p_InstanceInterleaveGuardAreaList ([],_)= []
     |  p_InstanceInterleaveGuardAreaList
(InstanceInterleaveGuardArea::t,(InstanceEventLayer,InstanceLayer,h::t',PortEventLayer)) =
            p_InstanceInterleaveGuardArea
(InstanceInterleaveGuardArea,(InstanceEventLayer,InstanceLayer,h,PortEventLayer))::
            p_InstanceInterleaveGuardAreaList
(t,(InstanceEventLayer,InstanceLayer,t',PortEventLayer))
```

```
     (* end of p_InstanceInterleaveGuardAreaList *)

    (*************************************************************************
    p_InterleaveArea : InterleaveArea * InstanceGuardArea * InstanceGuardArea list *
InstanceElseGuardArea option*
            (InstanceEventLayer, InstanceLayer,ConnectorLayer,PortEventLayer)
                                    -> cn_type.FunctionStatementOrDef
    *************************************************************************)
    fun p_InterleaveArea (((UnguardedOperandArea,UnguardedOperandAreaList),_,_):InterleaveArea,
        (* InterleaveArea *)
             InstanceInterleaveGuardArea,InstanceInterleaveGuardAreaList,
            (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer))
        =
            cn_type.FUNCTIONSTATEMENT (
                cn_type.BEHAVIOURSTATEMENTS (
                    cn_type.INTERLEAVEDCONSTRUCT (
                        p_InstanceInterleaveGuardArea
(InstanceInterleaveGuardArea,(InstanceEventLayer,InstanceLayer,UnguardedOperandArea,PortEventLayer))
                            ::p_InstanceInterleaveGuardAreaList
(InstanceInterleaveGuardAreaList,(InstanceEventLayer,InstanceLayer,UnguardedOperandAreaList,PortEven
tLayer))
                    )
                )
            )
    (* end of p_InterleaveArea *)

    (*************************************************************************
    p_InstanceInterleaveArea : InstanceInterleaveArea *
                (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                    -> cn_type.FunctionStatementOrDef
    *************************************************************************)
    fun p_InstanceInterleaveArea
((InstanceInlineExpressionBeginSymbol,InstanceInterleaveGuardArea,InstanceInterleaveGuardAreaList)
    :InstanceInterleaveArea,
                 p as (InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)) =
        let
            val InterleaveArea = given_functions.get_attached_InterleaveArea
                            (InstanceInlineExpressionBeginSymbol,ConnectorLayer)
        in
            p_InterleaveArea
                    (InterleaveArea,InstanceInterleaveGuardArea,InstanceInterleaveGuardAreaList,p)
        end
    (* end of p_InstanceInterleaveArea *)

    (*************************************************************************
        p_InstanceInlineExpressionEventArea :
                InstanceInlineExpressionEventArea *

(InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
                                            -> cn_type.FunctionStatementOrDef
    *************************************************************************)
    fun p_InstanceInlineExpressionEventArea ((INSTANCEIFAREA InstanceIfArea),p)         =
p_InstanceIfArea (InstanceIfArea,p)
     |  p_InstanceInlineExpressionEventArea ((INSTANCEFORAREA InstanceForArea),p)       =
p_InstanceForArea (InstanceForArea,p)
     |  p_InstanceInlineExpressionEventArea ((INSTANCEWHILEAREA InstanceWhileArea),p)   =
p_InstanceWhileArea (InstanceWhileArea,p)
     |  p_InstanceInlineExpressionEventArea ((INSTANCEDOWHILEAREA InstanceDoWhileArea),p)   =
p_InstanceDoWhileArea (InstanceDoWhileArea,p)
     |  p_InstanceInlineExpressionEventArea ((INSTANCEALTAREA InstanceAltArea),p)       =
p_InstanceAltArea (InstanceAltArea,p)
     |  p_InstanceInlineExpressionEventArea ((INSTANCEINTERLEAVEAREA InstanceInterleaveArea),p) =
p_InstanceInterleaveArea (InstanceInterleaveArea,p)

       in
              [p_InstanceInlineExpressionEventArea (InstanceInlineExpressionEventArea,p)]
       end
 (* end of p_InstanceEventArea *)

(*************************************************************************
p_InstanceEventLayer : InstanceEventArea list *
                                -> cn_type.StatementBlock
*************************************************************************)
fun p_InstanceEventLayer [] p                       = []
 |  p_InstanceEventLayer (InstanceEventArea::t) p   = p_InstanceEventArea (InstanceEventArea,p)@
                                p_InstanceEventLayer t p
(* end of p_InstanceEventLayer *)

(*************************************************************************
    p_TestcaseBodyArea : TestcaseBodyArea -> cn_type.StatementBlock
```

```
        The InstanceEventlayer defines the order in which the events are placed
        upon an instance axis. Therefore, this function recursively iterates
        over the InstanceEventlayer list.

*************************************************************************)
fun p_TestcaseBodyArea (InstanceLayer,TextLayer,InstanceEventLayer,PortEventLayer, ConnectorLayer) =
    p_InstanceEventLayer InstanceEventLayer
(InstanceEventLayer,InstanceLayer,ConnectorLayer,PortEventLayer)
(* end of p_TestcaseBodyArea *)

(************************************************************************
    p_TestcaseHeading : (TestcaseHeading, TestcaseBodyArea) -> cn_type.TestcaseDef
************************************************************************)
fun p_TestcaseHeading ((TestcaseIdentifier,TestcaseFormalParList,ConfigSpec,LocalDefinitions),
                       TestcaseBodyArea) =
        let
                (* Place GFT testcase diagram in display attribute and          *)
                (* append those attributes that are defined on the diagram.    *)
                (* Note that Comments are currently ignored                     *)
                val TextLayer                                =
                        given_functions.extract_TextLayer_from_testcase TestcaseBodyArea
                val (testcase_comments,testcase_attributes) = p_TextLayer TextLayer
                val withstatement   = SOME (
                                        [
                                         [cn_type.DISPLAY "ETSI TTCN-3 GFT v1.0",
                                          cn_type.DISPLAY ("Testcase Diagram -
                                                           "^TestcaseIdentifier)
                                         ]
                                        ]@testcase_attributes
                                      )
        in
               (
               TestcaseIdentifier,
               TestcaseFormalParList,
               ConfigSpec,
               p_TestcaseBodyArea TestcaseBodyArea, (* Don't forget local definitions *)
               withstatement
               )
        end
(* end of p_TestcaseHeading *)

(************************************************************************
    gft_testcase_to_cn : testcaseDiagram -> cn_type.TestcaseDef
************************************************************************)
fun gft_testcase_to_cn (TestcaseHeading,TestcaseBodyArea) =
                p_TestcaseHeading (TestcaseHeading,TestcaseBodyArea)
(* end of gft_testcase_to_cn *)

end (* end of gfttocn structure *)
```

For further details, please refer to the file gfttocn.sml. It is not given here, but can be provided on request from ETSI MTS.

Annex D (informative)

Mapping TTCN-3 core language to GFT

This annex defines an executable mapping from TTCN-3 core language [1] to GFT. The purpose behind this activity has been to aid the validation of the graphical grammar and language concepts. It can also be used to verify that the GFT to core and core to GFT mappings are bidirectional.

In order to provide both an abstract and readable implementation we have chosen to use the functional programming language, Standard ML of New Jersey (SML/NJ). SML/NJ is open source and is freely available from Bell Laboratories [http://cm.bell-labs.com/cm/cs/what/smlnj/].

D.1 Approach

The approach for the executable model has been to firstly represent both the core language and GFT grammars as separate SML data types (structures) `cn_type` and `gft_type` respectively. We then define a set of mapping functions that map the GFT data type onto the core data type for each GFT object (i.e., test case diagram, test step diagram, control diagram, and function diagram). The SML signatures for these mapping functions are as follows:

```
cn_testcase_to_gft : cn_type.TestcaseDef -> gft_type.TestcaseDiagram
cn_teststep_to_gft : cn_type.TeststepDef -> gft_type.TeststepDiagram
cn_function_to_gft : cn_type.FunctionDef -> gft_type.FunctionDiagram
cn_control_to_gft  : cn_type.TTCN3Module -> gft_type.ControlDiagram
```

D.1.2 Overview of SML/NJ

Please refer to C.1.2 for an overview on SML/NJ.

D.2 Modelling GFT graphical grammar in SML

D.2.1 SML modules

Please refer to C.2.1 for an overview on the used SML modules.

D.2.2 Function naming and references

Please refer to C.2.2 for an overview on function naming and references.

D.2.3 Given functions

Please refer to C.2.1 for an overview on given functions.

D.2.4 Core and GFT SML types

The types are contained in gft_type.sml and CNType.sml. They are not given here, but can be provided on request from ETSI MTS.

D.2.5 Core to GFT mapping functions

Since the core to GFT mapping functions are symmetric to the GFT to core mapping functions, only their signatures are given here.

```
*************************************************************************
p_StartTimerStatement : StartTimerStatement
                        -> gft_type.InstanceTimerStartArea
*************************************************************************

*************************************************************************
p_StopTimerStatement : StopTimerStatement
                        -> gft_type.InstanceTimerStopArea
*************************************************************************
```

```
*****************************************************************************
p_TimeoutStatement : TimeoutStatement
                          -> gft_type.InstanceTimeoutArea
*****************************************************************************

*****************************************************************************
p_TimerStatements : TimerStatements
                          -> gft_type.InstanceTimerEventArea
*****************************************************************************

*****************************************************************************
p_BehaviourStatements : BehaviourStatements -> gft_type.TestcaseBodyArea
*****************************************************************************

*****************************************************************************
p_TestcaseInstance : TestcaseInstance -> gft_type.TestcaseExecution
*****************************************************************************

*****************************************************************************
p_FunctionInstance : TestcaseIntance -> gft_type.Invocation
*****************************************************************************

*****************************************************************************
p_TeststepInstance : TeststepInstance -> gft_type.Invocation
*****************************************************************************

*****************************************************************************
p_ReturnStatements : ReturnStatements -> gft_type.ReturnArea
*****************************************************************************

*****************************************************************************
p_GuardStatement : GuardStatements -> gft_type.GuardArea
*****************************************************************************

*****************************************************************************
p_AltConstruct :  AltConstruct -> gft_type.AltArea
*****************************************************************************

*****************************************************************************
p_InterleavedConstruct: InterleavedConstruct -> gft_type.InterleaveArea
*****************************************************************************

*****************************************************************************
p_LabelStatement : LabelStatements -> gft_type.LabellingEventArea
*****************************************************************************

*****************************************************************************
p_GotoStatement : GotoStatements -> gft_type.GotoArea
*****************************************************************************

*****************************************************************************
p_WhileStatement : WhileStatements -> gft_type.WhileArea
*****************************************************************************

*****************************************************************************
p_DoWhileStatement : DoWhileStatements -> gft_type.DoWhileArea
*****************************************************************************

*****************************************************************************
p_ForStatement : ForStatements -> gft_type.ForArea
*****************************************************************************

*****************************************************************************
p_RepeatStatement : RepeatStatements -> gft_type.RepeatSymbol
*****************************************************************************

*****************************************************************************
p_DeactivateStatement : DeactivateStatements -> gft_type.DefaultHandling
*****************************************************************************

*****************************************************************************
p_ConnectStatements : ConnectStatements -> gft_type.ActionStatement
*****************************************************************************

*****************************************************************************
p_MapStatements : MapStatements -> gft_type.ActionStatement
*****************************************************************************
```

```
*****************************************************************************
p_DisconnectStatements : DisConnectStatements -> gft_type.ActionStatement
*****************************************************************************

*****************************************************************************
p_UnmapStatements : UnmapStatements -> gft_type.ActionStatement
*****************************************************************************

*****************************************************************************
p_DoneStatements : DoneStatements -> gft_type.ActionStatement
*****************************************************************************

*****************************************************************************
p_StartTCStatements : StartTCStatements -> gft_type.ActionStatement
*****************************************************************************

*****************************************************************************
p_StopTCStatements : StopTCStatements -> gft_type.ActionStatement
*****************************************************************************

*****************************************************************************
p_ComponentType : ComponentType -> gft_type.ComponentInstanceArea
*****************************************************************************

*****************************************************************************
p_ConfigurationStatements : ConfigurationStatements
                                              -> gft_type.ActionStatement
*****************************************************************************

*****************************************************************************
p_SUTStatements : SUTStatements
              -> gft_type.ActionStatement * gft_type.ControlActionStatement
*****************************************************************************

*****************************************************************************
p_VerdictStatements : VerdictSatement -> gft_type.SetVerdictArea
*****************************************************************************

*****************************************************************************
p_SendStatement : SendStatement * TestcaseFormalPar -> gft_type.SendArea
*****************************************************************************

*****************************************************************************
p_CallStatement : CallStatement * TestcaseFormalPar -> gft_type.CallArea
*****************************************************************************

*****************************************************************************
p_GetCallStatement : GetCallStatement * TestcaseFormalPar
                                                  -> gft_type.GetcallArea
*****************************************************************************

*****************************************************************************
p_ReplyStatement : ReplyStatement * TestcaseFormalPar -> gft_type.ReplyArea
*****************************************************************************

*****************************************************************************
p_RaiseStatement : RaiseStatement * TestcaseFormalPar -> gft_type.RaiseArea
*****************************************************************************

*****************************************************************************
p_ReceiveStatement : ReceiveStatement * TestcaseFormalPar
                                          -> gft_type.ReceiveArea
*****************************************************************************

*****************************************************************************
p_TriggerStatement : TriggerStatement * TestcaseFormalPar
                                          -> gft_type.TriggerArea
*****************************************************************************

*****************************************************************************
p_GetCallStatement : GetCallStatement * TestcaseFormalPar
                                              -> gft_type.GetCallArea
*****************************************************************************

*****************************************************************************
p_GetReplyStatement : TestcaseFormalPar * GetReplyStatement
        -> gft_type.GetReplyWithInCallArea * gft_type.GetReplyOutsideCallArea
*****************************************************************************
```

```
*****************************************************************************
p_CatchStatement : CatchStatement * TestcaseFormalPar
         -> gft_type.CatchWithInCallArea * gft_type.CatchOutsideCallArea
*****************************************************************************

*****************************************************************************
p_CheckStatement : CheckStatement * TestcaseFormalPar
                                          -> gft_type.CheckArea
*****************************************************************************

*****************************************************************************
p_ClearStatement : ClearStatement -> gft_type.ClearOpKeyWord
*****************************************************************************

*****************************************************************************
p_StartStatement : StartStatement -> gft_type.PortOperationText
*****************************************************************************

*****************************************************************************
p_StopStatement : StopStatement -> gft_type.PortOperationText
*****************************************************************************

*****************************************************************************
p_BasicStatement : BasicStatement
                          -> gft_type.ActionStatement
*****************************************************************************

*****************************************************************************
p_CommunicationStatements : CommunicationStatements
                  -> gft_type.ConnectorLayer * gft_type.InstanceLayer
*****************************************************************************

*****************************************************************************
p_ControlTimerStatements : ControlTimerStatements
                          -> gft_type.InstanceTimerEventArea
*****************************************************************************

*****************************************************************************
p_PortGetReplyOp : PortGetReplyOp * CommunicationStatement
                                                  -> gft_type.CheckData
*****************************************************************************

*****************************************************************************
p_CallBodyOps : CallBodyOps -> gft_type.CallBodyOpsLayer
*****************************************************************************

*****************************************************************************
p_ControlStatement : ControlStatement * FunctionLocalDef
                  -> gft_type.ControlEventArea * gft_type.ControlActionArea
*****************************************************************************

*****************************************************************************
p_ControlStatementOrDef : ControlStatementOrDef
                          -> gft_type.ControlActionStatement
*****************************************************************************

*****************************************************************************
p_ControlStatementOrDefList : ControlStatementOrDefList
                          -> gft_type.ControlActionArea
*****************************************************************************


*****************************************************************************
p_ModuleControlBody : ModuleControlBody
                          -> gft_type.ControlBodyArea
*****************************************************************************


*****************************************************************************
p_ModuleControlPart : ModuleControlPart
                          -> gft_type.ControlHeading
*****************************************************************************


*****************************************************************************
p_FunctionLocalInst: FunctionLocalInst ->  string
*****************************************************************************
```

```
*******************************************************************************
p_SingleWithAttrib : SingleWithAttrib -> String
*******************************************************************************

*******************************************************************************
p_WithStatement : WithStatement -> gft_type.TextLayer
*******************************************************************************

*******************************************************************************
p_TestcaseFormalPar: TestcaseFormalPar -> TestcaseFormalValuePar
*******************************************************************************

*******************************************************************************
p_FormalValuePar : FormalValuePar -> Type
*******************************************************************************

*******************************************************************************
p_FunctionStatement : FunctionStatement -> gft_type.TestcaseBodyArea
*******************************************************************************

*******************************************************************************
p_FunctionStatementOrDef : FunctionStatementOrDef
                                    ->  gft_type.LocalDefinition
*******************************************************************************

*******************************************************************************
p_StatementBlock : StatementBlock * WithStatement

         -> gft_type.TestcaseBodyArea
*******************************************************************************

*******************************************************************************
p_TeststepDef : TeststepDef -> gft_type.TeststepDiagram
*******************************************************************************

*******************************************************************************
p_FunctionDef : FunctionDef -> gft_type.FunctionDiagram
*******************************************************************************

*******************************************************************************
p_TTCN3Module : TTCN3Module -> gft_type.ControlDiagram
*******************************************************************************

*******************************************************************************
p_TestcaseDef : TestcaseDef -> gft_type.TestcaseDiagram
*******************************************************************************

*******************************************************************************
cn_teststep_to_gft : TeststepDef -> gft_type.TeststepDiagram
*******************************************************************************

*******************************************************************************
cn_function_to_gft : FunctionDef -> gft_type.FunctionDiagram
*******************************************************************************

*******************************************************************************
cn_control_to_gft : TTCN3Module -> gft_type.ControlDiagram
*******************************************************************************

*******************************************************************************
cn_testcase_to_gft : TestcaseDef -> gft_type.TestcaseDiagram
*******************************************************************************
```

For further details, please refer to the file cntogft.sml. It is not given here, but can be provided on request from ETSI MTS.

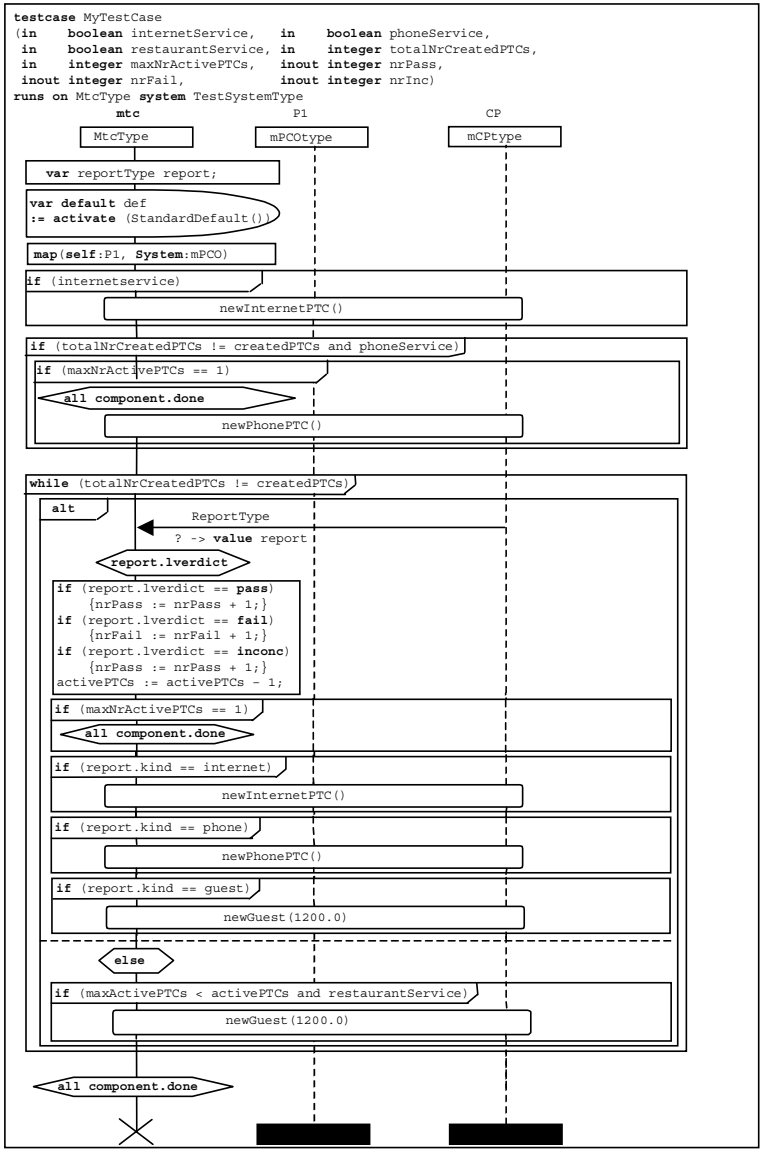
Annex E (informative)

Examples

E.1 The Restaurant example: Figures E.1 to E.9

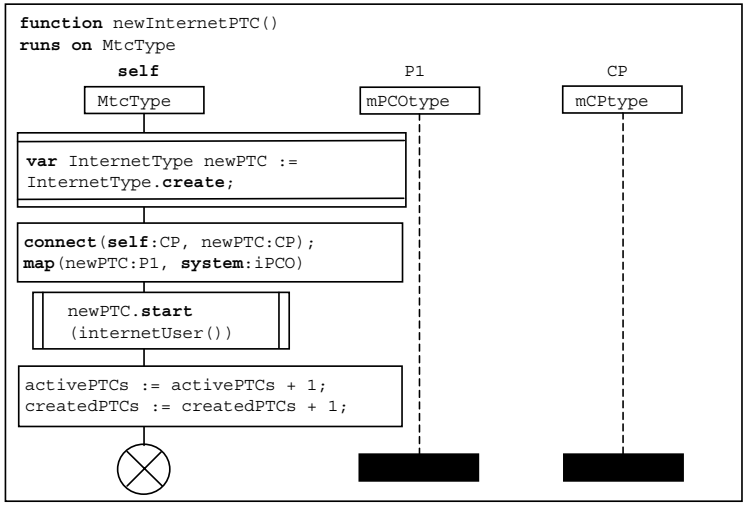
E.2 The INRES example: Figures E.10 to E.15

E.1 The restaurant example

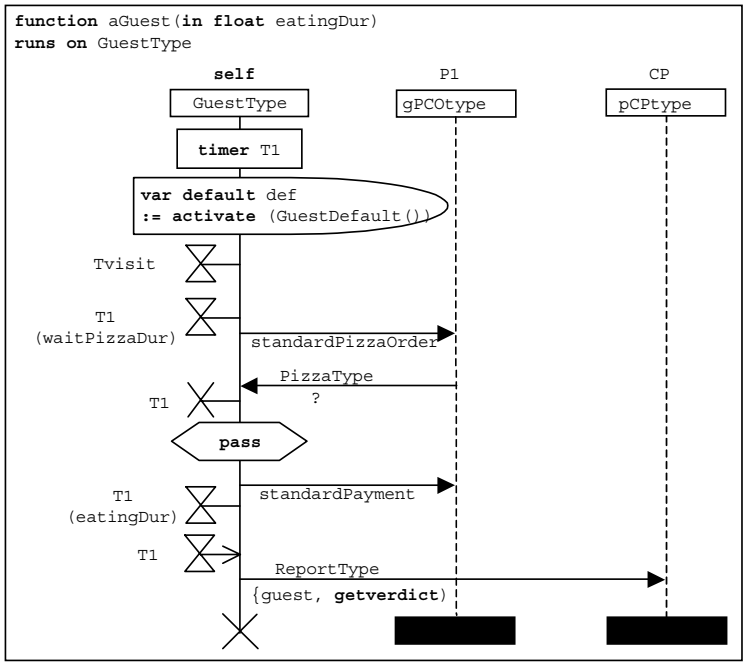



```
testcase MyTestCase (
    in boolean          internetService,     // SERVICES
    in boolean          phoneService,
    in boolean          restaurantService,
    in integer          totalNrCreatedPTCs,  // TERMINATION
    in integer          maxNrActivePTCs,     // CONTROL
    inout integer       nrPass,              // RETURN
    inout integer       nrFail,
    inout integer       nrInc
    )
  runs on MtcType
  system TestSystemType
{
    var ReportType report;

    var default def := activate (StandardDefault());

    map(self:P1, system:mPCO);

    if (internetService)  {
        newInternetPTC();
    }

    if (totalNrCreatedPTCs != createdPTCs
        and phoneService) {

        if (maxNrActivePTCs == 1) {
            all component.done;
        }
        newPhonePTC();
    }

    while ( totalNrCreatedPTCs != createdPTCs ) {

        alt {
            []  CP.receive(ReportType:?) -> value report {

                setverdict(report.lverdict);

                if (report.lverdict == pass) { nrPass := nrPass + 1; }
                if (report.lverdict == fail) { nrFail := nrFail + 1; }
                if (report.lverdict == inconc) { nrInc := nrInc + 1; }
                activePTCs := activePTCs - 1;

                if (maxNrActivePTCs == 1) {
                    all component.done;
                }
                if (report.kind == internet) {
                    newInternetPTC();
                }
                if (report.kind == phone) {
                    newPhonePTC();
                }
                if (report.kind == guest) {
                    newGuest(1200.0);
                }
            }


            [else] {

                if (maxNrActivePTCs < activePTCs
                    and restaurantService) {
                      newGuest(1200.0);
                    }
            }
        }
    }

    all component.done;
    stop;
  }
```

Figure E.1/Z.142 – Restaurant example – MyTestCase test case




```
function newInternetPTC ()
   runs on MtcType {

   var InternetType newPTC := InternetType.create;

   connect(self:CP, newPTC:CP);
   map(newPTC:P1, system:iPCO);

   newPTC.start(internetUser());

   activePTCs  := activePTCs + 1;
   createdPTCs := createdPTCs + 1;

   return;

}
```




```
function aGuest (in float eatingDur) runs on GuestType {

   timer T1;

   var default def := activate(GuestDefault());
   Tvisit.start; // component timer
   T1.start(waitPizzaDur);
   P1.send(standardPizzaOrder);
   P1.receive(PizzaType : ?);
   T1.stop;
   setverdict(pass);
   P1.send(standardPayment);
   T1.start(eatingDur);  // eating
   T1.timeout;
   CP.send(ReportType : {guest, getverdict});
   stop;
  } // end function aGuest
```

Figure E.2/Z.142 – Restaurant example – newInternetPTC and aGuest functions

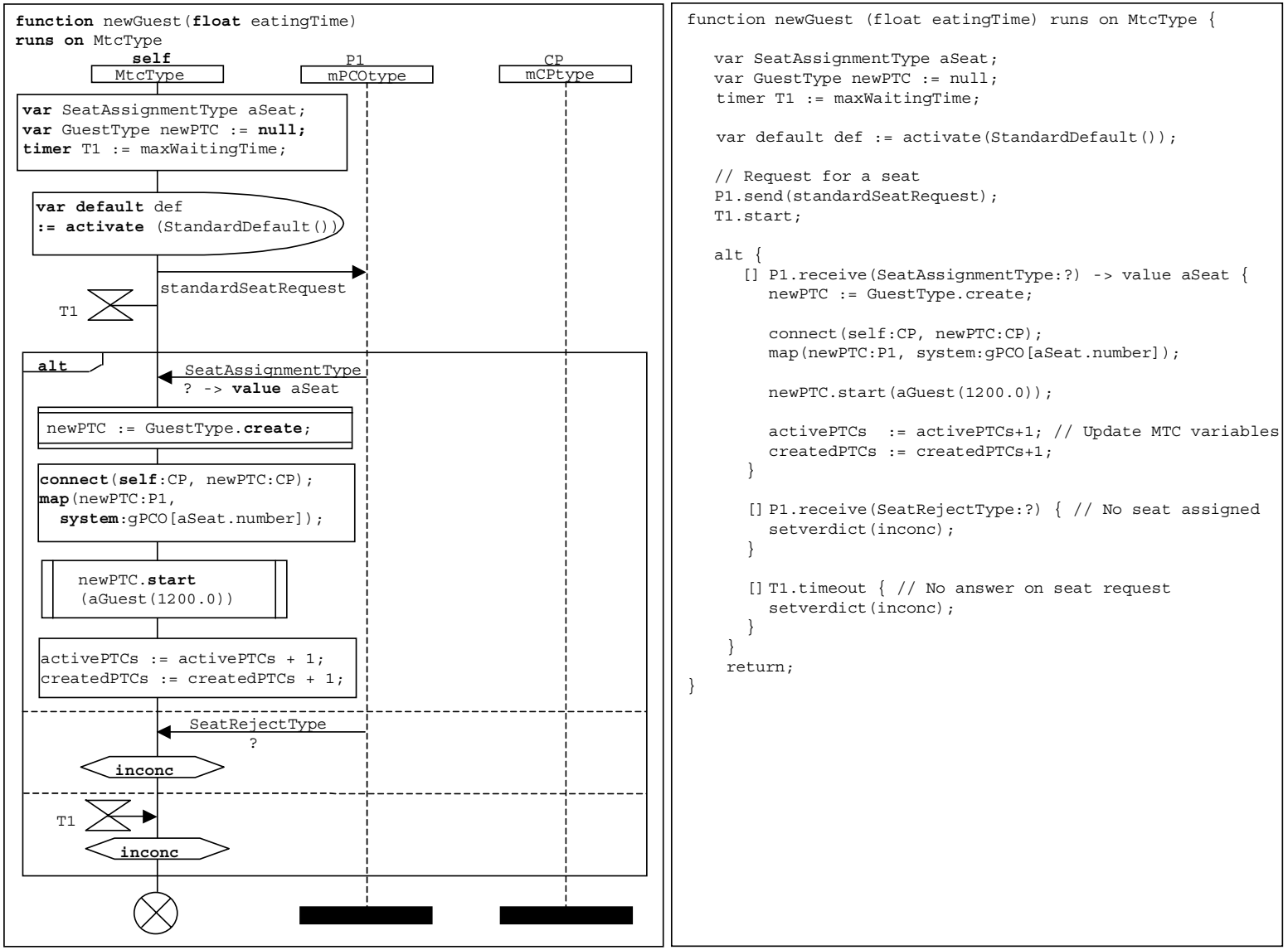
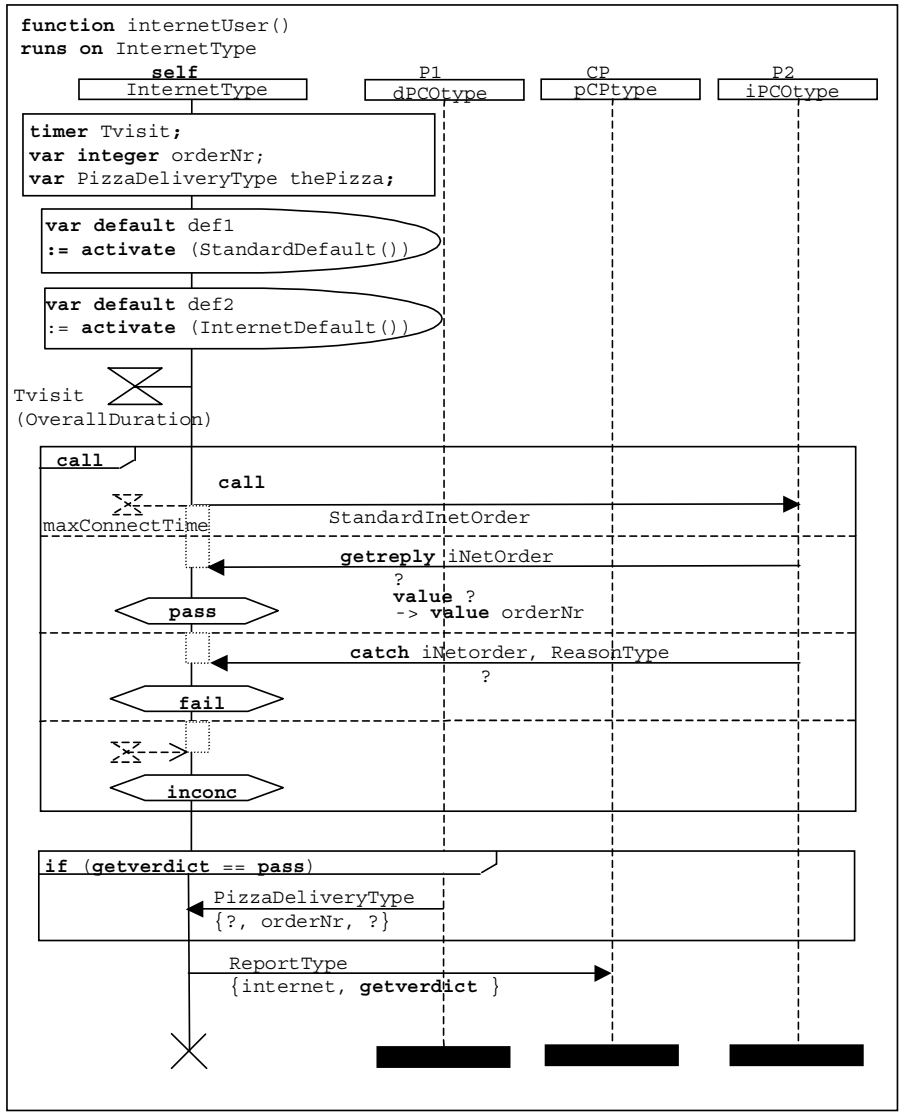
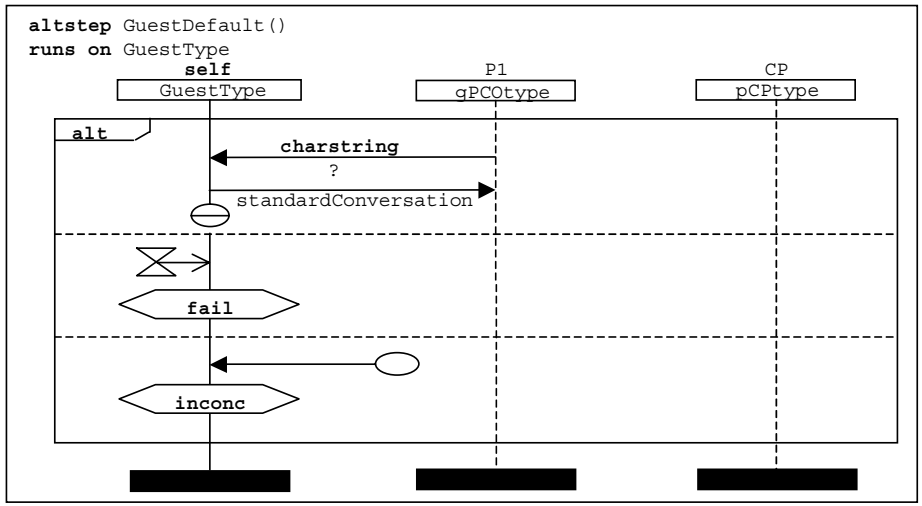


Figure E.3/Z.142 – Restaurant example – newGuest function

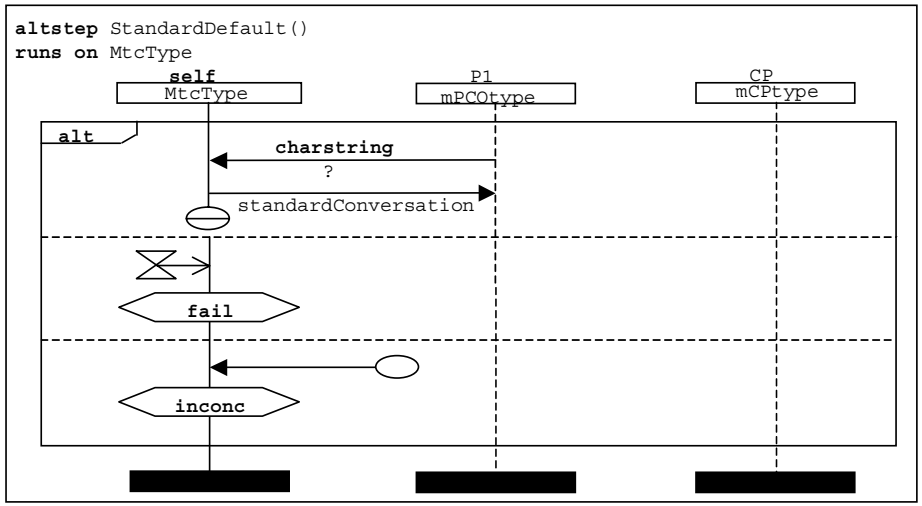



```
function internetUser () runs on InternetType {
  // ***
  // *** Purpose: Specifies the behaviour of an
  // *** internet guest
  // ***

  timer Tvisit;
  var integer orderNr;
  var PizzaDeliveryType thePizza;

  var default def1 := activate(StandardDefault());
  var default def2 := activate(InternetDefault());
  Tvisit.start(OverallDuration);

  P2.call(StandardINetOrder, maxConnectTime) {
     [] P2.getreply (iNetOrder:? value ?)
              -> value orderNr {
           setverdict(pass);
        }
     [] P2.catch (iNetOrder, ReasonType : ?) {

           setverdict(fail);
        }
     [] P2.catch (timeout) {
           setverdict(inconc);
        }
   };
   if (getverdict == pass) {
     P1.receive(PizzaDeliveryType
           : { ?, orderNr, ?});
   }
   CP.send(ReportType : {internet, getverdict});
   stop;
 }
```

Figure E.4/Z.142 – Restaurant example – internetUser function

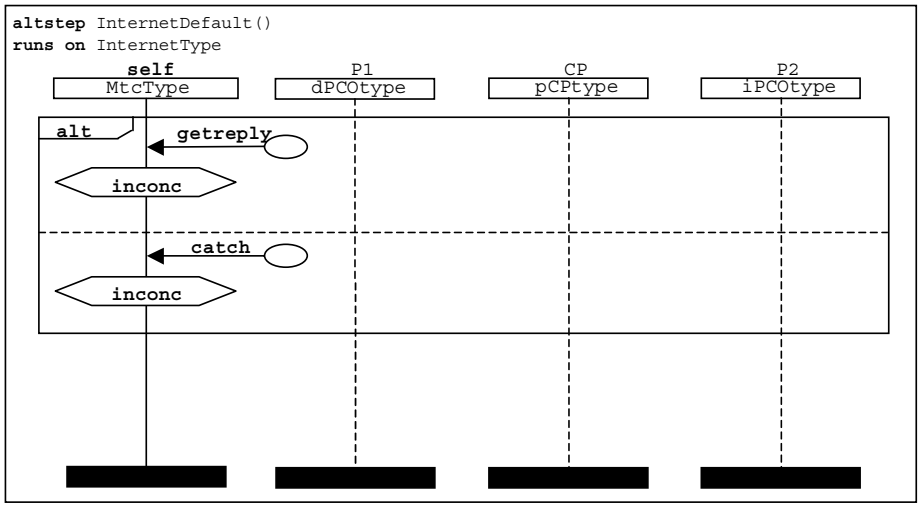



```
altstep GuestDefault() runs on GuestType {
   // ***
   // *** Purpose: Default behaviour for
   // *** message based ports
   // ***

   [] P1.receive(charstring : ?) {
      P1.send(standardConversation);
      repeat;
   }

   [] any timer.timeout {
      setverdict(fail);
   }

   [] any port.receive {
      setverdict(inconc);
   }
}
```

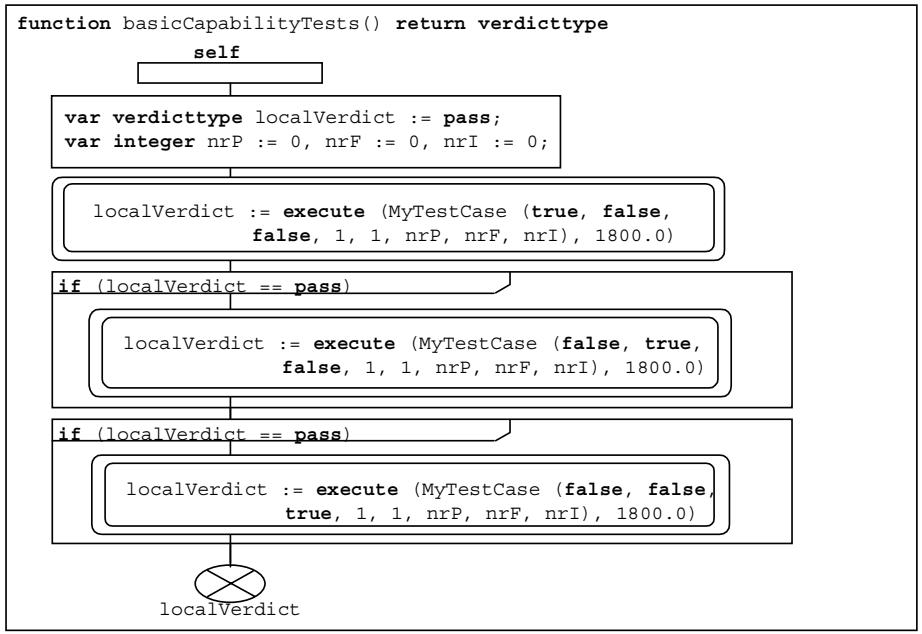



```
altstep StandardDefault() runs on MtcType {
   // ***
   // *** Purpose: Default behaviour for
   // **** message based ports
   // ***

   [] P1.receive(charstring : ?) {
      P1.send(standardConversation);
      repeat;
   }

   [] any timer.timeout {
      setverdict(fail);
   }

   [] any port.receive {
      setverdict(inconc);
   }
}
```

Figure E.5/Z.142 – Restaurant example – GuestDefaut and StandardDefault functions

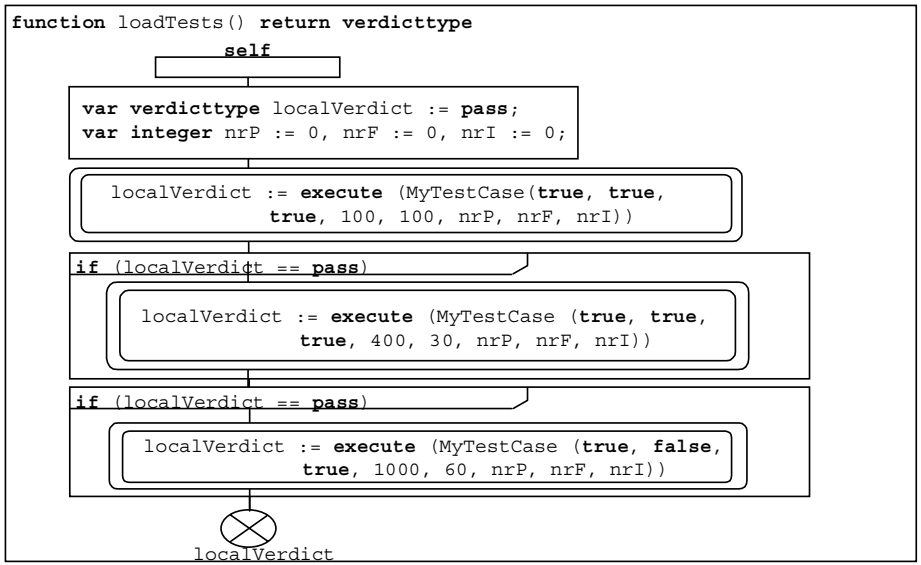



```
altstep InternetDefault()
   runs on InternetType {
   // ***
   // *** Purpose: Default behaviour for
   // **** the procedure based port
   // ***

   [] any port.getreply {
      setverdict(inconc);
   }

   [] any port.catch {
      setverdict(inconc);
   }
}
```

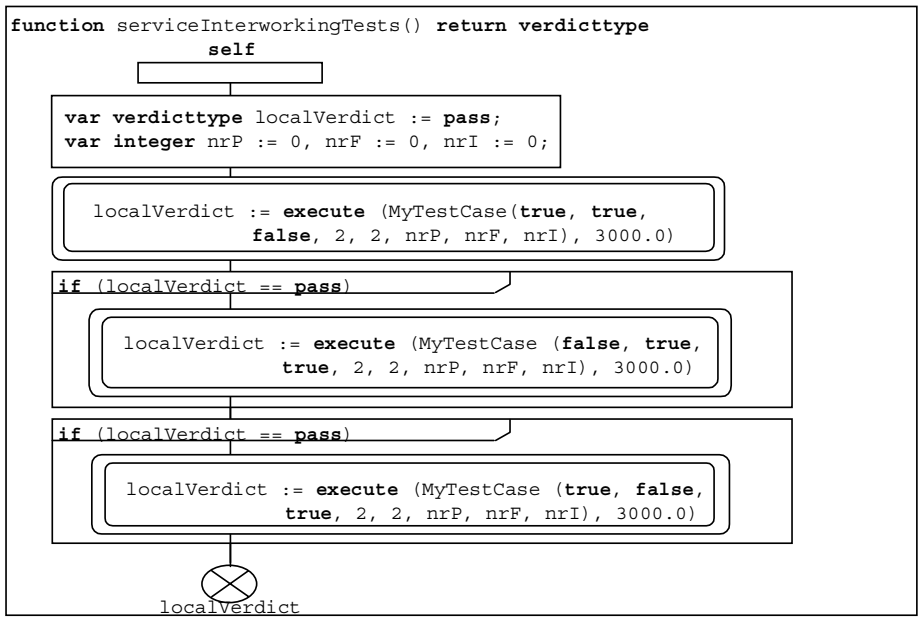



```
function basicCapabilityTests ()
   return verdicttype {
   var verdicttype localVerdict := pass;
   var integer nrP := 0, nrF := 0, nrI  := 0;

   // *** INTERNET ORDER ***
   localVerdict := execute(MyTestCase (true,false,
        false,1,1,nrP,nrF,nrI),1800.0);

   // *** PHONE ORDER
   if (localVerdict == pass) {
      localVerdict := execute(MyTestCase
      (false,true,false,1,1,nrP,nrF,nrI),1800.0);
   }

   // *** RESTAURANT ORDER ***
   if (localVerdict == pass) {
      localVerdict := execute(MyTestCase
      (false,false,true,1,1,nrP,nrF,nrI),1800.0);
   }
   return (localVerdict);
}
```

Figure E.6/Z.142 – Restaurant example – internetDefault altstep and basicCapabilityTests function

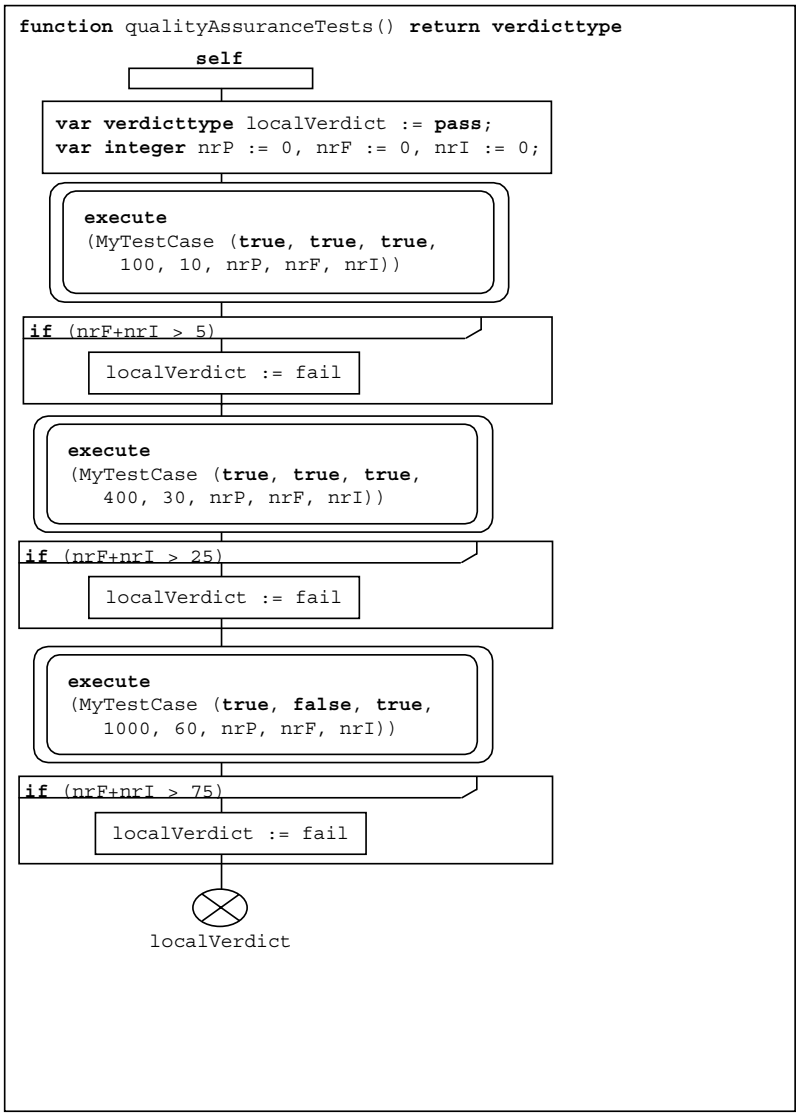



```
function loadTests () return verdicttype {
   var verdicttype localVerdict := pass;
   var integer nrP := 0, nrF := 0, nrI  := 0;

   // *** Minimal load ***
   localVerdict := execute(MyTestCase(
        true,true,true,100,10,nrP,nrF,nrI));

   // *** Medium load ***
   if (localVerdict == pass) {
      localVerdict := execute(MyTestCase(
         true,true,true,400,30,nrP,nrF,nrI));
    }

   // *** Maximal load ***
   if (localVerdict == pass) {
      localVerdict := execute(MyTestCase(
         true,false,true,1000,60,nrP,nrF,nrI));
    }
    return (localVerdict);
}
```




```
function serviceInterworkingTests ()
   return verdicttype {
   var verdicttype localVerdict := pass;
   var integer nrP := 0, nrF := 0, nrI  := 0;

   // *** INTERNET ORDER & PHONE ORDER ***
   localVerdict := execute(MyTestCase(
      true,true,false,2,2,nrP,nrF,nrI),3000.0);

   // *** PHONE ORDER & RESTAURANT ORDER
   if (localVerdict == pass) {
      localVerdict := execute(MyTestCase(
      false,true,true,2,2,nrP,nrF,nrI),3000.0);
   }

   // *** RESTAURANT ORDER & INTERNET ORDER***
   if (localVerdict == pass) {
      localVerdict := execute(MyTestCase(
      true,false,true,2,2,nrP,nrF,nrI),3000.0);
   }

   return (localVerdict);
}
```

Figure E.7/Z.142 – Restaurant example – loadTests and serviceInterworkingTests functions




```
function qualityAssuranceTests ()
   return verdicttype {

   var verdicttype localVerdict := pass;
   var integer nrP := 0,
               nrF := 0,
               nrI  := 0;

   // *** Quality under Minimal load ***
   execute(MyTestCase(true,true,true,100,10,
         nrP,nrF,nrI));

   if ( nrF + nrI > 5 ) {
      localVerdict := fail;
   }

   // *** Quality under Medium load ***
   execute(MyTestCase(true,true,true,400,30,
         nrP,nrF,nrI));

   if ( nrF + nrI > 25 ) {
      localVerdict := fail;
   }

   // *** Quality under Maximal load ***
   execute(MyTestCase(true,false,true,1000,60,
         nrP,nrF,nrI));

   if ( nrF + nrI > 75 ) {
      localVerdict := fail;
   }

   return (localVerdict);

} // end function qualityAssuranceTests
```

Figure E.8/Z.142 – Restaurant example – qualityAssuranceTests function

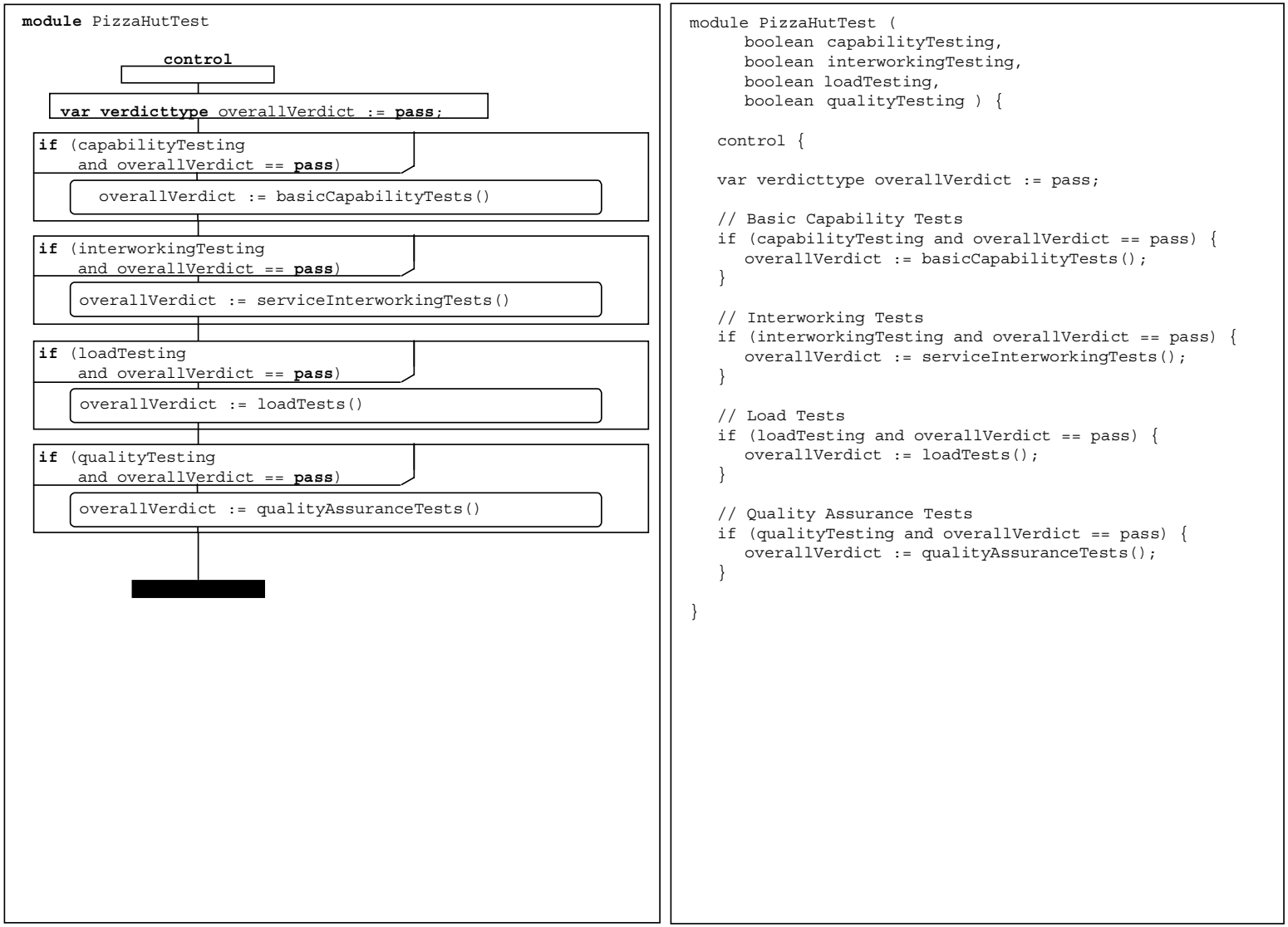
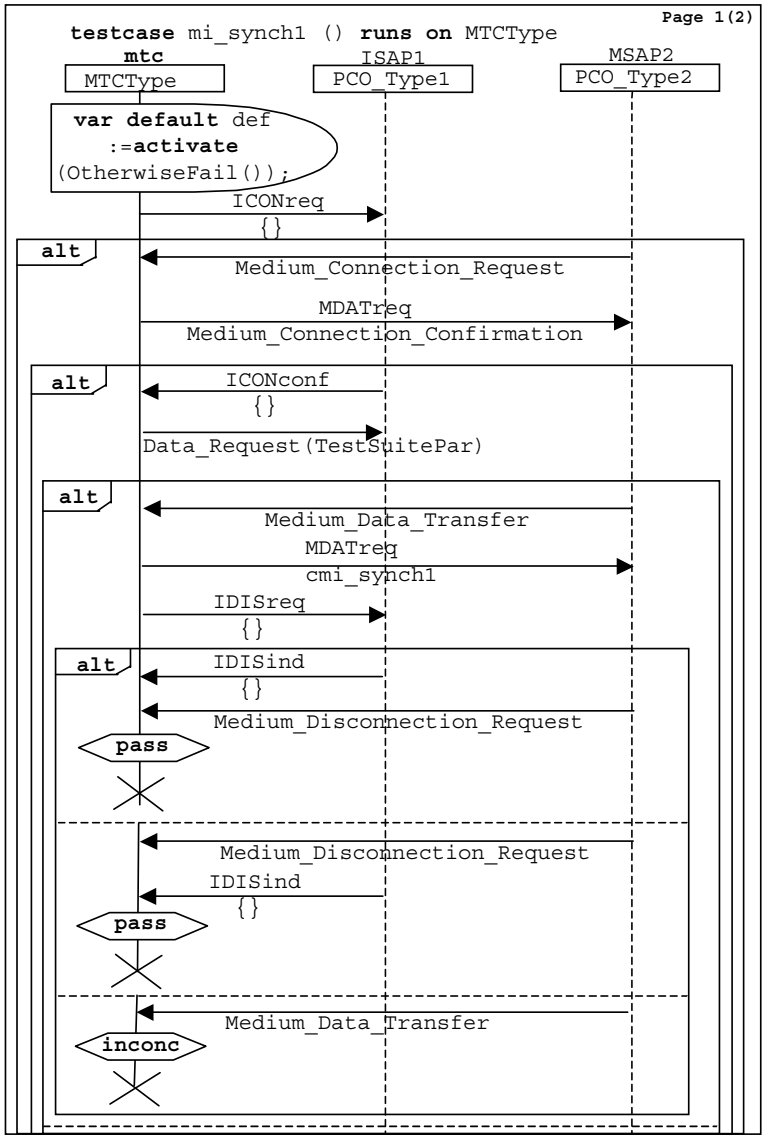


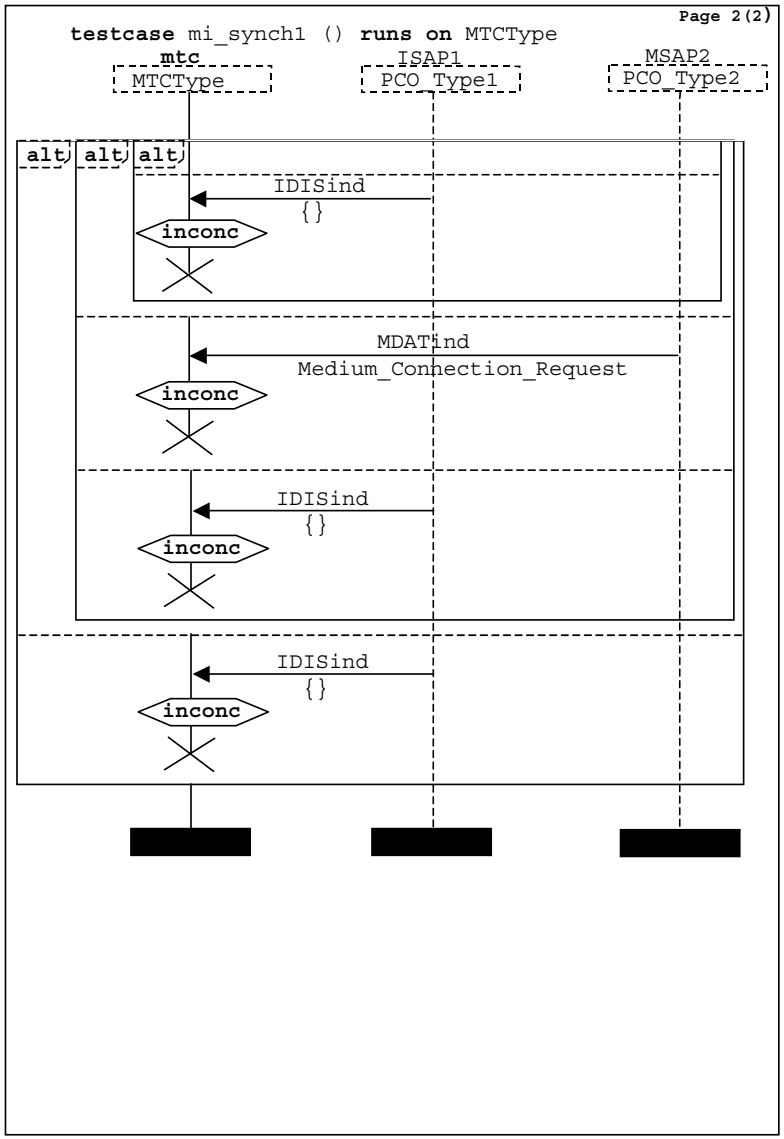
Figure E.9/Z.142 – Restaurant example – PizzaHutTest module

E.2 The INRES example



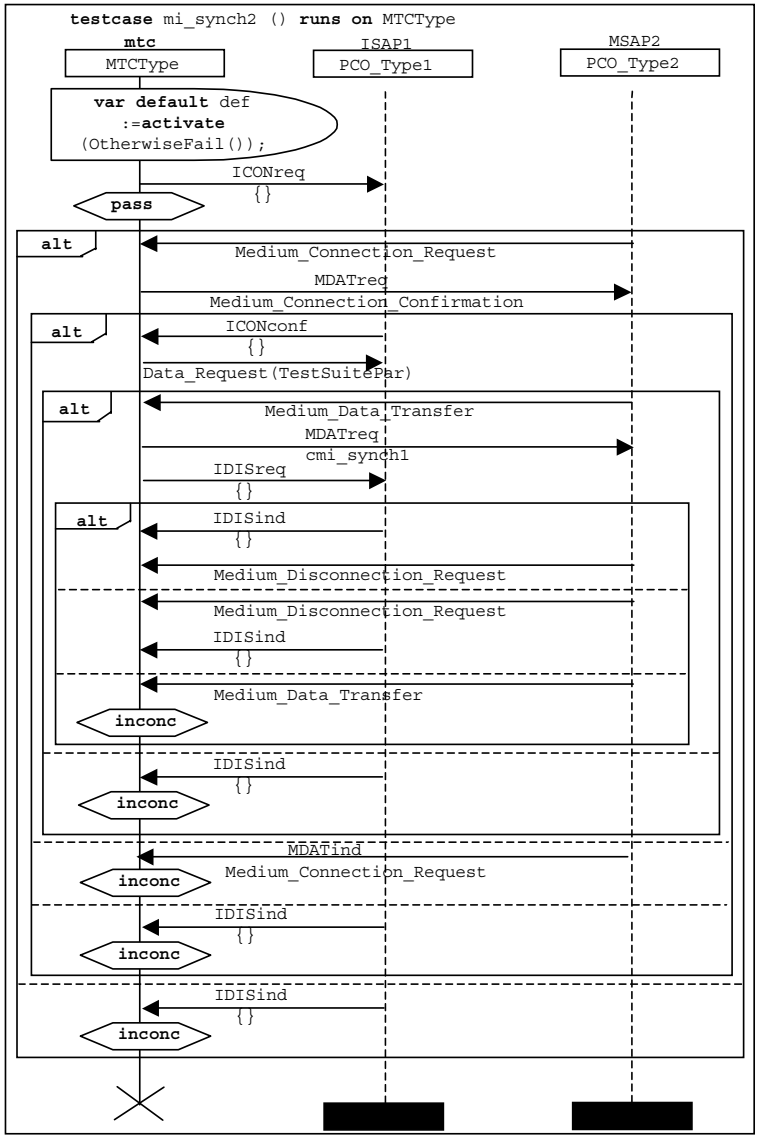

```
testcase mi_synch1 () runs on MTCType {

   /* Default activation */
   var default def := activate(OtherwiseFail());

   /* Inline template definition */
   ISAP1.send( ICONreq:{} );

   alt { /* alt1 */
      [] MSAP2.receive( Medium_Connection_Request ) {
                          /* use of a template */
         MSAP2.send( MDATreq:Medium_Connection_Confirmation );
                          /*optional template type*/

      alt { /* alt2 */
         [] ISAP1.receive ( ICONconf:{} ) {
            ISAP1.send ( Data_Request(TestSuitePar) );


            alt { /* alt3 */
               [] MSAP2.receive( Medium_Data_Transfer ) {
                  MSAP2.send ( MDATreq:cmi_synch1() );
                  ISAP1.send ( IDISreq:{} );



                  alt { /* alt4 */
                     [] ISAP1.receive (IDISind:{} ) {
                        MSAP2.receive(
                              Medium_Disconnection_Request );
                        setverdict(pass);
                        stop;
                     }

                     [] MSAP2.receive(
                              Medium_Disconnection_Request ){
                        ISAP1.receive( IDISind:{} );
                        setverdict(pass);
                        stop;
                     }

                     [] MSAP2.receive(Medium_Data_Transfer ) {
                        setverdict(inconclusive);
                        stop;
                     }
                  } /* end alt4 */
               }
```

Figure E.10/Z.142 – INRES example – mi_synch1 1(2) test case



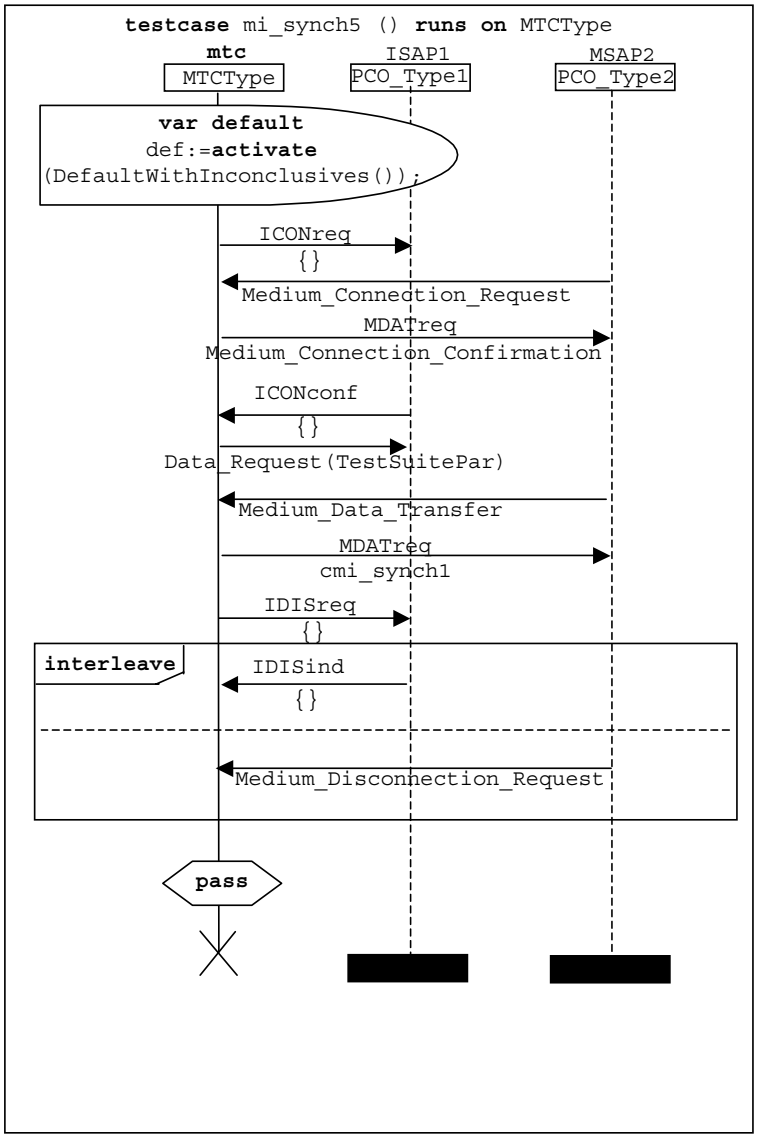

```
/* testcase mi_synch1 () continuation */

              [] ISAP1.receive( IDISind:{} ) {
                 setverdict(inconclusive);
                 stop;
              }
           } /* end alt3 */
        }

        [] MSAP2.receive(
                MDATind:Medium_Connection_Request) {
           setverdict(inconclusive);
        stop;
        }

        [] ISAP1.receive( IDISind:{} ) {
        setverdict(inconclusive);
        stop;
        }
     }
   } /* end alt2 */


[] ISAP1.receive( IDISind:{} ) {
      setverdict(inconclusive);
      stop;
    }
   } /* end alt1 */
 } /* End testcase mi_synch1 */
```

Figure E.11/Z.142 – INRES example – mi_synch1 2(2) test case



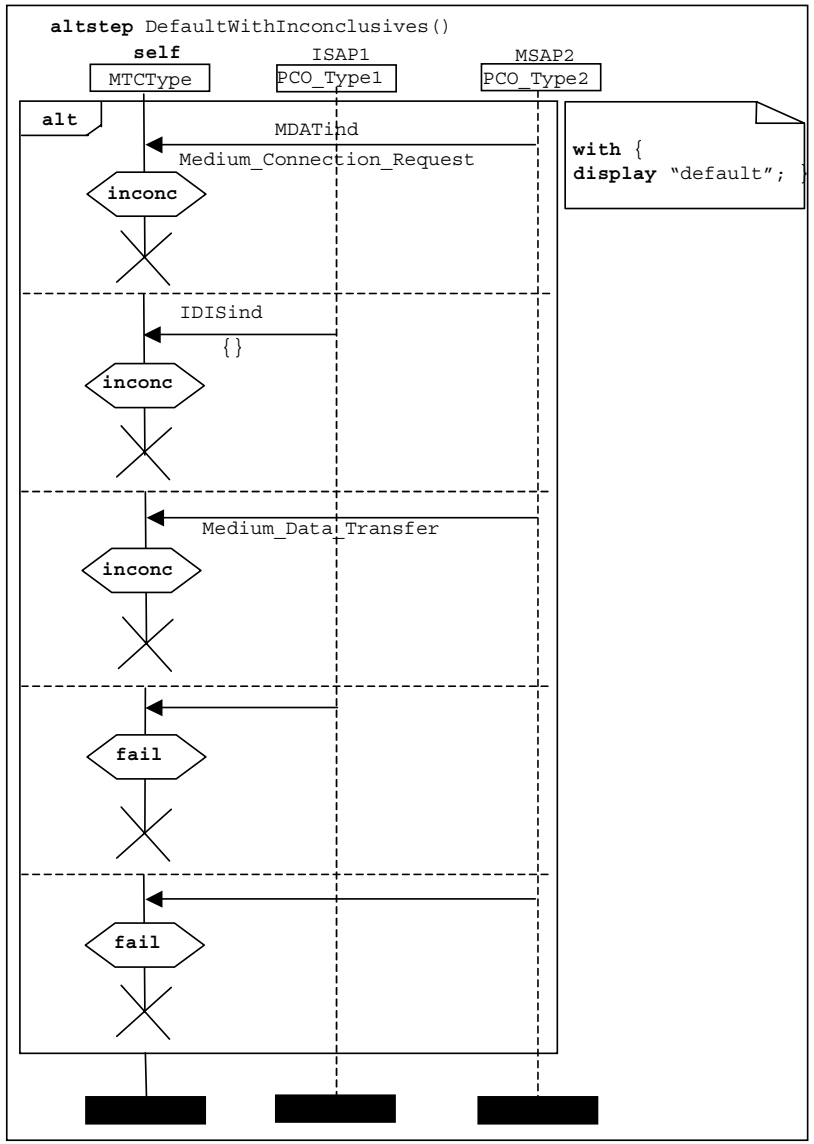

```
testcase mi_synch2 () runs on MTCType {



   var default def := activate(OtherwiseFail());
                      /* Default activation */

   ISAP1.send( ICONreq:{} );
   setverdict(pass);

   alt {
      [] MSAP2.receive( Medium_Connection_Request ) {
         MSAP2.send ( MDATreq:Medium_Connection_Confirmation );
         alt {
            [] ISAP1.receive ( ICONconf:{} ) {
               ISAP1.send ( Data_Request(TestSuitePar) );
               alt {
                  [] MSAP2.receive ( Medium_Data_Transfer ) {
                     MSAP2.send ( MDATreq:cmi_synch1 );
                     ISAP1.send ( IDISreq:{} );
                     alt {
                        [] ISAP1.receive (IDISind:{} ) { /* PASS */
                           MSAP2.receive(
                              Medium_Disconnection_Request );
                        }
                        [] MSAP2.receive(
                              Medium_Disconnection_Request ){
                           ISAP1.receive( IDISind:{} ); /* PASS */
                        }

                        [] MSAP2.receive ( Medium_Data_Transfer ) {
                           setverdict(inconclusive);
                        }
                     }
                  }
                  [] ISAP1.receive( IDISind:{} ) {
                     setverdict(inconclusive);
                  }
               }
            }
            [] MSAP2.receive( MDATind:Medium_Connection_Request ) {
               setverdict(inconclusive);
            }
            [] ISAP1.receive( IDISind:{} ) {
               setverdict(inconclusive);
            }
         }
      }
      [] ISAP1.receive( IDISind:{} ) {
         setverdict(inconclusive);
      }
   }
   stop; } /* End testcase mi_synch2 */
```

Figure E.12/Z.142 – INRES example – mi_synch2 test case



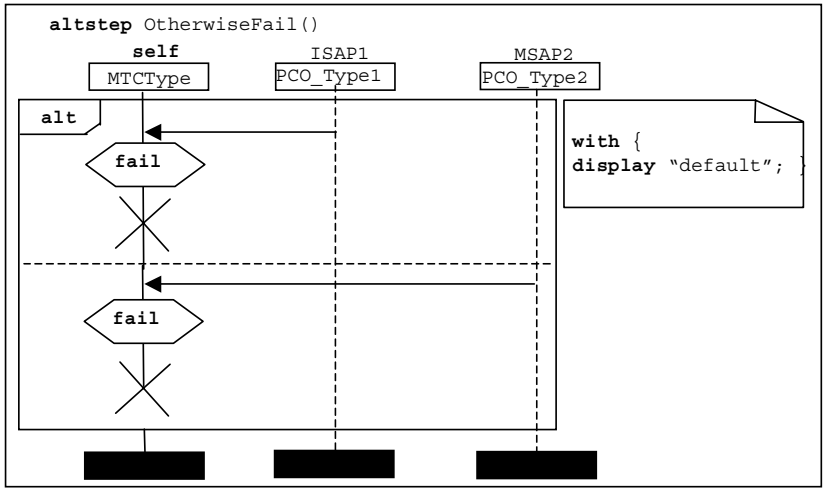

```
testcase mi_synch5 () runs on MTCType {


  var default
       def := activate(DefaultWithInconclusives );
                    /* Default activation */
  /* message ONE and response to ONE */
  ISAP1.send( ICONreq:{} );
  MSAP2.receive(Medium_Connection_Request );

  /* message TWO and response to TWO */
  MSAP2.send(
          MDATreq:Medium_Connection_Confirmation );
  ISAP1.receive ( ICONconf:{} );

  /* message THREE and response to THREE */
  ISAP1.send ( Data_Request(TestSuitePar) );
  MSAP2.receive ( Medium_Data_Transfer );

  /* messages FOUR and FIVE */
  MSAP2.send ( MDATreq:cmi_synch1 );
  ISAP1.send ( IDISreq:{} );

 interleave {
    /* the two responses to messages FOUR and
       FIVE can arrive in any order */
    [] ISAP1.receive(IDISind:{}) {};
    [] MSAP2.receive(
         Medium_Disconnection_Request ) {};
  }

  setverdict(pass);

  stop;

} /* End testcase mi_synch5 */
```

Figure E.13/Z.142 – INRES example – mi_synch5 test case

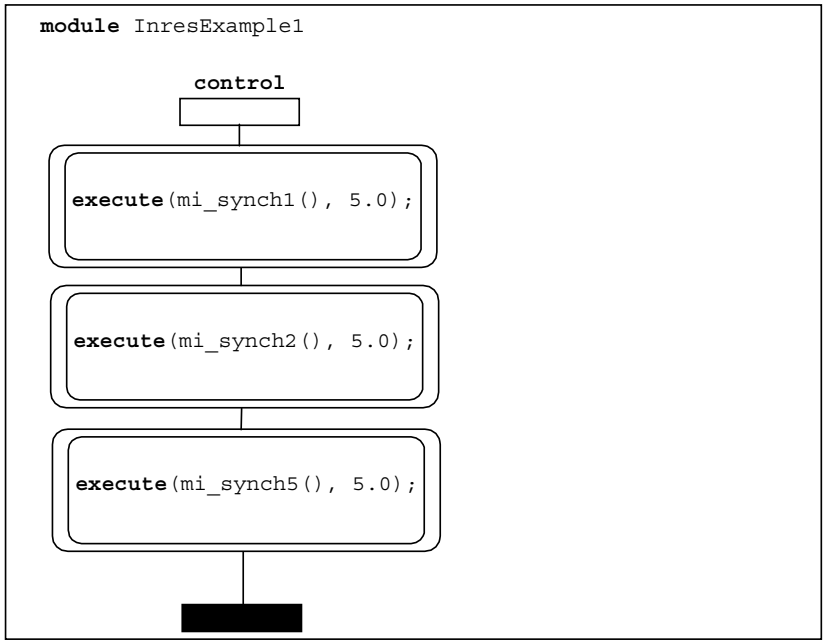



```
altstep DefaultWithInconclusives() {

    /* INCONCLUSIVE CASES */


   [] MSAP2.receive( MDATind:Medium_Connection_Request) {

      setverdict(inconclusive);
      stop;
   }


   [] ISAP1.receive ( IDISind:{} ) {

      setverdict(inconclusive);
      stop;
   }



   [] MSAP2.receive ( Medium_Data_Transfer ) {

      setverdict(inconclusive);
      stop;
   }


  /* FAIL CASES */

   [] ISAP1.receive {

      setverdict(fail);
      stop;

   }


   [] MSAP2.receive {

      setverdict(fail);
      stop;
   }


} with { display "default"; }
```

Figure E.14/Z.142 – INRES example – DefaultWithInconclusives altstep




```
altstep OtherwiseFail() {

   [] ISAP1.receive {

      setverdict(fail);

      stop;
   }

   [] MSAP2.receive {

      setverdict(fail);

      stop;
  }
} with { display "default"; }
```




```
module InresExample1 {
   ...

   control InresExample {

      execute (mi_synch1(), 5.0);

      execute (mi_synch2(), 5.0);

      execute (mi_synch5(), 5.0);

   } // end control part

}
```

Figure E.15/Z.142 – INRES example – OtherwiseFail altstep and InresExample1 module definitions

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 multimedios
Serie I	Red digital de servicios integrados
Serie J	Redes de cable y transmisión de programas radiofónicos y televisivos, y de otras señales multimedios
Serie K	Protección contra las interferencias
Serie L	Construcción, instalación y protección de los cables y otros elementos de planta exterior
Serie M	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, 2006