国 际 电 信 联 盟

ITU-T

国际电信联盟
电信标准化部门

Z.142

(03/2006)

Z系列：电信系统使用的语言和一般性软件情况

正式描述技巧（FDT）— 测试和测试控制记法（TTCN）

测试和测试控制记法第三版（TTCN-3）：图形表示格式（GFT）

ITU-T Z.142建议书

国际电信联盟

ITU-T Z系列建议书

电信系统使用的语言和一般性软件情况

规范和描述语言（SDL）	Z.100-Z.109
正式描述技巧的应用	Z.110-Z.119
信息排序表（MSC）	Z.120-Z.129
扩展的目标描述语言（eODL）	Z.130-Z.139
测试和测试控制记法（TTCN）	**Z.140-Z.149**
用户要求记法（URN）	Z.150-Z.159
编程语言	
CHILL：ITU-T 高级语言	Z.200-Z.209
人机语言	
总则	Z.300-Z.309
基本语法和对话程序	Z.310-Z.319
用于视频显示终端的扩展 MML	Z.320-Z.329
人机接口规范	Z.330-Z.349
面向数据的人机接口	Z.350-Z.359
电信网络管理使用的人机接口	Z.360-Z.379
质量	
电信软件的质量	Z.400-Z.409
涉及协议的建议书中有关质量的内容	Z.450-Z.459
方法	
认证与测试的方法	Z.500-Z.519
中间件	
分布或处理环境	Z.600-Z.609

欲了解更详细信息，请查阅*ITU-T*建议书目录。

ITU-T Z.142建议书

测试和测试控制记法第三版（TTCN-3）：图形表示格式（GFT）

摘　要

本建议书定义了 GFT，ITU-T Z.140 建议书中定义的 TTCN-3（测试和测试控制记法第三版）核心语言的图形表示格式。GFT 以消息序列图的子集形式来图形表示测试行为，如带测试特定扩展的 ITU-T Z.120 建议书所定义的那样。GFT 提供了众多图形符号，以便图形表示 TTCN-3 测试用例、函数、可选步骤和控制部件。无论何时，当需要以图形形式来定义或记录测试行为时，都可以应用 GFT。

本建议书基于 ITU-T Z.140 建议书中定义的 TTCN-3 核心语言。它尤其适于将测试显示为 GFT。它不限于任何特定类型的测试规范。

来　源

ITU-T 第 17 研究组（2005-2008）按照 ITU-T A.8 建议书规定的程序，于 2006 年 3 月 16 日批准了 ITU-T Z.142 建议书。

前　言

国际电信联盟（ITU）是从事电信领域工作的联合国专门机构。ITU-T（国际电信联盟电信标准化部门）是国际电信联盟的常设机构，负责研究技术、操作和资费问题，并且为在世界范围内实现电信标准化，发表有关上述研究项目的建议书。

每四年一届的世界电信标准化全会（WTSA）确定 ITU-T 各研究组的研究课题，再由各研究组制定有关这些课题的建议书。

WTSA 第 1 号决议规定了批准建议书须遵循的程序。

属 ITU-T 研究范围的某些信息技术领域的必要标准，是与国际标准化组织（ISO）和国际电工技术委员会（IEC）合作制定的。

注

本建议书为简明扼要起见而使用的“主管部门”一词，既指电信主管部门，又指经认可的运营机构。

遵守本建议书的规定是以自愿为基础的，但建议书可能包含某些强制性条款（以确保例如互操作性或适用性等），只有满足所有强制性条款的规定，才能达到遵守建议书的目的。“应该”或“必须”等其它一些强制性用语及其否定形式被用于表达特定要求。使用此类用语不表示要求任何一方遵守本建议书。

知识产权

国际电联提请注意：本建议书的应用或实施可能涉及使用已申报的知识产权。国际电联对无论是其成员还是建议书制定程序之外的其它机构提出的有关已申报的知识产权的证据、有效性或适用性不表示意见。

至本建议书批准之日止，国际电联尚未收到实施本建议书可能需要的受专利保护的知识产权的通知。但需要提醒实施者注意的是，这可能并非最新信息，因此特大力提倡他们通过下列网址查询电信标准化局（TSB）的专利数据库：http://www.itu.int/ITU-T/ipr/。

目 录

引言

TTCN-3 的图形表示格式（GFT）基于用于定义消息序列图（MSC）的 ITU-T Z.120 建议书[3]。GFT 使用带测试特定扩展的 MSC 的一个子集。大部分扩展只是文本扩展。定义图形扩展是为了便于阅读 GFT 图形。可能的话，像 MSC 那样来定义 GFT，因此，依据 GFT，对已建立的 MSC 工具稍做修改，即可用来图形化定义 TTCN-3 测试用例。

TTCN-3 核心语言在 ITU-T Z.140 建议书[1]中进行定义，它提供了一整套基于文本的语法、静态语义和操作语义，并定义了如何使用带 ASN.1 的语言。GFT 表示格式提供了一种用于显示核心语言的可选方法（见图 1）。

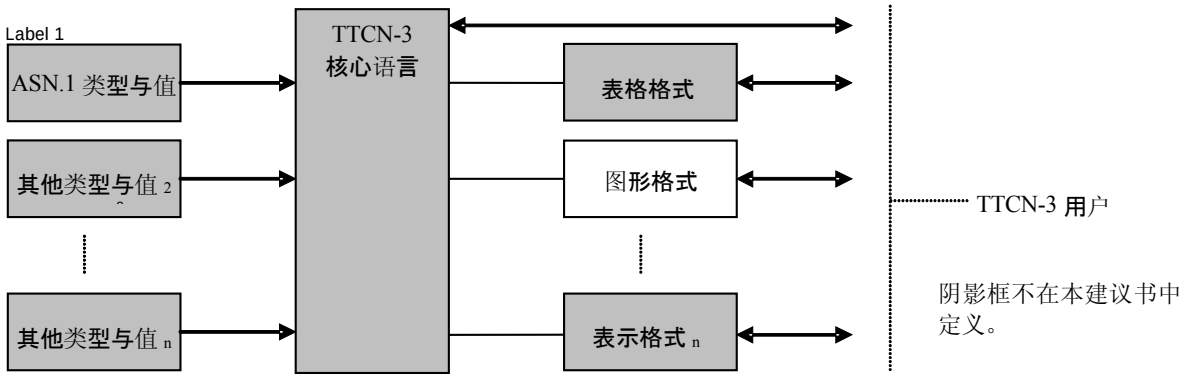


图 1/Z.142－核心语言的用户观点以及各种不同的表示格式

核心语言可以独立于 GFT 使用。不过，如果没有核心语言，那么不能使用 GFT。GFT 的使用和实现基于核心语言。

本建议书定义：

- GFT语言概念；
- GFT使用指南；
- GFT语法；
- 从TTCN-3核心语言以及到TTCN-3核心语言的映射。

这些特性一起形成 GFT — TTCN-3 的图形表示格式。

ITU-T Z.142建议书

测试和测试控制记法第三版（TTCN-3）：图形表示格式（GFT）

1 范围

本建议书定义了 TTCN-3 核心语言的图形表示格式, 如 ITU-T Z.140 建议书[1]中所定义的那样。该表示格式使用了消息序列图的一个子集，如带测试特定扩展的 ITU-T Z.120 建议书[3]所定义的那样。

本建议书基于 ITU-T Z.140 建议书[1]中定义的 TTCN-3 核心语言。它尤其适于将测试显示为 GFT。它不限于任何特定类型的测试规范。

其他格式的规范超出了本建议书的讨论范围。

2 参考文献

下列 ITU-T 建议书和其它参考文献的条款，在本建议书中的引用而构成本建议书的条款。在出版时，所指出的版本是有效的。所有的建议书和其它参考文献均会得到修订，本建议书的使用者应查证是否有可能使用下列建议书或其它参考文献的最新版本。当前有效的 ITU-T 建议书清单定期出版。本建议书引用的文件自成一体时不具备建议书的地位。

[1] ITU-T Recommendation Z.140 (2006), *Testing and Test Control Notation version 3 (TTCN-3): Core language.*

[2] ITU-T Recommendation Z.141 (2006), *Testing and Test Control Notation version 3 (TTCN-3): Tabular presentation format (TFT).*

[3] ITU-T Recommendation Z.120 (2004), *Message sequence chart (MSC).*

[4] ITU-T Recommendation X.292 (2002), *OSI conformance testing methodology and framework for protocol Recommendations for ITU-T applications – The Tree and Tabular Combined Notation (TTCN).*
ISO/IEC 9646-3:1998, *Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 3: The Tree and Tabular Combined Notation (TTCN).*

3 缩写

本建议书使用以下缩写：

BNF	巴科斯范式
CATG	计算机辅助测试生成
GFT	TTCN-3的图形表示格式
MSC	消息序列图
MTC	主测试部件
PTC	并行测试部件
SUT	接受测试的系统
TFT	TTCN-3的表格表示格式
TTCN	测试和测试控制记法

4 概述

根据 ITU-T X.292 建议书[4]中定义的 OSI 一致性测试方法, 测试通常以确定测试目的为起点。测试目的定义为：

“对一个经良好定义的测试目标的普通描述，与适当的 *OSI* 规范中所规定的那样，着重于单一的一致性要求或者一组相关的一致性要求”。

确定所有测试目的之后，开发一个包含一个或多个抽象测试用例的抽象测试套件。抽象测试用例定义测试者的处理行为，验证部分（或全部）测试目的时需要这些行为。

将这些术语应用于消息序列图（MSC），我们可定义其两大类用途：

1） *为定义测试目的而使用MSC*——典型地，作为一个用例或系统规范一部分而形成的MSC规范，可视为一个测试目的，即它以能够被测试的行为描述形式来描述有关SUT的要求。例如，图2表示一个简单的MSC，它描述了表示SUT及其A、B、C三个接口的实例之间相互作用。在这样一个系统的实际执行过程中，接口A、B、C可映射至服务访问点或端口上。图2中的MSC仅描述与SUT的相互作用，并不描述验证SUT行为所需的测试部件行为，即它是一个测试目的描述。

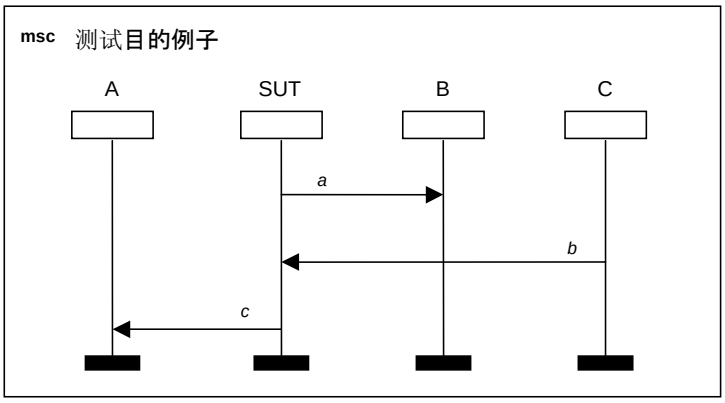


图 2/Z.142－描述SUT与其接口相互作用的MSC

2） *为定义抽象测试用例而使用MSC*——描述抽象测试用例的MSC规范，规定了验证相应测试目的所需的测试部件行为。图3表示一个简单的MCS抽象测试用例描述。它表明，主测试部件（MTC）通过PortA、PortB、PortC与SUT交换消息*a*、*b*和*c*，以实现图2种所示的测试目的。消息*a*和*c*由SUT经端口A和端口B（图2）发送，并被MTC（图3）经相同端口接收。消息*b*由MTC发送，并由SUT接收。

注 — 图2和图3中的例子只是一些简单的例子，以示MSC不同的测试用途。在包括若干过程的分布式SUT或者带有若干测试部件的分布式测试配置中，图将更为复杂。

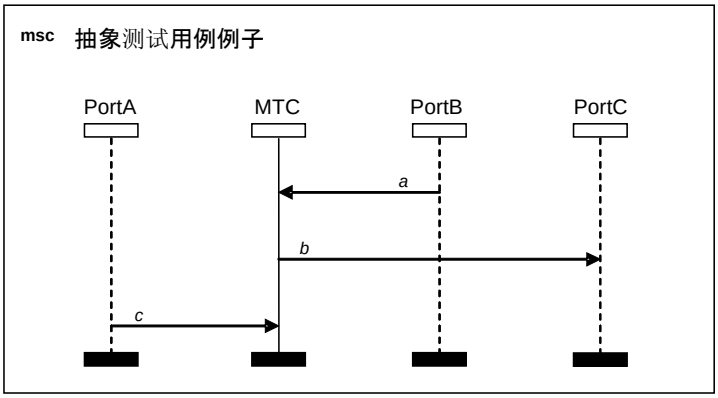


图 3/Z.142－描述MTC与SUT接口相互作用的MSC

在确定 MSC 这两类用法的过程中，可以确定两个不同的工作范围（如图 4 所示）：

a） *自MSC测试目的描述生成抽象测试用例*——TTCN-3核心语言或GFT可用来表示抽象测试用例。 不过，可以看到，自测试目的生成的测试用例是重要的，涉及计算机辅助测试生成（CATG）技术的使用与开发。

b） *为TTCN-3（GFT）开发图形表示格式，并定义GFT与TTCN-3之间的映射。*

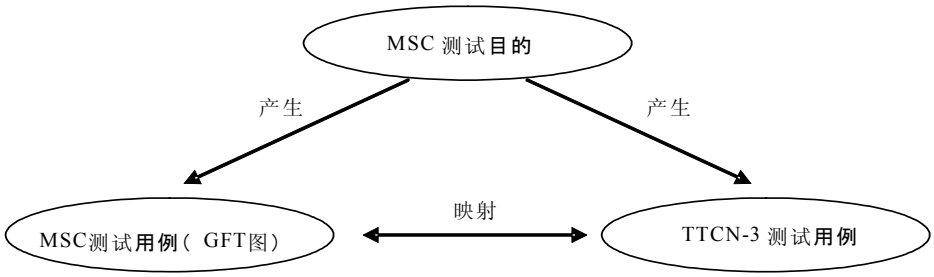


图 4/Z.142－MSC测试目的描述、MSC测试用例描述与TTCN-3之间的关系

本建议书着重于条目 b)，即定义 GFT 以及 GFT 与 TTCN-3 核心语言之间的映射。

5 GFT语言概念

与测试用例或函数的行为一样，GFT 以图形方式来表示 TTCN-3 的行为。它不提供类似类型和模板声明的、有关数据的图形。

GFT 没有定义任何有关 TTCN-3 模块结构的图形表示，但规定了有关这种图形表示的要求（另见第 7 节）。

注 — 在模块定义部分，定义和声明的次序与分组定义了TTCN-3模块的结构。

GFT 没有为以下各项内容定义任何图形表示：

- 模块参数定义；
- 输入定义；
- 类型定义；
- 特征声明；
- 模板声明；
- 常量声明；
- 外部常量声明；以及
- 外部函数声明。

不具备相应 GFT 表示的 TTCN-3 定义和声明，可以以 TTCN-3 核心语言形式或者以 TTCN-3（TFT）表格表示形式来表示（ITU-T Z.141 建议书[2]）。

GFT 为 TTCN-3 行为描述提供了图形。这意味着 GFT 图为以下任意一项提供了图形表示：

- TTCN-3模块的控制部分；
- TTCN-3测试用例；
- TTCN-3函数；或者
- TTCN-3可选步骤。

TTCN-3 模块与相应 GFT 表示之间的关系如图 5 所示。

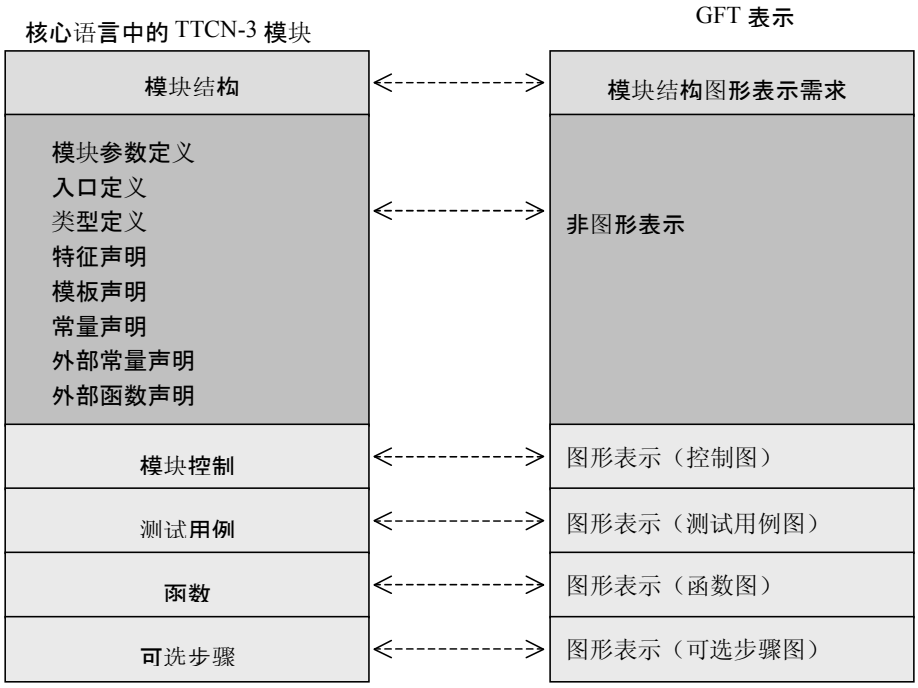


图 5/Z.142－TTCN-3核心语言与相应GFT描述之间的关系

GFT 基于 MSC（ITU-T Z.120 建议书[3]），因此，一个 GFT 图映射于一个 MSC 图上。尽管 GFT 使用大部分图形 MSC 符号，但某些 MSC 符号的题名已适应测试的需要，此外，为强调测试特定的问题，还定义了一些新的符号。不过，这些新的符号可映射于有效的 MSC 上。

- 端口实例的表示；
- 测试部件的创建；
- 测试部件的启动；
- 自函数调用返回；
- 可选步骤的循环；
- 基于过程调用的时间监控；
- 测试用例的执行；
- 缺省的激活与取消激活；
- 标签和跳转；以及
- 在调用语句中的定时器。

在第 8 节中介绍了一个在 GFT 中使用的、所有符号的完整列表。

6 GFT与TTCN-3核心语言之间的映射

GFT 为 TTCN-3 行为定义提供了图形化手段。TTCN-3 核心语言模块的控制部分和各个函数、可选步骤以及测试用例等，可以映射到相应的 GFT 图上，反之亦然。这意味着：

- 模块控制部分可映射到控制图上（参见第9.2节），反之亦然；
- 测试用例可映射到测试用例图上（参见第9.3节），反之亦然；
- 核心语言中的函数可映射到函数图上（参见第9.4节），反之亦然；
- 可选步骤可映射到可选步骤图上（参见第9.5节），反之亦然。

注 1 — GFT不为模块定义部分中的模块参数、类型、常量、特征、模板、外部常量和外部函数的定义提供任何图形表示。这些定义可直接在核心语言中进行表示，或者通过使用其他表示格式，如表格化表示格式来表示。

在模块控制中的各项声明、操作和语句，以及各测试用例、可选步骤或函数，可映射到相应的 GFT 表示上，反之亦然。

在模块控制、测试用例、可选步骤或函数定义中的声明、操作和语句次序，与相关控制、测试用例、可选步骤或函数图中相应的 GFT 表示的次序是一致的。

注 2 — 通过图标题中（仅仅是声明）的GFT构件次序，以及与控制实例（控制图）或部件实例（测试用例图、可选步骤图或函数图）一起的GFT构件次序，来定义GFT图中的GFT构件次序。

7 模块结构

如图 6 所示，TTCN-3 模块具备树结构。TTCN-3 模块由模块定义部分和模块控制部分构成。模块定义部分包括定义和声明，它们可进一步通过分组来构成。模块控制部分不能细分为子结构；它定义测试用例执行的次序与条件。

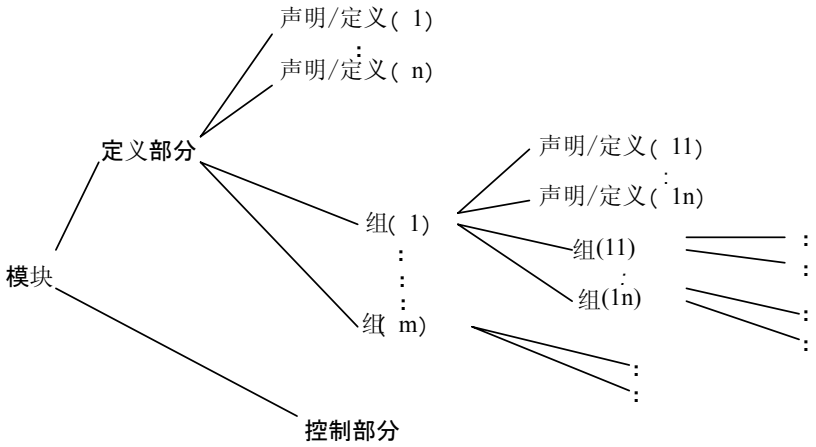


图 6/Z.142－TTCN-3模块结构

GFT 为模块树结构的所有“行为”叶提供图形，即为模块控制部分、函数、可选步骤和测试用例提供图形。GFT 不为模块树结构定义任何具体的图形，不过，对 GFT 的适当工具支持需要 TTCN-3 模块结构的一种图形表示。TTCN-3 模块结构可以以组织者的观点形式（图 7）或 MSC 类文档表示形式（图 8）来提供。一个先进的工具甚至可支持相同对象的不同表示；例如，图 7 中组织者的观点表明，一些定义在若干表示格式中提供：例如，在核心语言中，以 TFT 表格形式以及当作 GFT 图，MySpecialFunction 函数是可用的。

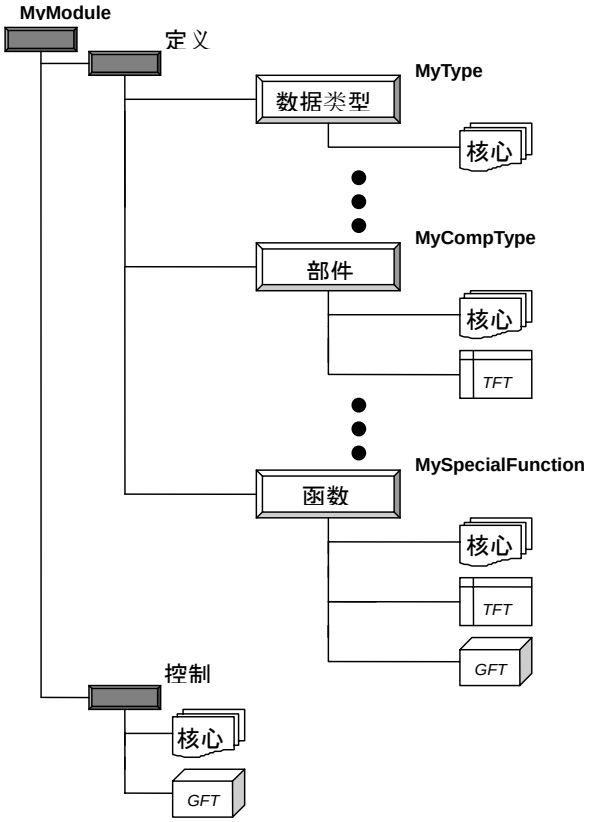


图 7/Z.142－TTCN-3模块结构组建者眼中的各种不同表示格式

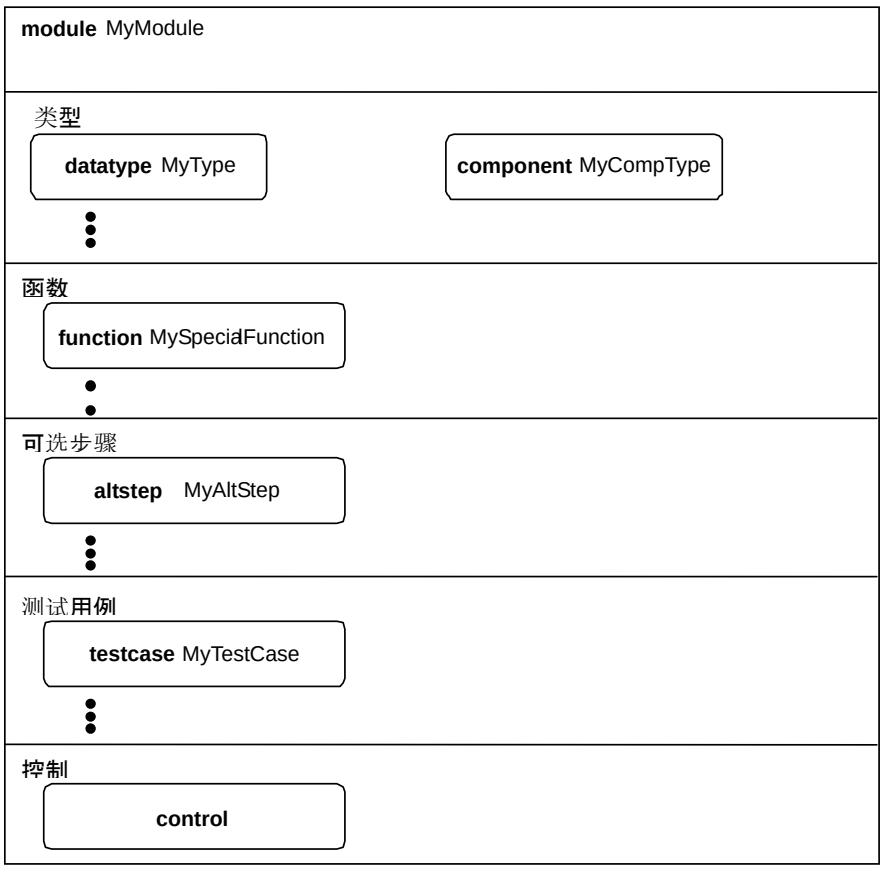


图 8/Z.142－TTCN-3模块结构类似图形MSC文档的表示

8 GFT符号

本节给出了所有在 GFT 图中使用的图形符号，并对其在 GFT 中的典型用法做了注释。

表 1/Z.142－GFT符号

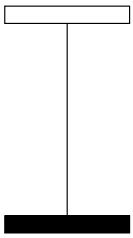
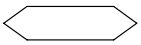
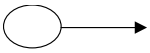
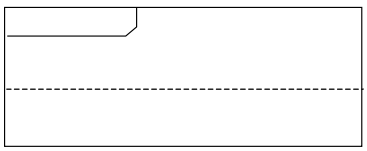
GFT 元素	符　　号	描　　述
框架符号		用于构建 GFT 图。
引用符号		用于表示调用函数和可选步骤。
端口实例符号		用于表示端口实例。
部件实例符号		用于表示测试部件和控制实例。
行为框符号		用于文本形式的 TTCN-3 声明和语句，附于一个部件符号上。
条件符号		用于文本形式的 TTCN-3 布尔表达式、判定设置、端口操作（启动、结束和清除）和完成语句，附于一个部件符号上。
标签符号		用于 TTCN-3 标签和跳转，附于一个部件符号上。
跳转符号		用于 TTCN-3 标签和跳转，附于一个部件符号上。
内嵌表达式符号		用于 TTCN-3 if-else、for、while、do-while、alt、call 和 interleave 语句，附于一个部件符号上。

表 1/Z.142－GFT符号

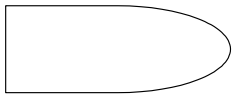
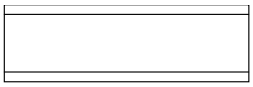
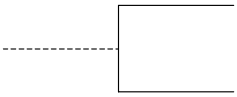
GFT 元素	符　　号	描　　述
缺省符号		用于 TTCN-3 激活和取消激活语句，附于一个部件符号上。
停止符号		用于 TTCN-3 停止语句，附于一个部件符号上。
返回符号		用于 TTCN-3 返回语句，附于一个部件符号上。
重复符号		用于 TTCN-3 重复语句，附于一个部件符号上。
创建符号		用于 TTCN-3 创建语句，附于一个部件符号上。
启动符号		用于 TTCN-3 启动语句，附于一个部件符号上。
消息符号		用于 TTCN-3 send、call、reply、raise、receive、getcall、getreply、catch、trigger 和 check 语句，附于一个部件符号和端口符号上。
找到符号		用于表示来自任何端口的 TTCN-3 receive、getcall、getreply、catch、trigger 和 check，附于一个部件符号上。
暂停区域符号		与一个阻塞调用一起使用，位于一个调用内嵌表达式中，并附于一个部件符号上。
启动定时器符号		用于 TTCN-3 启动定时器操作，附于一个部件符号上。
超时定时器符号		用于 TTCN-3 超时操作，附于一个部件符号上。
停止定时器符号		用于 TTCN-3 停止定时器操作，附于一个部件符号上。
启动隐性定时器符号		用于阻塞调用中 TTCN-3 隐性定时器的启动，位于一个调用内嵌表达式中，并附于一个部件符号上。
超时隐性定时器符号		用于阻塞调用中 TTCN-3 超时异常，位于一个调用内嵌表达式中，并附于一个部件符号上。

表 1/Z.142－GFT符号

GFT 元素	符　　号	描　　述
执行符号		用于 TTCN-3 执行测试用例语句，附于一个部件实例符号上。
文本符号		用于带有语句和注释的 TTCN-3，置于一个 GFT 图中。
事件注释符号		用于事件相关的 TTCN-3 注释，附于有关部件实例或端口实例符号的事件上。

9　GFT图

GFT 提供以下图形类型：

a）　有关TTCN-3模块控制部分图形表示的控制图；

b）　有关TTCN-3测试用例图形表示的测试用例图；

c）　有关TTCN-3可选步骤图形表示的可选步骤图；以及

d）　有关TTCN-3函数图形表示的函数图。

不同类型的图拥有一些公共的属性。

9.1　公共属性

GFT 图的公共属性涉及图区域、图标题和分页。

9.1.1　图区域

各 GFT 控制、测试用例、可选步骤和函数图，都将拥有一个框架符号（也称为图形框架）来定义图区域。定义完整且在语法上正确的 GFT 图所需的所有符号与文字，都将位于图区域内。

注 — 如同MSC控制门，GFT没有任何语言构件，MSC控制门置于图形框架之外，但与之相连。

9.1.2　图标题

各 GFT 图都将拥有一个图标题。图标题将置于图形框架的左上角。

图标题将是各类 GFT 图的唯一标识。实现这一点的一般规则是从关键字 **testcase**、**altstep** 或 **function** 来构造标题，随后是应用图形来表示的测试用例、可选步骤或函数的 TTCN-3 特征。对一个 GFT 控制图，唯一的标题从关键字 **module** 来构造，随后是模块名称。

注 — 在MSC中，关键字**msc**总是处于图形名称之前，以确定MSC图。GFT图没有这种通用的关键字来确定GFT图。

9.1.3 分页

GFT 图可分页构建，一个大型的 GFT 图可分为若干页。对一个分成若干页的图，每一页都将拥有一个页码，它位于右上角，唯一地确定页。如果图未分成若干页，那么页码是可选的。

注 1 — 具体的页码编码方案被认为是一个工具问题，因此它超出了本建议书的讨论范围。一个简单的页码编码方案可以只分配一个页码，而一个高级的页码编码方案可支持图的重构，仅仅通过使用不同页上的页码编码信息即可。

注 2 — 超出一般页码编码的分页要求被认为是工具问题，因此超出了本建议书的讨论范围。出于可读性目的，图标题可以在各页上予以显示，实例的实例线将在另一页上延续，可能与页的底线相连，而延续之实例的实例头可能在该页上重复，以描述这种延续性。

9.2 控制图

GFT 控制图提供了 TTCN-3 模块控制部分的图形表示。控制图的标题将是关键字 **module**，后跟模块名称。GFT 控制图将只包括一个部件实例（也称为控制实例），其实例名称为 **control**，不带任何类型信息。控制实例描述 TTCN-3 模块控制部分的行为。在控制图的文本符号中，将规定 TTCN-3 模块控制部分相关的属性。图 9 勾画了 GFT 控制图的基本形状以及相应的 TTCN-3 核心描述。

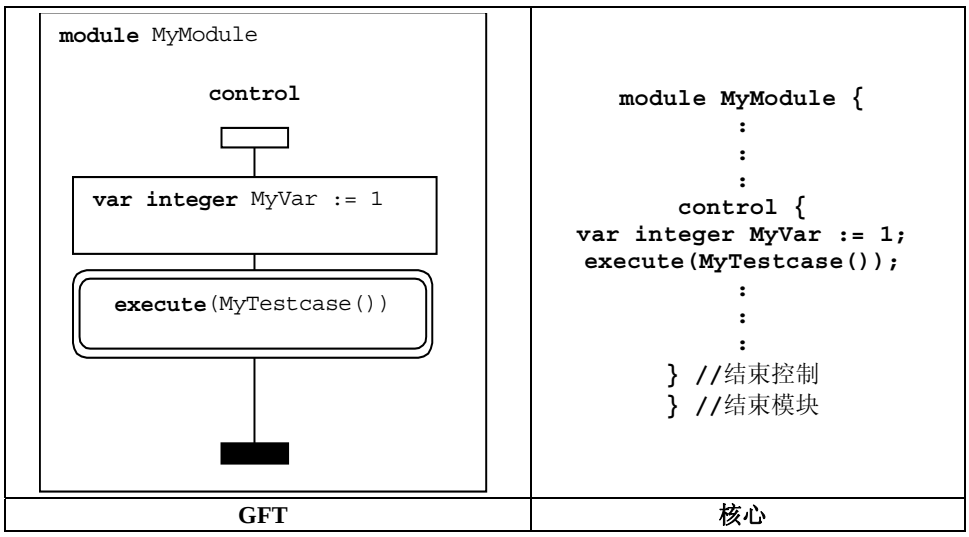


图 9/Z.142－GFT控制图和相应核心语言的原型

在控制部分中，为了执行测试用例，可用布尔表达式选择或取消选择测试用例。表达式、赋值、**log** 语句、**label** 和 **goto** 语句、**if-else** 语句、**for** 循环语句、**while** 循环语句、**do while** 循环语句、**stop** 执行语句，以及定时器语句，都可用于控制测试用例的执行。此外，可使用函数来对测试用例及其执行的先决条件进行分组，它们通过模块控制部分来调用。

这些语言特性的 GFT 表示将分别在以下章节中予以描述，但针对模块控制部分的除外，模块控制部分的图形符号与控制实例相连，而不是与测试部件实例相连。

关于表达式、赋值、**log**、**label** 和 **goto**、**if-else**、**for** 循环、**while** 循环、**do while** 循环和 **stop** 等的 GFT 表示，请参见第 11.4 节，关于定时器操作，请参见第 11.9 节，关于函数及其调用，请参见第 9.4 节和第 11.2.2 节。

9.3 测试用例图

GFT 的测试用例图提供 TTCN-3 测试用例的图形表示。测试用例图的标题将是关键字 **testcase**，后跟测试用例的完整特征。完整意味着至少应存在测试用例名称和参数列表。在核心语言中，**runs on** 子句是强制性的，而 **system** 子句是可选的。如果在相应的核心语言中规定了系统子句，那么还将在测试用例图的标题中提供该子句。

GFT 测试用例图将包括一个测试部件实例和一个端口实例，测试部件实例描述 **mtc**（也称为 mtc 实例）的行为，端口实例针对的是 **mtc** 所有的各端口。mtc 实例相关的名称将为 **mtc**。mtc 实例相关的类型是可选的，但是，如果提出了类型信息，那么它将等同于测试用例特征 **runs on** 子句中提到的部件类型。端口实例相关的名称将等同于 **mtc** 部件类型定义中定义的端口名称。端口实例相关的类型信息是可选的。如果存在类型信息，那么端口名称和端口类型将与 **mtc** 的部件类型定义一致。**mtc** 和端口类型显示在部件或端口实例的标题符号中。

与 GFT 中存在的测试用例相关的各属性将在测试用例图的文本符号中予以说明。图 10 勾画了 GFT 测试用例图的基本形状和相应的 TTCN-3 核心描述。

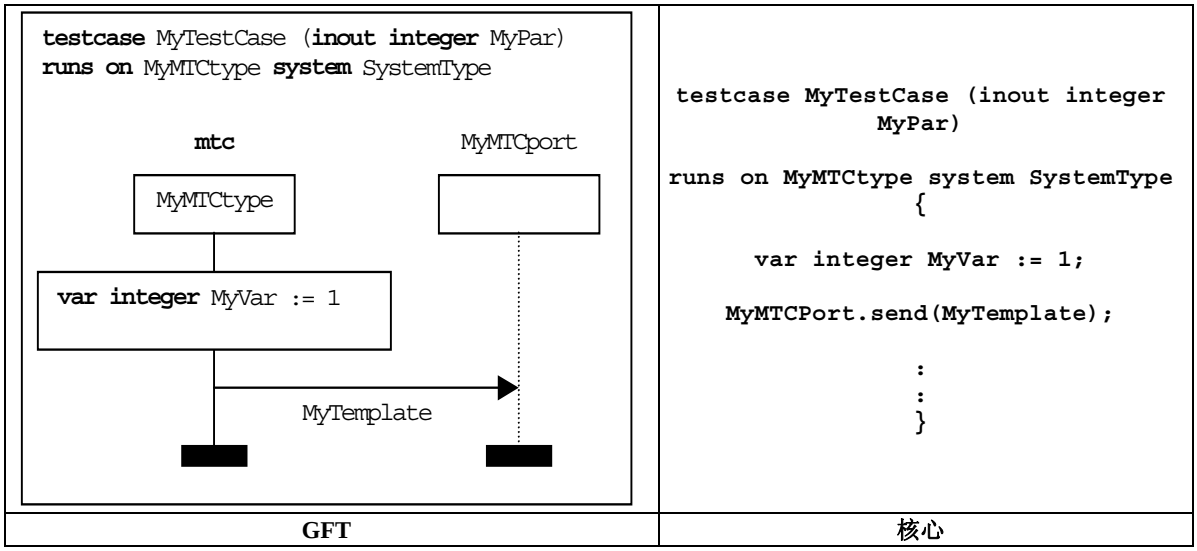


图 10/Z.142－GFT测试用例图和相应核心语言的原型

测试用例表示动态的测试行为，并可创建测试部件。测试用例可包含声明、语句、通信和定时器操作，以及函数或可选步骤的调用。

9.4 函数图

通过函数图，GFT 表示 TTCN-3 函数。函数图的标题将是关键字 **function**，后跟函数的完整特征。完整意味着至少应存在函数名称和参数列表。在核心语言中，**return** 和 **runs on** 子句是可选的。如果在相应的核心语言中规定了这些子句，那么还会在函数图的标题中出现这些子句。

GFT 函数图将包括一个描述函数行为的测试部件实例和一个有关函数可用之各端口的端口实例。

注 — 函数可用之端口的名称和类型都将作为参数进行传递，或者是在**runs on**子句中引用之部件类型定义中定义的端口名称和类型。

测试部件实例相关的名称将为 **self**。测试部件实例相关的类型是可选的，但是，如果存在类型信息，那么它将与 **runs on** 子句中的部件类型相一致。

端口实例相关的名称与类型将与端口参数（如果可用的端口作为参数加以传递）相一致，或者与在 **runs on** 子句中引用之部件类型定义中的端口声明相一致。端口实例的类型信息是可选的。

Self 和端口名称在部件顶部和各自端口实例的标题符号中进行显示。部件类型和端口类型在部件内和各自端口实例的标题符号中进行显示。

在函数图的文本符号中，将规定 GFT 中存在的函数的相关属性。图 11 勾画了 GFT 函数图的基本形状和相应的 TTCN-3 核心描述。

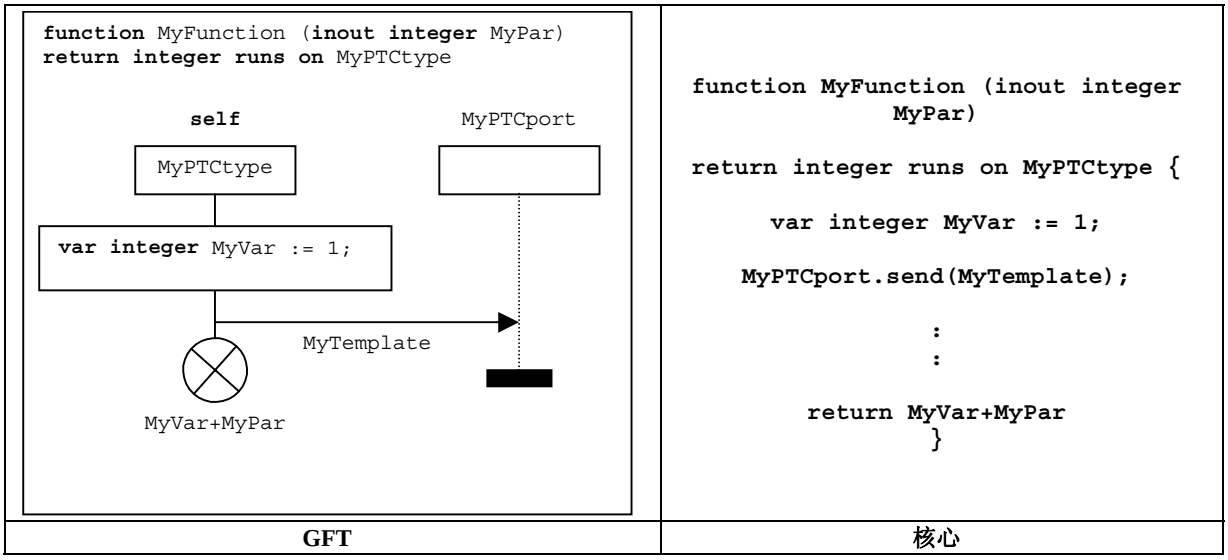


图 11/Z.142－GFT函数图和相应核心语言的原型

函数用来规定和构建测试行为，定义缺省行为，或者构成模块中的计算。函数可包含声明、语句、通信和定时器操作、函数或可选步骤的调用以及可选的返回语句。

9.5 可选步骤图

通过可选步骤图，GFT 表示 TTCN-3 可选步骤。可选步骤图的标题将是关键字 **altstep**，后跟可选步骤的完整特征。完整意味着至少应存在可选步骤名称和参数列表。在核心语言中，**runs on** 子句是可选的。如果在相应的核心语言中规定了 **runs on** 子句，那么还会在可选步骤图的标题中出现该子句。

GFT 可选步骤图将包括一个测试部件实例，该实例描述可选步骤的行为，并包括一个可选步骤可用之各端口的端口实例。

注 — 可选步骤可用之端口的名称和类型作为参数加以传递，或者是在**runs on**子句中引用之部件类型定义中定义的端口名称和类型。

测试部件实例相关的名称将是 **self**。测试部件实例相关的类型是可选的，但是，如果存在类型信息，那么它将与 **runs on** 子句中的部件类型相一致。

端口实例相关的名称与类型将与端口参数（如果可用的端口作为参数加以传递）相一致，或者与在 **runs on** 子句中引用之部件类型定义中的端口声明相一致。端口实例的类型信息是可选的。

self 和端口名称在部件顶部和各自端口实例的标题符号中进行显示。部件类型和端口类型在部件内和各自端口实例的标题符号中进行显示。

在 GFT 可选步骤图的文本符号中将规定与可选步骤相关的属性。图 12 勾画了 GFT 可选步骤图的基本形状和相应的 TTCN-3 核心描述。

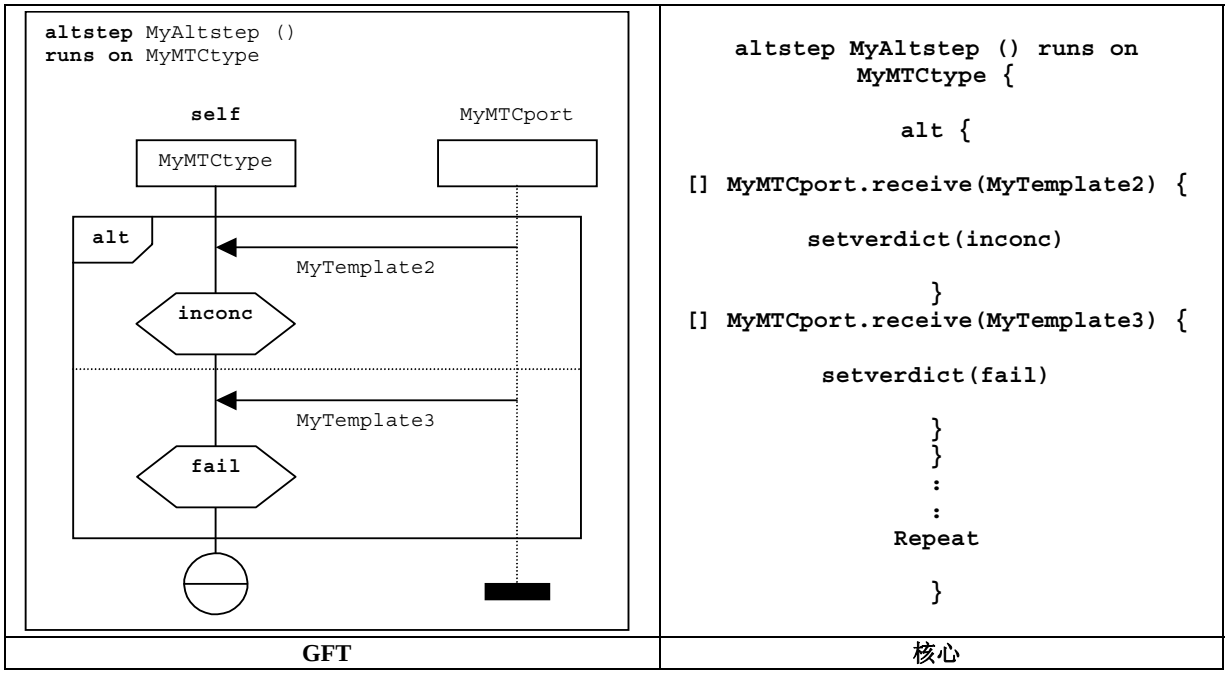


图 12/Z.142－GFT可选步骤图和相应核心语言的原型

可选步骤用于规定缺省行为或构成 **alt** 语句的选项。可选步骤可包含声明、语句、通信和定时器操作，以及函数或可选步骤的调用。

10 GFT图中的实例

GFT 图包括以下类型的实例：

- 描述模块控制部分控制流的*控制实例*；
- 描述测试部件控制流的*测试部件实例*，该测试部件执行一个测试用例、函数或可选步骤；
- 表示不同测试部件使用的端口的*端口实例*。

10.1 控制实例

在 GFT 控制图中（见第 9.2 节），将只存在一个控制实例。控制实例描述模块控制部分的控制流。GFT 控制实例将由带有强制性名称 **control** 的部件实例符号做图形化描述，该强制性名称置于实例标题符号的顶部。 不存在任何与控制实例相关的实例类型信息。图 13 a）显示了控制实例的基本形状。

10.2 测试部件实例

各 GFT 测试用例、函数或可选步骤图包括一个描述该实例控制流的测试部件实例。GFT 测试部件实例将由带以下强制性名称的实例符号做图形化描述：

- 在测试用例图情况下，置于实例标题符号顶部的强制性名称**mtc**；
- 在函数或可选步骤图情况下，置于实例标题符号顶部的强制性名称**self**。

在实例标题符号中，可提供可选的测试部件类型。它必须与在 GFT 图标题中关键字 **runs on** 后提供的测试部件类型相一致。

图 13 b）显示了测试用例图中测试部件实例的基本形状。图 13 c）显示了函数或可选步骤图中测试部件实例的基本形状。

10.3 端口实例

GFT 端口实例可以在测试用例、可选步骤和函数图中使用。端口实例表示可由测试部件使用的端口，该测试部件执行规定的测试用例、可选步骤或函数。GFT 端口实例将由带实例虚线的部件实例符号做图形化描述。所表示端口的名称是强制性信息，并将置于实例标题符号的顶部。在实例标题符号中，可提供（可选的）端口类型。图 13 d）显示了端口实例的基本形状。

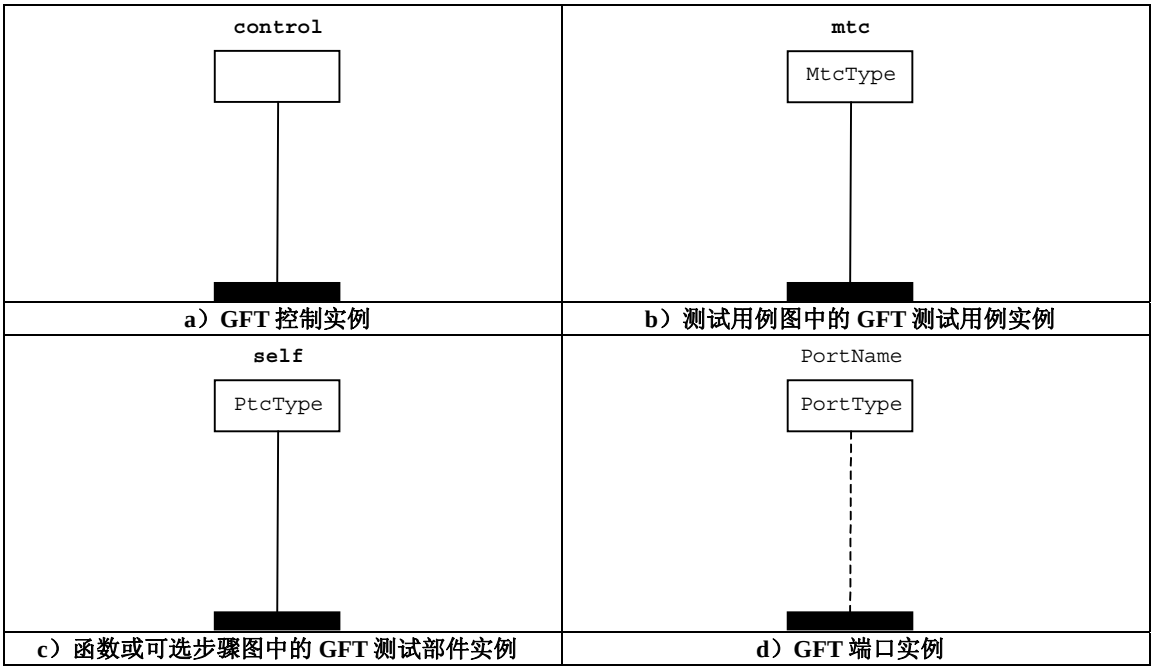


图 13/Z.142－GFT图中实例种类的原型

11 GFT图的元素

本节定义用于表示特定 TTCN-3 语法元素（分号、注释）的一般绘图规则。它描述如何展示 GFT 图的执行过程以及与 TTCN-3 语言元素相关的图形符号。

11.1 一般绘图规则

GFT 中的一般绘图规则与分号的用法、行为符号中的 TTCN-3 语句以及注释有关。

11.1.1 分号的用法

在 TTCN-3 核心语言中，除了行为符号之外，所有的 GFT 符号都将只包括一个语句。只有一个行为符号可包含 TTCN-3 语句的序列（参见第 11.1.2 节）。

如果在 TTCN-3 核心语言中 GFT 符号只包括一个语句，那么分号是可选的（参见图 14 a）和图 14 b））。

分号将一个行为符号内的一系列语句分隔开。对序列中的最后一个语句，分号是可选的（图 14 c））。

在 GFT 图标题后的普通 TTCN-3 核心语言中，还可规定变量、常量和定时器声明的序列。分号也将分隔这些声明。对这一序列中的最后一个声明，分号是可选的。

11.1.2 行为符号的用法

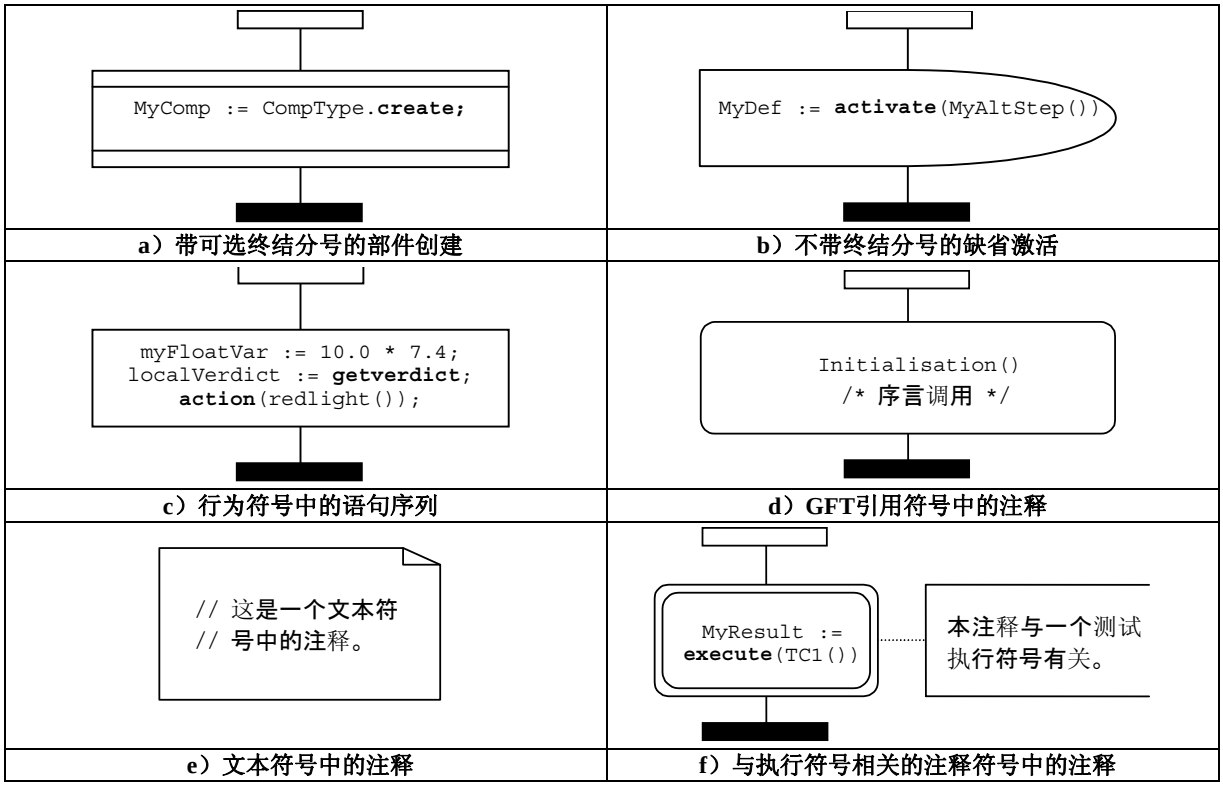
在行为符号中，规定了以下 TTCN-3 声明、语句和操作：声明（带有第 11.3 节中定义的限制）、赋值、**log**、**connect**、**disconnect**、**map**、**unmap** 和 **action**。

将在行为符号变量中规定的声明、语句和操作序列，可以在一个单个的行为符号中予以规定。不必为每个声明、语句或操作使用单独的行为符号。

11.1.3 注释

GFT 提供了以下三种在 GFT 图中添加注释的可能性：

- 可在符号题名后的GFT符号中加入注释，并使用有关TTCN-3核心语言的注释语法（图14 d））。
- 可将TTCN-3核心语言注释语法中的注释加入文本符号中，并自由地置于GFT图区域中（图14 e））。
- 注释符号可用于将注释与GFT符号相连。在注释符号中的注释可以自由文本的形式来提供，即不必使用核心语言的注释定界符“/*”、“*/”和“//”（图14 f））。



a）带可选终结分号的部件创建　　b）不带终结分号的缺省激活

c）行为符号中的语句序列　　d）GFT引用符号中的注释

e）文本符号中的注释　　f）与执行符号相关的注释符号中的注释

图 14/Z.142——一般绘图规则作用的例子

11.2 调用GFT图

本节描述如何调用单个种类的 GFT 图。由于在 TTCN-3 中没有任何有关执行控制部分的语句（因为它通过主程序来执行一个程序是可比的，并超出了 TTCN-3 的范围），本节讨论测试用例、函数和可选步骤的执行。

11.2.1 测试用例的执行

通过使用执行测试用例符号（参见图 15）来表示测试用例的执行。**execute** 语句的语法位于该符号中。该符号可包含：

- 有关带可选参数和时间监控的测试用例的**execute**语句；
- 可选地，向**verdicttype**变量赋予返回的判定；以及
- 可选地，内嵌声明**verdicttype**变量。

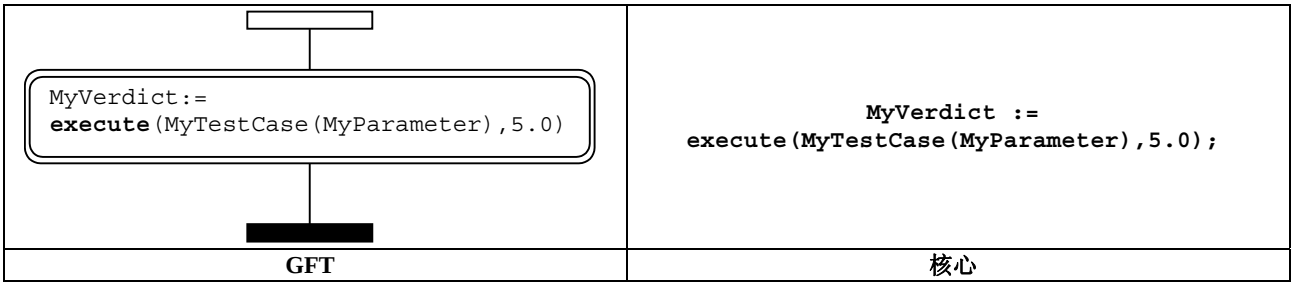


图 15/Z.142－测试用例执行

11.2.2 函数的调用

除了外部的和预先定义的函数（图 17），以及除了在 TTCN-3 语言元素内调用的、具有 GFT 表示法的函数（图 18），函数的调用都由引用符号来表示（图 16）。

函数调用的语法置于引用符号中。该符号可包含：

- 带可选参数的函数调用；
- 向变量可选地赋予返回的值；以及
- 可选地内嵌变量声明。

引用符号仅用作用户定义的函数，而这些函数在当前的模块中定义。它不得用作外部函数或预先定义的 TTCN-3 函数，这些函数将在其行为形式中（图 17）或其他 GFT 符号中（参见图 18 中的例子）以其文本形式加以表示。

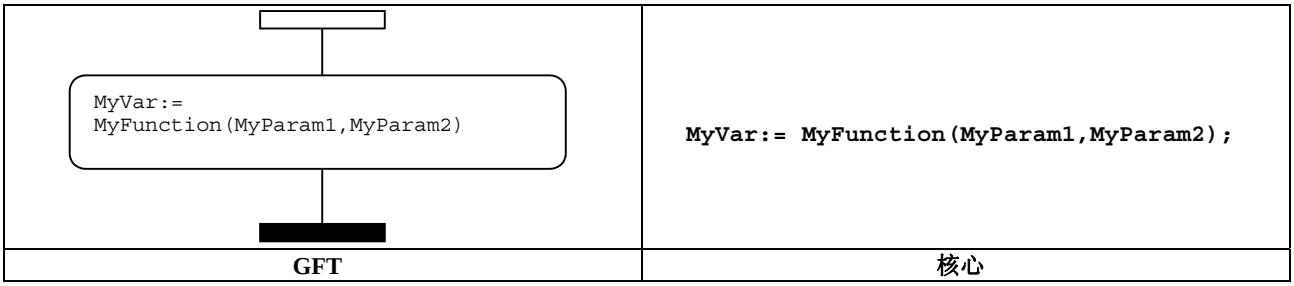


图 16/Z.142－调用用户定义的函数

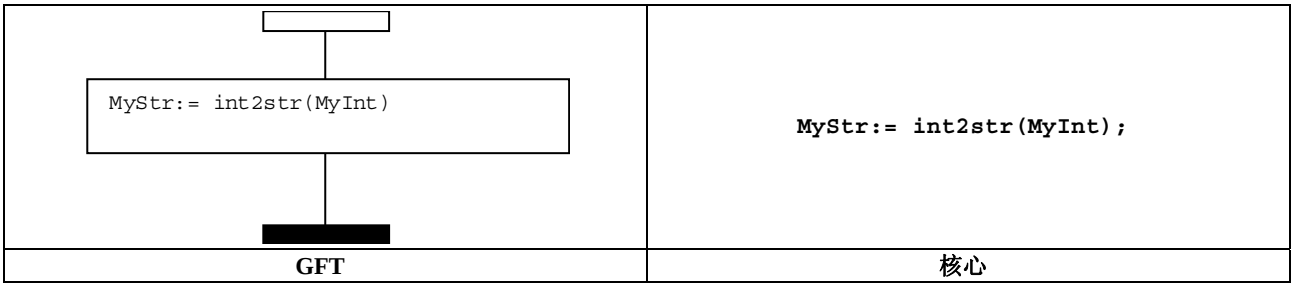


图 17/Z.142－调用预定义/外部函数

在该符号中，在带有相关 GFT 符号的 TTCN-3 结构中调用的函数表示为文本。

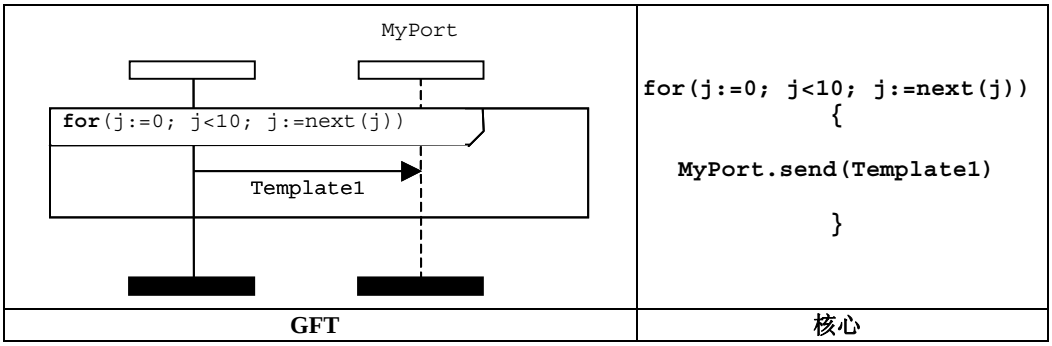


图 18/Z.142－在GFT符号内调用用户定义的函数

11.2.3 可选步骤的调用

通过使用引用符号（参见图 19）来表示可选步骤的调用。可选步骤调用的语法置于该符号中。符号可包含带有可选参数的可选步骤调用；它将只在可选行为中使用，在该行为中，可选步骤调用将是可选语句的操作数之一（也可参见第 11.5.2 节中的图 32）。

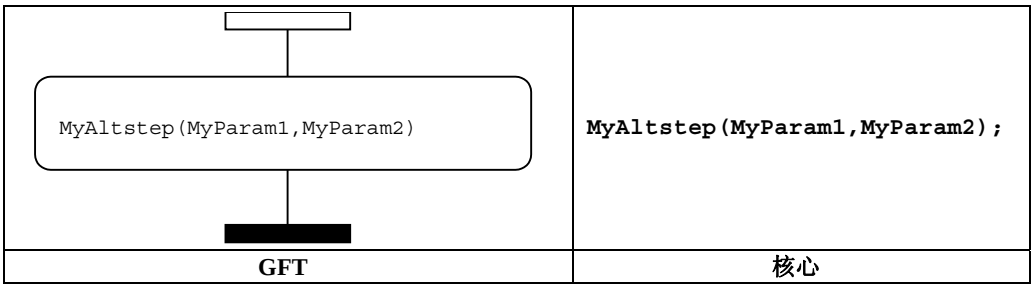


图 19/Z.142－调用可选步骤

另一种可能性是通过激活的缺省隐含地调用可选步骤。详情请参见第 11.6.2 节。

11.3 声明

TTCN-3 允许在语句块的起点声明和初始化定时器、常量与变量。在若干符号中，为了声明，GFT 使用 TTCN-3 核心语言的语法。符号的类型取决于初始化的规范；例如，通过 **activate** 操作来初始化的 **default** 类型的变量将在缺省符号中进行规范（参见第 11.6 节）。

11.3.1 行为符号中定时器、常量和变量的声明

在行为符号中，将做出以下声明：

- 定时器声明；
- 未初始化的变量声明；
- 已初始化的变量和常量声明；
  - — 如果未通过包括通信函数的函数进行初始化；或者
  - — 如果声明是：
    - 未通过**create**操作进行初始化的部件类型；
    - 未通过**activate**操作进行初始化的**default**类型；
    - 未通过**execute**语句进行初始化的**verdicttype**类型；
    - 简单的基本类型；
    - 基本的字符串类型；
    - **anytype**类型；
    - 端口类型；
    - **address**类型；或者
    - 用户定义的结构类型，它带有一些字段，这些字段满足在该着重号中提到的、有关“已初始化的变量与常量声明”的全部限制条件。

注 — 请参见表3/Z.140[1]对TTCN-3类型所做的综述。

将在行为符号中做出的声明序列可置于一个行为符号中，并且无需在单独的行为符号中做出声明。图 20 a）和图 20 b）中展示了一些带行为符号的声明例子。

11.3.2 内嵌表达式符号中常量和变量的声明

在 **if-else**、**for**、**while**、**do-while**、**alt** 或 **interleave** 语句中初始化的部件类型的常量与变量声明，将在相同的内嵌表达式符号中予以表示。

11.3.3 创建符号中常量和变量的声明

通过 **create** 操作初始化的部件类型的常量与变量声明，将在一个创建符号中做出。与行为符号中的声明形成对比，每个通过 **create** 操作进行初始化的声明，都将在单独的创建符号中予以表示。创建符号中变量声明的一个例子如图 20 c）所示。

11.3.4 缺省符号中常量和变量的声明

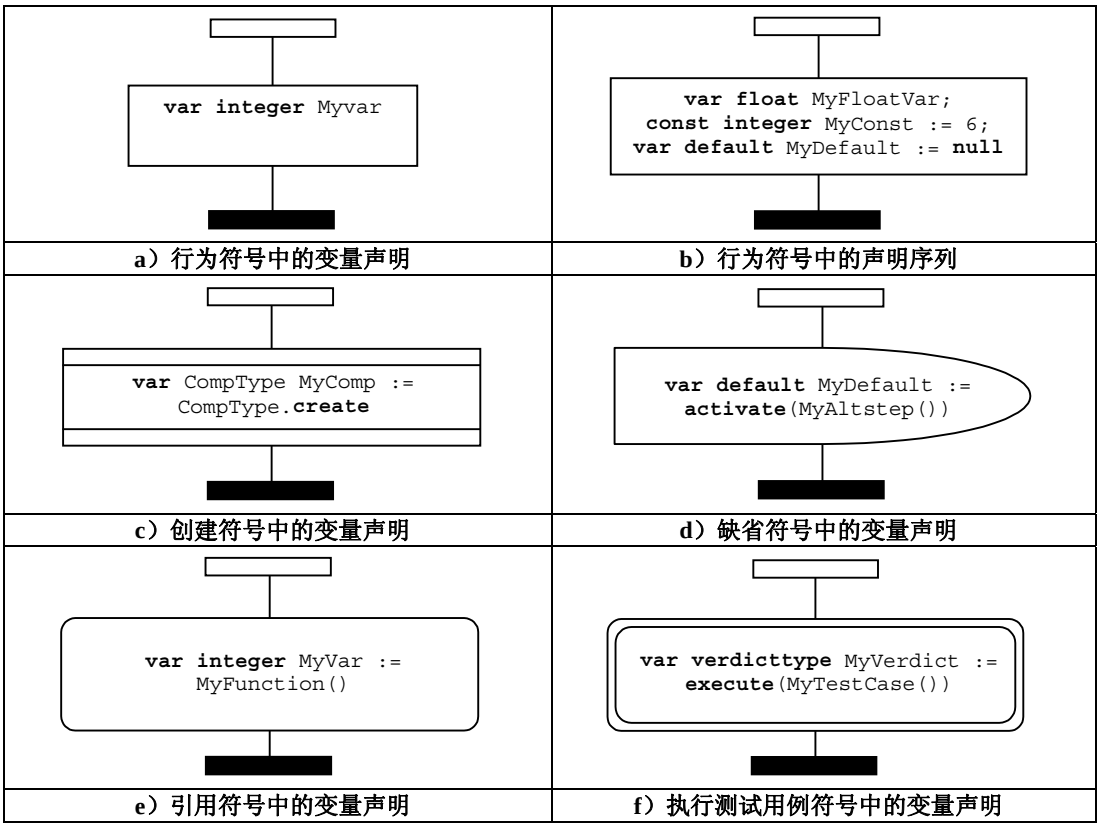
通过 **activate** 操作初始化的 **default** 类型的常量与变量声明，将在缺省符号中做出。与行为符号中的声明形成对比，每个通过 **activate** 操作进行初始化的声明，都将在单独的缺省符号中予以表示。缺省符号中变量声明的一个如图 20 d）所示。

11.3.5 引用符号中常量和变量的声明

通过函数进行初始化的常量与变量声明，包括通信操作，将在引用符号中做出。与行为符号中的声明形成对比，每个通过函数进行初始化的声明，包括通信函数，都将在单独的引用符号中予以表示。引用符号中变量声明的一个例子如图 20 e）所示。

11.3.6 执行测试用例符号中常量和变量的声明

通过 **execute** 语句初始化的 **verdicttype** 类型的常量与变量声明，将在执行测试用例符号中做出。与行为符号中的声明形成对比，每个通过 **execute** 语句进行初始化的声明，都将在单独的执行测试用例符号中予以表示。在执行测试用例符号中的变量声明的一个如图 20 f）所示。



a）行为符号中的变量声明　　b）行为符号中的声明序列

c）创建符号中的变量声明　　d）缺省符号中的变量声明

e）引用符号中的变量声明　　f）执行测试用例符号中的变量声明

图 20/Z.142－GFT中声明举例

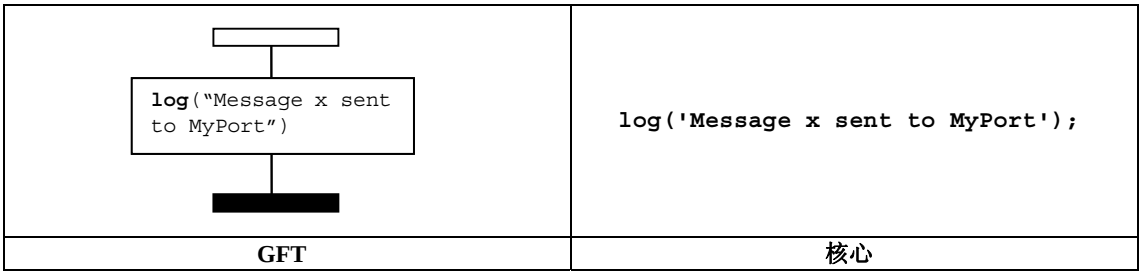
11.4 基本编程语句

基本编程语句是表达式、赋值、操作、循环结构等。所有的基本编程语句都可以在 GFT 图中用作控制部分、测试用例、函数和可选步骤。

GFT 不为表达式和赋值提供任何图形表示。在其使用的地方，它们用文本进行表示。为 **log**、**label**、**goto**、**if-else**、**for**、**while** 和 **do-while** 语句提供图形。

11.4.1 日志语句

log 语句将在行为符号中予以表示（参见图 21）。



GFT　　核心

图 21/Z.142－日志语句

11.4.2 标签语句

label 语句将用标签符号进行表示，它与一个部件实例相连。图 22 显示了一个名为 MyLabel 的 **label** 的简单例子。

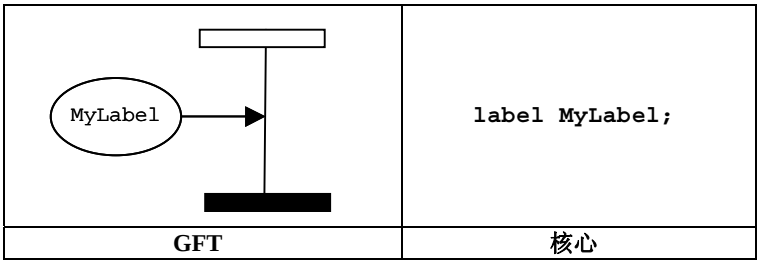


图 22/Z.142－标签语句

11.4.3 跳转语句

goto 语句将用跳转符号进行表示。它将置于部件实例的结尾处或者置于内嵌表达式符号中的操作数的结尾处。图 23 显示了一个 **goto** 的简单例子。

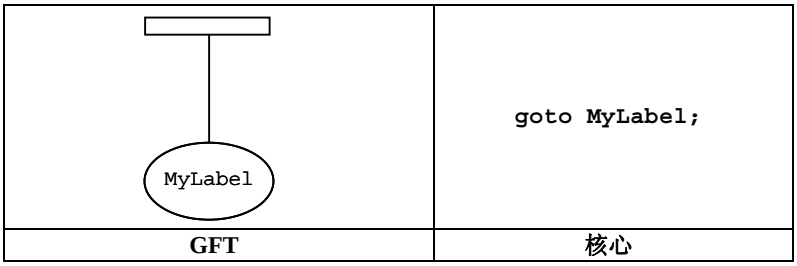


图 23/Z.142－跳转语句

11.4.4 if-else语句

如第 19.6 节/Z.140[1]中所定义的那样，将通过一个标有关键字 **if** 的内嵌表达式符号和一个布尔表达式来表示 **if-else** 语句。if-else 内嵌表达式符号可包含一个或两个操作数，用虚线分隔开。图 24 显示了带单个操作数的 **if** 语句，当布尔表达式 x > 1 计算为 TRUE 时，执行该语句。图 25 显示了 **if-else** 语句，其中，当布尔表达式 x > 1 计算为 TRUE 时，执行顶部操作数，当布尔表达式计算为 FALSE 时，执行底部操作数。

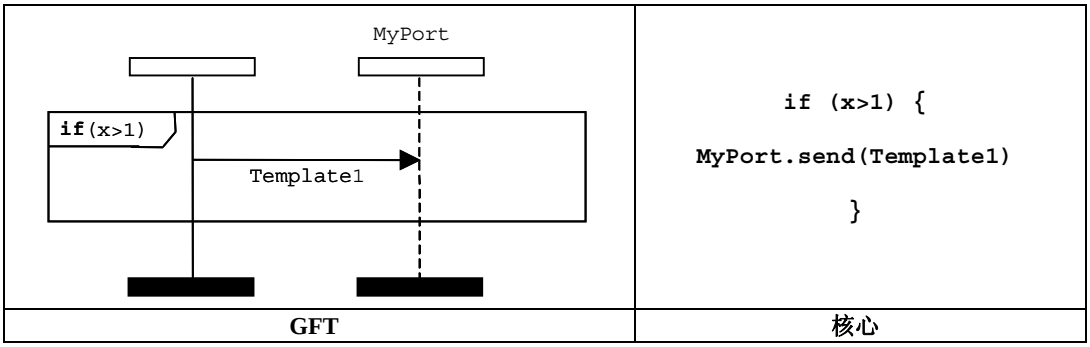


图 24/Z.142－If语句

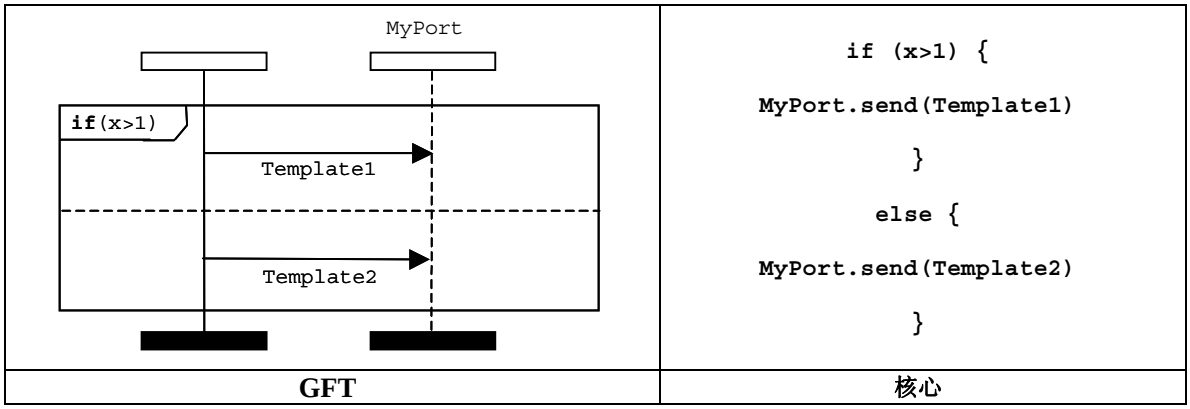


图 25/Z.142－If-else语句

11.4.5 For 语句

如第 19.7 节/Z.140[1]中所定义的那样，将通过一个标有 **for** 定义的内嵌表达式符号来表示 **for** 语句。**for** 循环体将作为内嵌表达式符号的操作数加以表示。图 26 表示一个简单的 **for** 循环，其中，循环变量在 **for** 语句中进行声明和初始化。

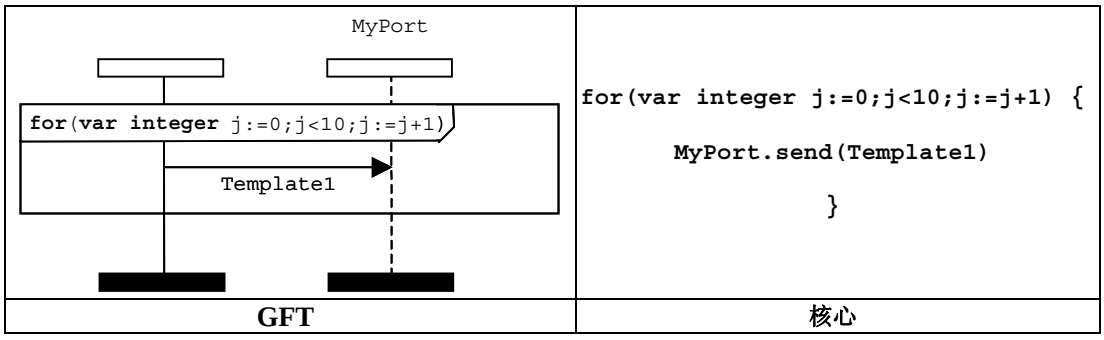


图 26/Z.142－For语句

11.4.6 While 语句

如第 19.8 节/Z.140[1]中所定义的那样，将通过一个标有 **while** 定义的内嵌表达式符号来表示 **while** 符号。**while** 循环体将作为 while 内嵌表达式符号的操作数加以表示。图 27 表示一个 **while** 语句的例子。

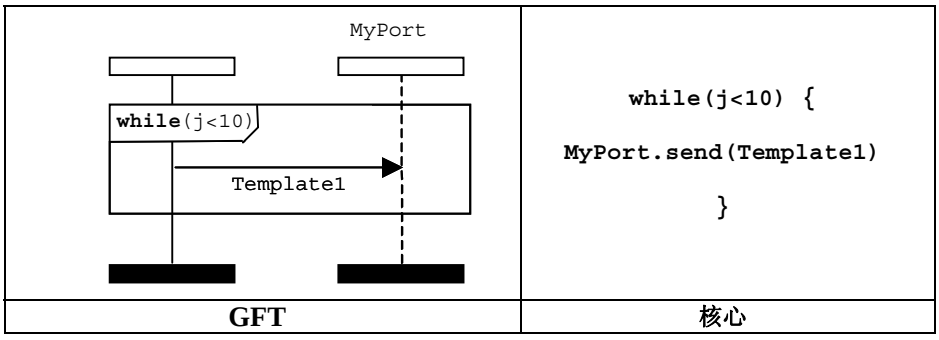


图 27/Z.142－While语句

11.4.7 Do-while 语句

如第 19.9 节/Z.140[1]中所定义的那样，将通过一个标有 **do-while** 定义的内嵌表达式符号来表示 **do-while** 语句。**do-while** 循环体将作为 do-while 内嵌表达式符号的操作数加以表示。图 28 表示一个 **do-while** 语句的例子。

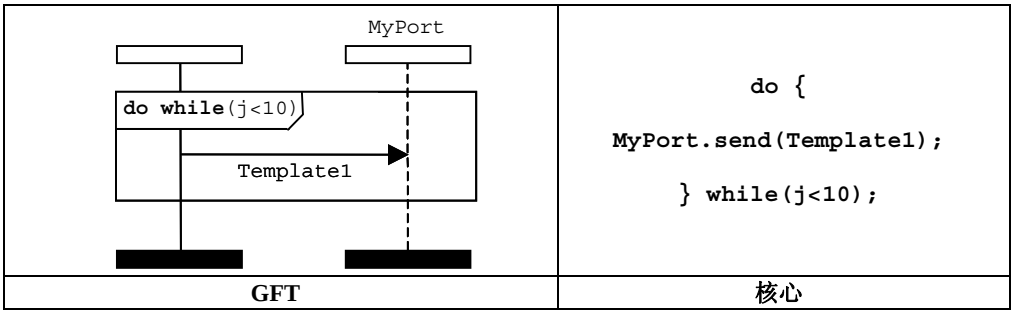


图 28/Z.142－Do-while语句

11.5 行为编程语句

行为编程语句可用在测试用例、函数和可选步骤中，唯一的例外是返回语句，它只能用在函数中。测试行为可以顺序表示，作为一组选项或使用一个交叉的语句。返回和循环用于控制行为流。

11.5.1 顺序行为

顺序行为通过加在测试部件实例上的事件次序来表示。事件的次序采用自上而下的方式，离部件实例符号顶部最近的事件当作第一个来计算。图 29 显示了一个简单的例子，其中，测试部件首先计算包含在行为符号中的表达式，然后向端口 MyPort 发送一个消息。

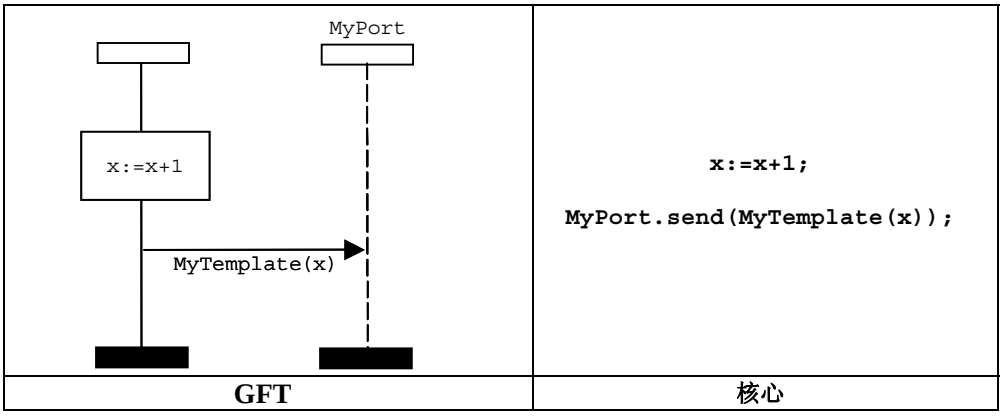


图 29/Z.142－顺序行为

排序也可通过引用测试用例、函数和可选步骤来描述。在这种情况下，引用被置于部件实例轴线上的次序决定了它们在其中进行计算的次序。图 30 表示一个简单的 GFT 图，其中，首先调用 MyFunction1，然后调用 MyFunction2。

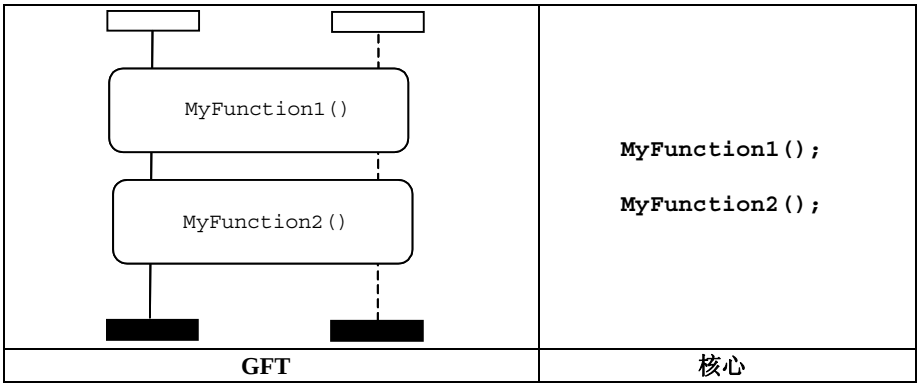


图 30/Z.142－使用引用排序

11.5.2 可选行为

将通过带有 **alt** 关键字的内嵌表达式符号来表示可选行为，**alt** 关键字置于左上角的顶部。可选行为的各操作数将通过虚线来分隔。自上而下对操作数进行计算。

注意：如果调用通信操作符，那么可选的内嵌表达式应总包括所有的端口实例。图 31 显示了可选行为，其中，一个消息事件要么由 Template1 定义的值接收，要么由 Template2 定义的值接收。图 32 显示了可选的内嵌表达式中可选步骤的调用。

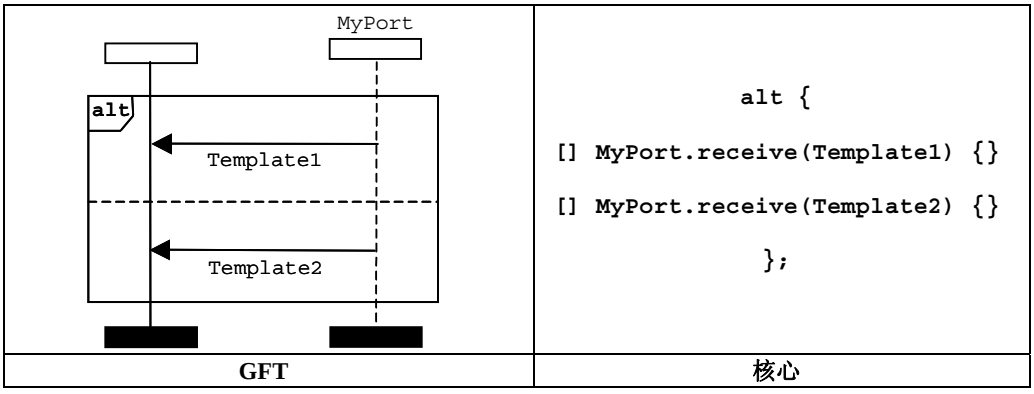


图 31/Z.142－可选的行为语句

此外，也可能将可选步骤作为可选操作数中的唯一事件来调用。这将通过引用符号来描绘（参见第 11.2.3 节）。

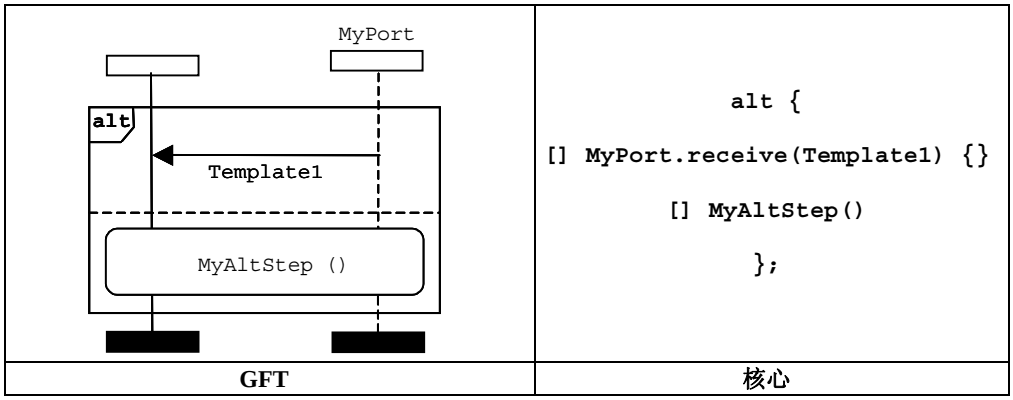


图 32/Z.142－带可选步骤调用的可选行为

11.5.2.1 选项的选择/取消选择

有可能通过包含在条件符号中的布尔表达式来取消使能/使能可选操作数，布尔表达式置于测试部件实例上。图 33 显示了一个简单的可选语句的例子，其中，第一个操作数用表达式 $x > 1$ 加以保护，第二个用表达式 $x \leq 1$ 加以保护。

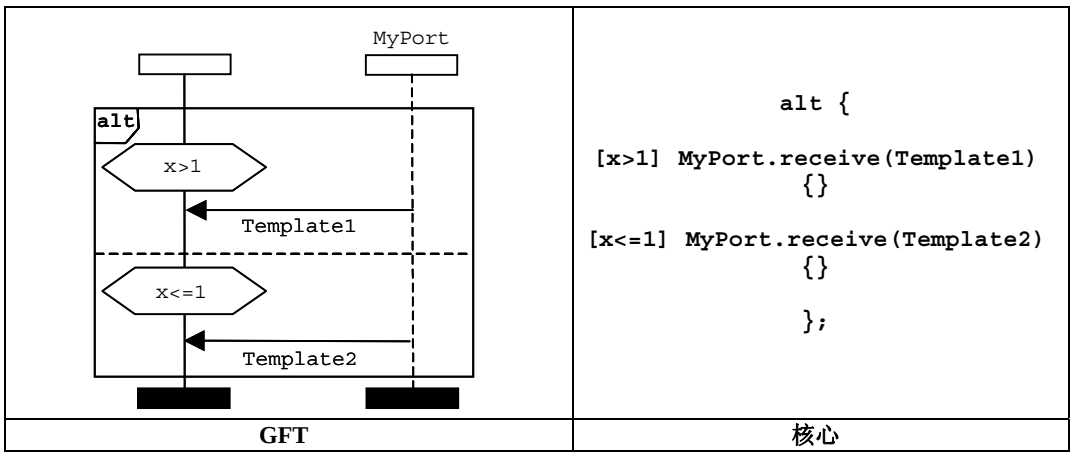


图 33/Z.142－选择/取消选择一个可选项

11.5.2.2 选项中的Else分支

else 分支将通过置于测试部件实例轴线上的条件符号来表示，它标有关键字 **else** 的。图 34 显示了一个简单的可选语句，其中，第二个操作数表示 **else** 分支。

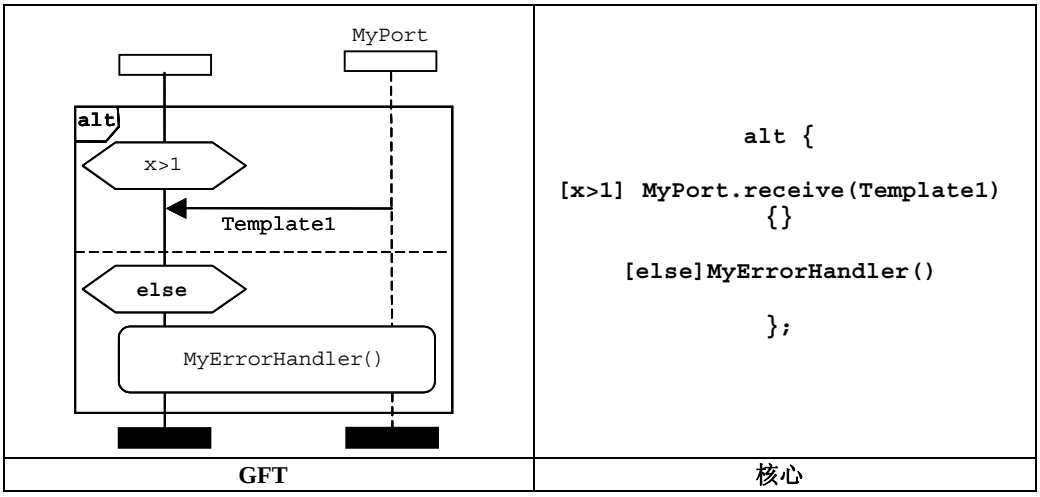


图 34/Z.142－一个可选项中的Else

注意：如果调用通信操作，那么在 else 分支中的引用符号应总包括所有的端口实例。

使用一个循环语句，可以规定对 alt 语句的重新计算，该循环语句通过循环符号来表示（参见第 11.5.3 节）。

使用引用符号，可以表示选项内可选步骤的调用（参见第 11.2.3 节）。

11.5.3 Repeat语句

repeat 语句将通过循环符号来表示。该符号将只用作 **alt** 语句中可选操作数的最后事件，或者用作可选步骤定义中顶部选项操作数的最后事件。图 35 显示了一个可选语句，其中，若成功接收带有与 Template2 匹配的值的消息，则第二个操作数将导致选项循环。

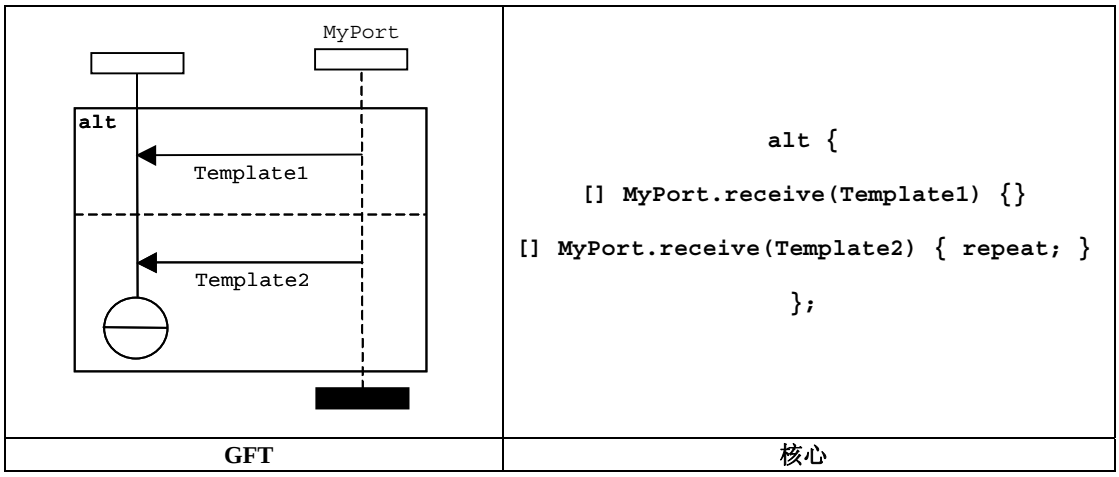
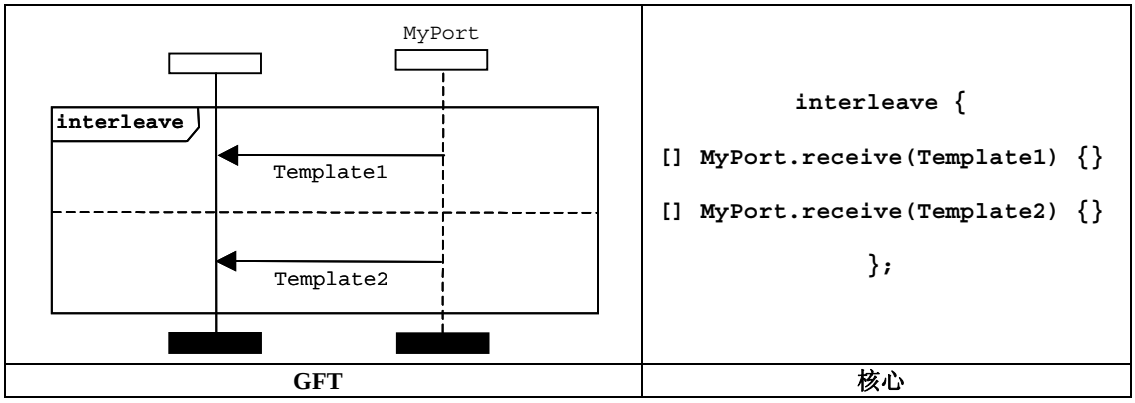


图 35/Z.142－一个可选项中的重复

11.5.4 插入的行为

将通过带有关键字 **interleave** 的内嵌表达式符号来表示交叉行为，关键字 **Interleave** 置于左上角的顶部（参见图 36）。各操作数都将使用虚线来分隔。自上而下对操作数进行计算。



注 — 如果涉及通信运营商，那么一个插入内嵌表达式应总是覆盖所有的端口实例。

图 36/Z.142－插入语句

11.5.5 Return语句

return 语句将通过返回符号来表示。这可能可选地与返回值有关。返回符号将只用在 GFT 函数图中。它将只用作部件实例的最后事件，或者用作内嵌表达式符号中操作数的最后事件。图 37 显示了一个使用不带返回值的返回语句的简单函数，而图 38 显示了一个返回值的函数。

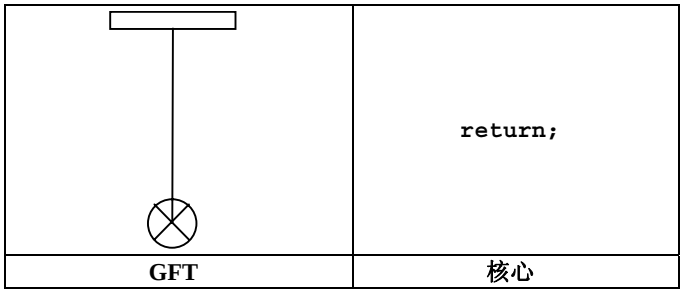


图 37/Z.142－不带返回值的返回符号

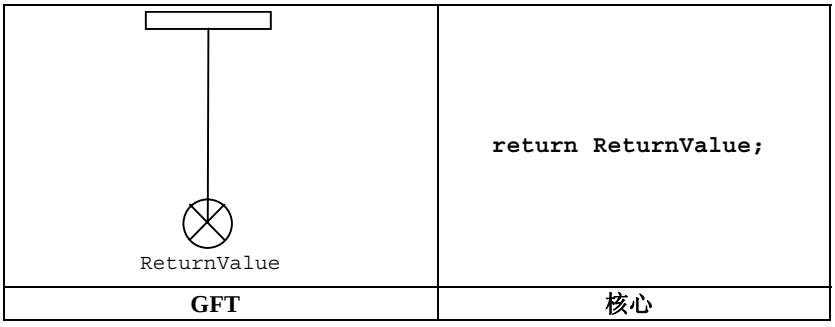


图 38/Z.142－带返回值的返回符号

11.6 缺省处理

GFT 为缺省的激活与取消激活提供图形表示（参见第 21 节/Z.140[1]）。

11.6.1 缺省引用

类型 **default** 的变量既可以在行为符号中进行声明，也可以在缺省符号中进行声明，作为激活语句的一部分。第 11.3.1 节和第 11.3.4 节阐述了如何在 GFT 中声明一个名为 MyDefaultType 的变量。

11.6.2 激活操作

缺省的激活将通过在缺省符号中放置 **activate** 语句来表示（参见图 39）。

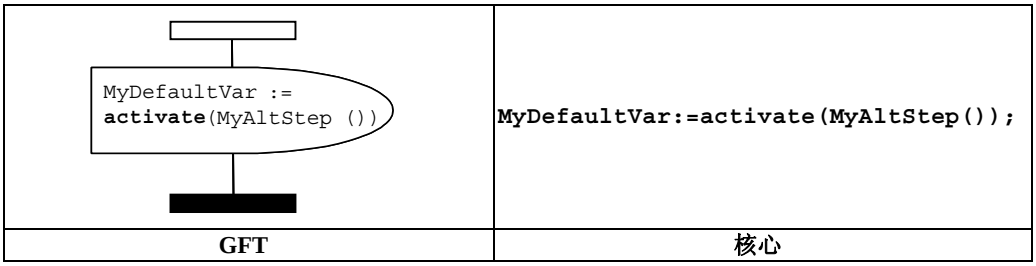


图 39/Z.142－缺省激活

11.6.3 取消激活操作

缺省的取消激活将通过在缺省符号中放置 **deactivate** 语句来表示（参见图 40）。如果未为 **deactivate** 语句提供任何操作数，那么所有的缺省都将被取消激活。

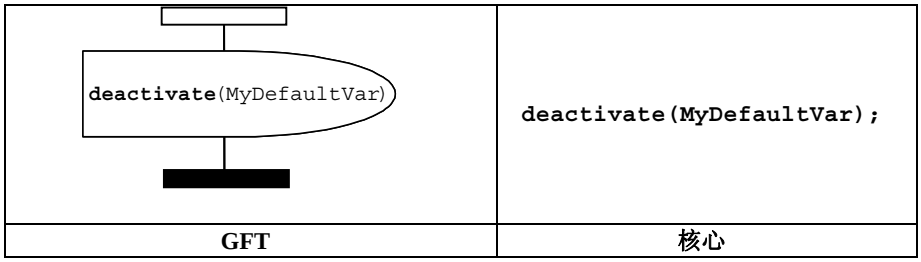


图 40/Z.142－取消缺省激活

11.7 配置操作

配置操作用于设立和控制测试部件。这些操作将只用在 GFT 测试用例、函数和可选步骤图中。

mtc、**self** 和 **system** 操作不存在任何图形表示；在其使用处，以文本形式来表示它们。

GFT 不为运行操作（是一个布尔表达式）提供任何图形表示。在其使用处，以文本形式来表示它。

11.7.1 创建操作

create 操作将在创建符号中予以表示，该符号与一个执行创建操作的测试部件实例相连（参见图 41）。创建符号包含创建语句。

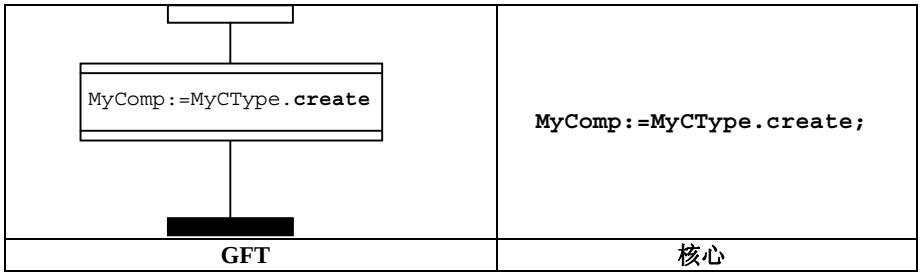


图 41/Z.142－创建操作

11.7.2 连接和映射操作

connect 和 **map** 操作将在一个行为框符号中予以表示，该符号与一个执行 **connect** 或 **map** 操作的测试部件相连（参见图 42）。行为框符号包含 **connect** 或 **map** 语句。

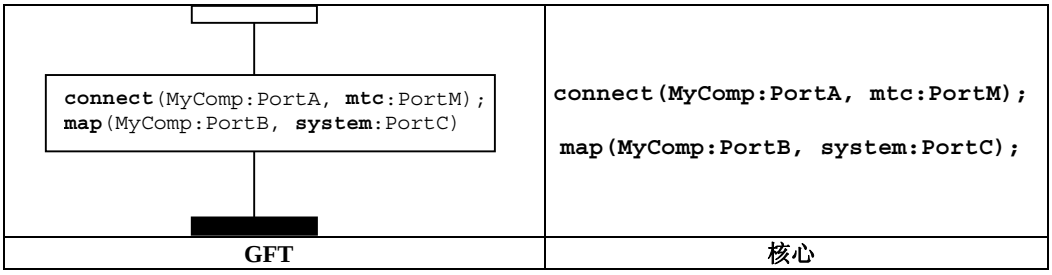


图 42/Z.142－连接和映射操作

11.7.3 断开连接和取消映射操作

disconnect 和 **unmap** 操作将在一个行为框符号中予以表示，该符号与一个执行 **disconnect** 或 **unmap** 操作的测试部件相连（参见图 43）。行为框符号包含 **disconnect** 或 **unmap** 语句。

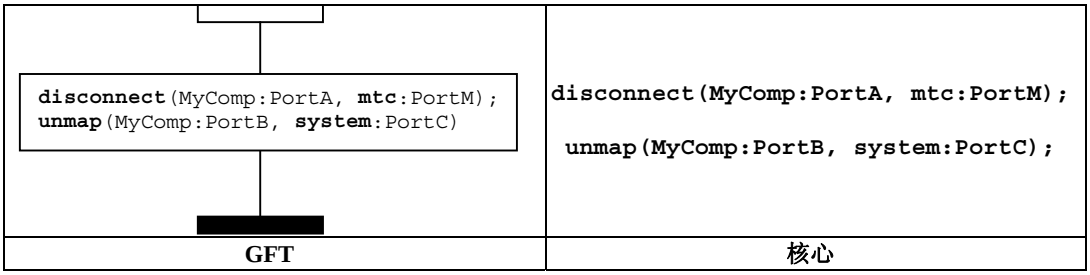


图 43/Z.142－断开连接和取消映射操作

11.7.4 启动测试部件操作

start 测试部件操作将在 **start** 符号中予以表示，该符号与执行 **start** 操作的测试部件实例相连（参见图 44）。启动符号包含 **start** 语句。

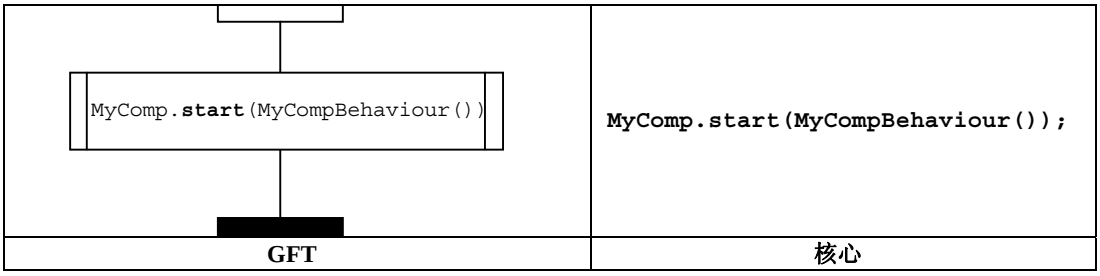


图 44/Z.142－启动操作

11.7.5 停止执行和停止测试部件操作

TTCN-3 有两种停止操作：通过使用*停止执行操作*，模块控制和测试部件可自行停止，或者通过使用*停止测试部件操作*，某个测试部件可停止其他测试部件。

stop 执行操作将在停止符号中予以表示，该符号与一个执行 **stop** 执行操作的测试部件实例相连（参见图 45）。它将只用作部件实例的最后事件，或者用作内嵌表达式符号中某个操作数的最后事件。

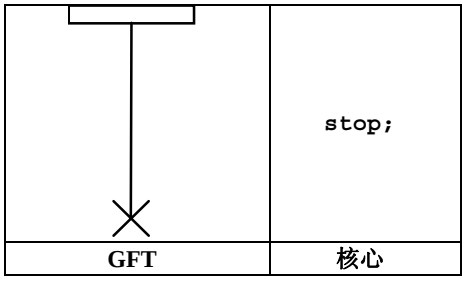


图 45/Z.142－停止执行操作

stop 测试部件操作将通过停止符号来表示，该符号与执行 **stop** 测试部件操作的测试部件实例相连（参见图 44）。它将具有一个相关的表达式，用于确定要停止的部件（参见图 46）。通过使用带有关键字 **all** 的停止部件操作，MTC 可一步停止所有 PTC（参见图 47 a））。通过停止 MTC，PTC 可以停止测试执行（参见图 47 b））。 如果部件自身停止（如 **self.stop**），或者停止测试执行（如 **mtc.stop**），那么 **stop** 测试部件操作将用作部件实例的最后事件，或者用作内嵌表达式符号中某个操作数的最后事件（参见图 47 c）和图 47 d））。

注 — 停止符号有一个相关的表达式。它并不总是能够静态地确定停止部件操作是否停止了执行停止操作或停止测试执行的实例。

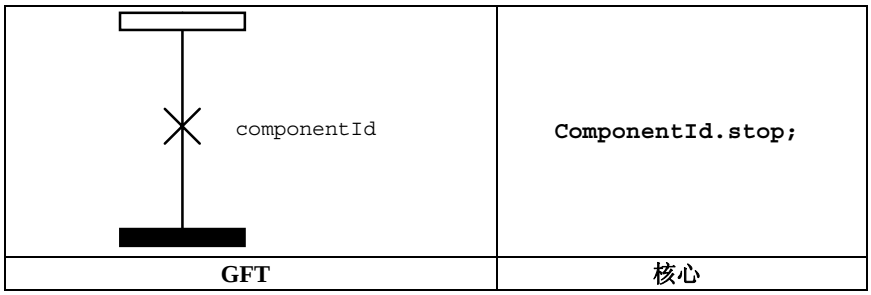


图 46/Z.142－停止测试部件操作

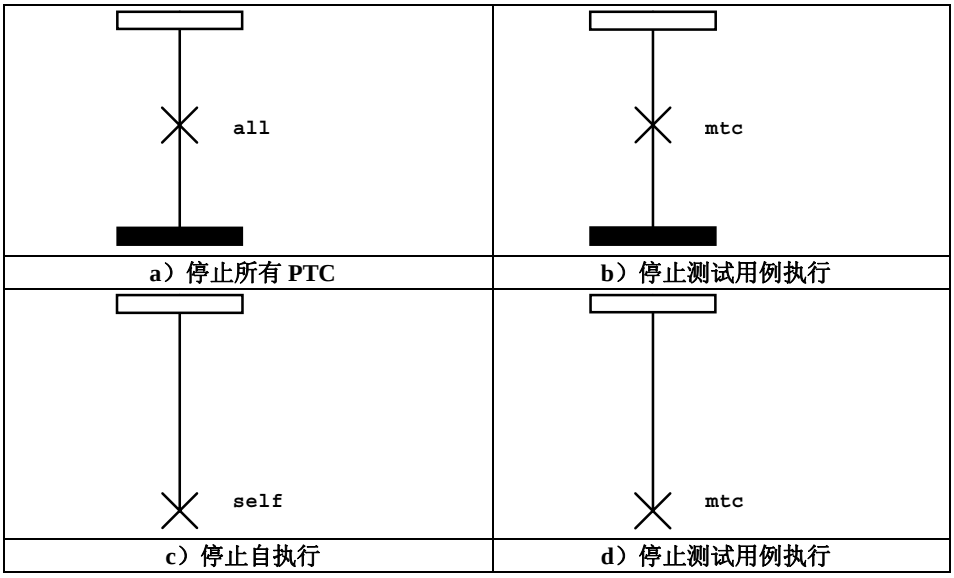


图 47/Z.142－停止测试部件操作的特殊用法

11.7.6 完成操作

done 操作将在条件符号中予以表示，该符号与执行 **done** 操作的测试部件实例相连（参见图 48）。条件符号包含 **done** 语句。

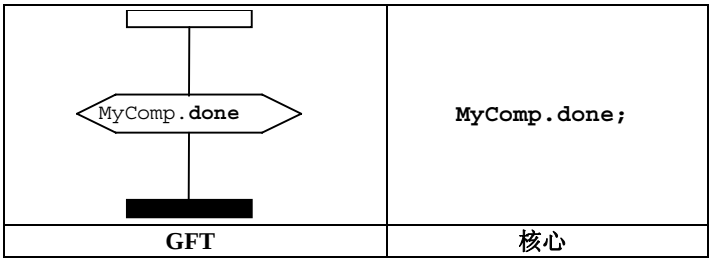


图 48/Z.142－完成操作

关键字 **any** 和 **all** 可用于 **running** 和 **done** 操作，但只能来自 MTC 实例。它们没有任何图形表示，但在其使用处，以文本形式进行表示。

11.8 通信操作

通信操作分为两组：

a） *发送操作*：测试部件发送一个消息（**send**操作）、调用一个程序（**call**操作）、答复一个已接受的调用（**reply**操作）或者引起一个异常（**raise**操作）。

b） *接收操作*：部件接收一个消息（**receive**操作）、接受一个程序调用（**getcall**操作）、接收一个有关之前调用程序的答复（**getreply** 操作）或者捕获一个异常（**catch**操作）。

11.8.1 发送操作的一般格式

所有的发送操作都使用一个来自测试部件实例的消息符号，它对信息传输的目的端口实例执行发送操作（参见图 49）。

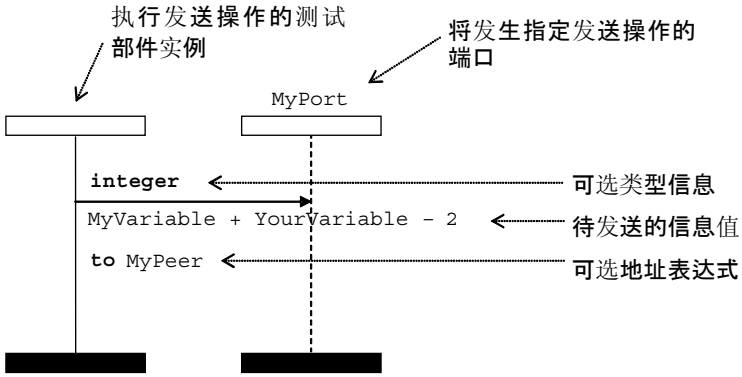


图 49/Z.142－发送操作的一般格式

发送操作包括*发送*部分，在阻塞的、基于程序的 **call** 操作情况下，还包括*响应*和*异常处理*部分。

发送部分：

- 指定将实施规定操作的端口；
- 定义待传输信息的可选类型和值；
- 提供一个可选的地址表达式，它在一对多连接情况下唯一地确定通信伙伴。

端口将通过一个端口实例来表示。**call**、**reply** 和 **raise** 操作的操作名称将在可选类型信息前面的消息符号顶部表示。**send** 操作是隐含的，也就是说，不得标明 **send** 关键字。待传送信息的值将置于消息符号之下。可选的地址表达式（用关键字 **to** 表示）将置于待传送信息的值之下。

call 操作的结构更为特别。详情请参见第 11.8.4.1 节。

11.8.2 接收操作的一般格式

所有的接收操作都使用一个来自端口实例的消息符号，它从测试部件实例接收信息（参见图 50）。

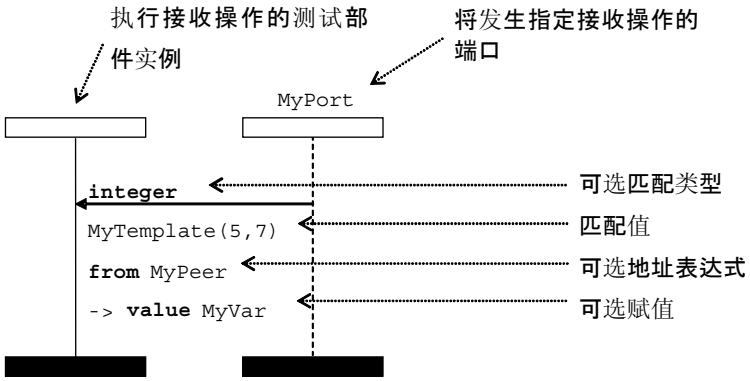


图 50/Z.142－带地址和赋值的接收操作的一般格式

接收操作包括接收部分和可选的赋值部分。

接收部分：

a） 指定将实施操作的端口；

b） 定义一个匹配的部分，它包括可选的类型信息和匹配的值，该值规定将与语句匹配的、可接受的输入；

c） 提供一个（可选的）地址表达式，它（在一对多连接情况下）唯一地确定通信伙伴。

端口将通过端口实例来表示。**getcall**、**getreply** 和 **catch** 操作的操作名称将在（可选）类型信息前面的消息符号顶部表示。**receive** 操作隐含提供，也就是说，不得标明关键字 **receive**。可接受输入的匹配值将置于消息符号之下。（可选的）地址表达式（用关键字 **from** 表示）将置于待传送信息的值之下。

（可选的）赋值部分（用“->”表示）将置于待传送信息的值之下，或者如果存在，那么置于地址表达式之下。它可以被分割为若干条线，例如，使值、参数和发送方赋值各位于单个的线上（参见图 51）。

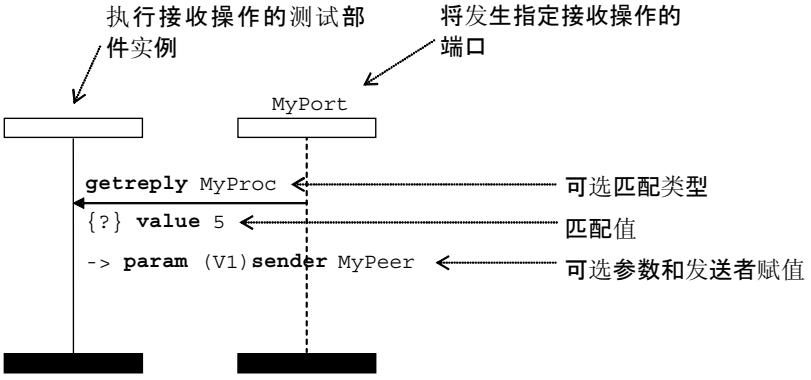


图 51/Z.142－带参数和发送者赋值的接收操作的一般格式

11.8.3 基于消息的通信

11.8.3.1 发送操作

发送操作将通过从测试部件到端口实例的一个向外消息符号来表示。可选的类型信息将置于消息箭头之上。（内嵌的）模板将置于消息箭头之下（参见图 52 和图 53）。

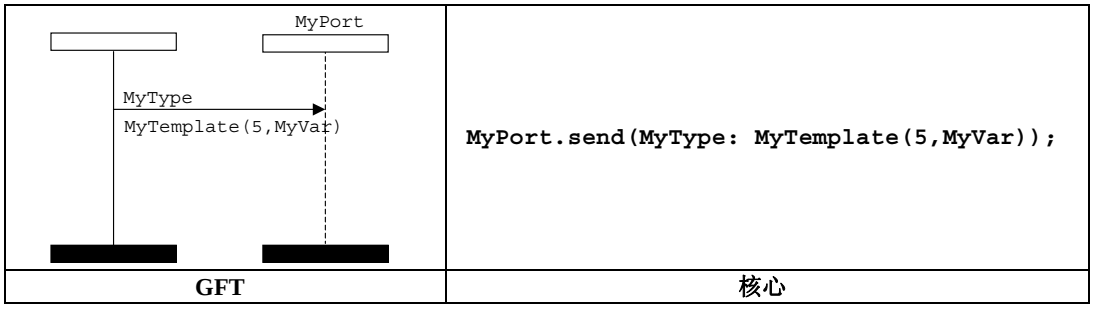


图 52/Z.142－带模板引用的发送操作

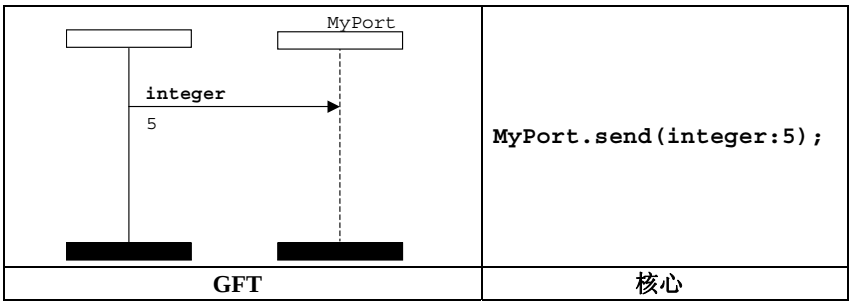


图 53/Z.142－带内嵌模板的发送操作

11.8.3.2 接收操作

接收操作将通过从端口实例到测试部件的一个向内消息箭头来表示。可选的类型信息将置于消息箭头之上。（内嵌的）模板将置于消息箭头之下（参见图 54 和图 55）。

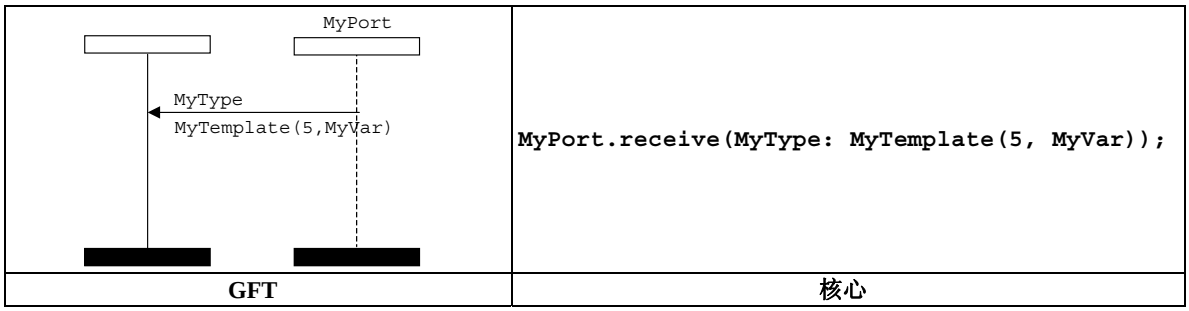


图 54/Z.142－带模板引用的接收操作

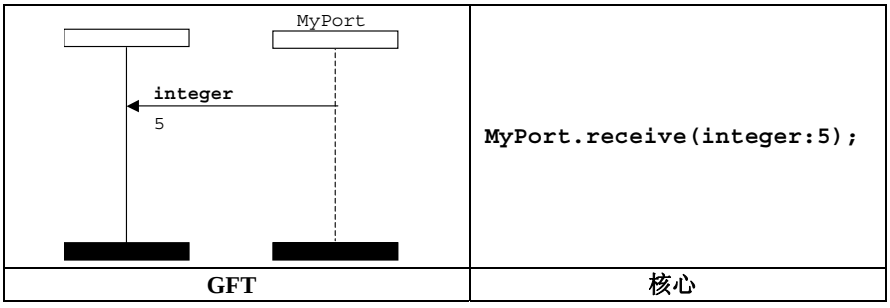


图 55/Z.142－带内嵌模板的接收操作

11.8.3.2.1 接收任何消息

接收任何消息的操作将通过从端口实例到测试部件、不附带任何更多信息的一个向内消息箭头来表示（参见图 56）。

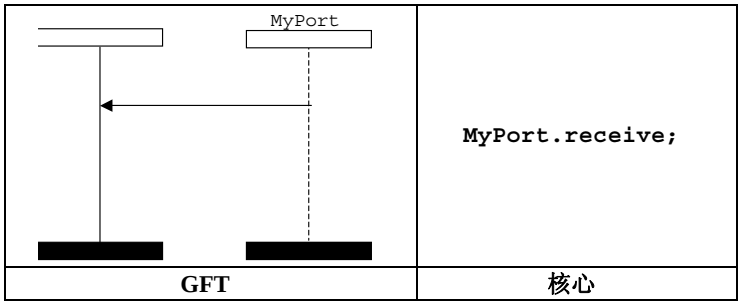


图 56/Z.142－接收任何消息

11.8.3.2.2 对任何端口的接收

对任何端口的接收操作将通过一个已发现的符号来表示，它表示任何指向测试部件的端口（参见图 57）。

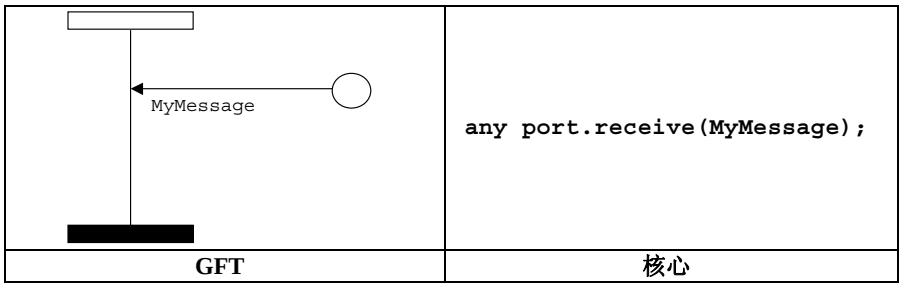


图 57/Z.142－对任何端口的接收

11.8.3.3 触发操作

触发操作将通过从端口实例到测试部件的一个向内消息箭头来表示，并且如果存在，那么通过类型消息之前、消息箭头之上的关键字 **trigger** 来表示。可选的类型信息置于消息箭头之上，紧跟关键字 **trigger** 之后。（内嵌的）模板置于消息箭头之下（参见图 58 和图 59）。

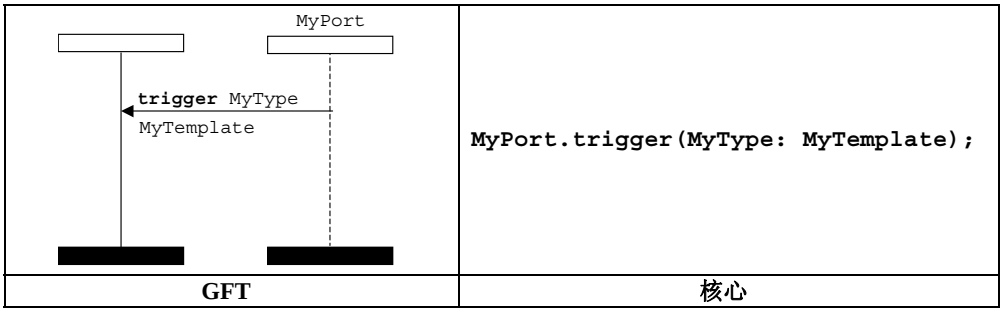


图 58/Z.142－带模板引用的触发操作

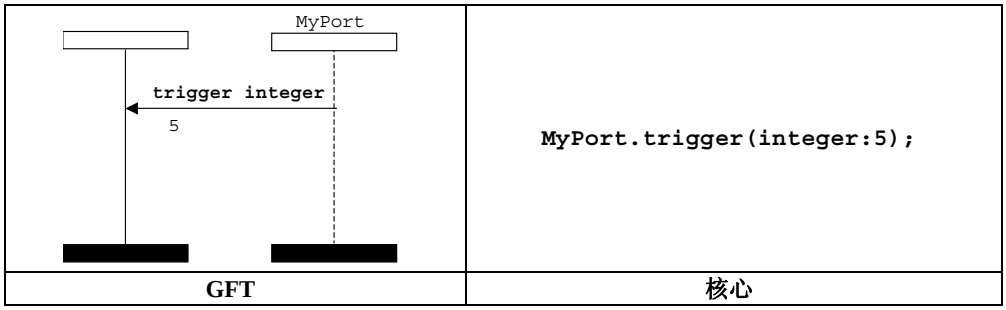


图 59/Z.142－带内嵌模板的触发操作

11.8.3.3.1 对任何消息的触发

对任何消息的触发操作将通过从端口实例到测试部件的一个向内消息箭头以及不附带任何更多信息的消息箭头之上的关键字 **trigger** 来表示（参见图 60）。

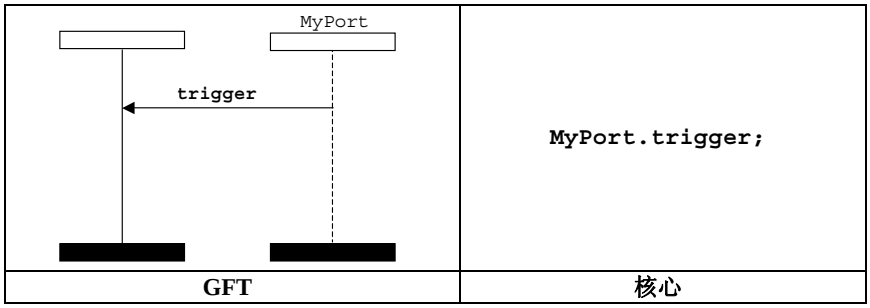


图 60/Z.142－对任何消息操作的触发

11.8.3.3.2 对任何端口的触发

对任何端口的触发操作将通过一个已发现的符号来表示，它表示任何指向测试部件的端口（参见图 61）。

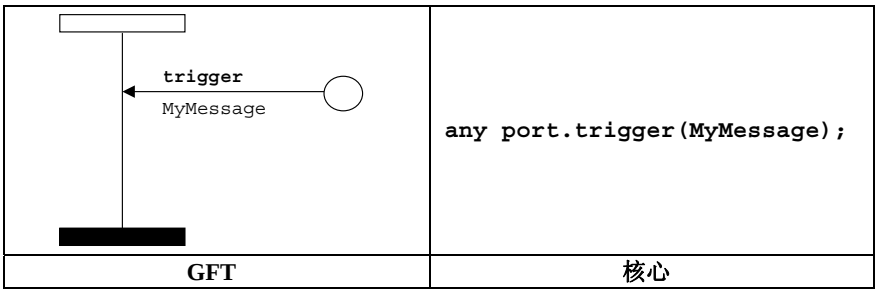


图 61/Z.142－对任何端口操作的触发

11.8.4 基于过程的通信

11.8.4.1 调用操作

11.8.4.1.1 调用阻塞过程

阻塞的 **call** 操作将通过从测试部件到端口实例（带有一个后继的暂停区域）的一个向外消息箭头来表示，并且如果存在，那么通过特征之前、消息箭头之上的关键字 **call** 来表示（参见图 62 和图 63）。

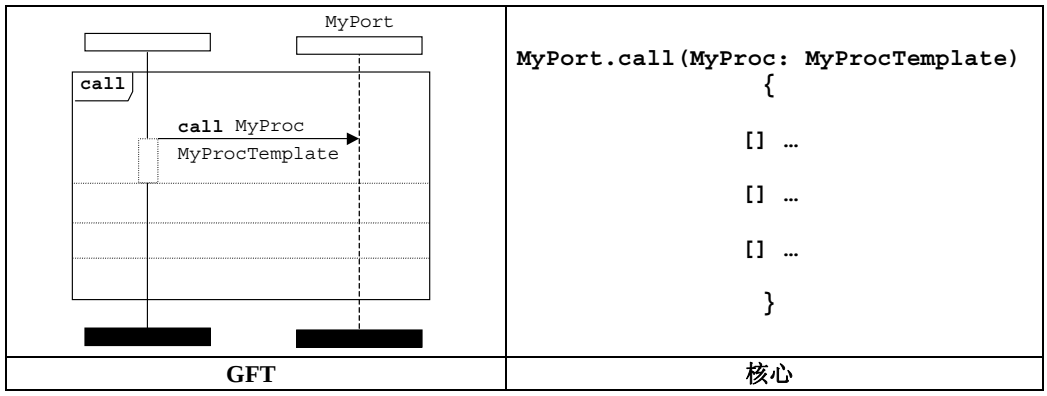


图 62/Z.142－带模板引用的阻塞调用操作

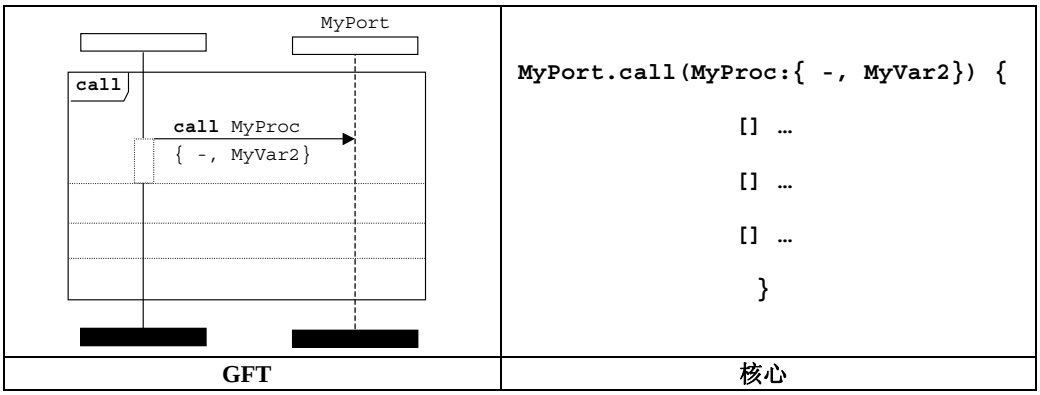


图 63/Z.142－带内嵌模板的阻塞调用操作

为了有利于规范对阻塞调用操作可能响应的可选项，引入了调用内嵌表达式。调用操作随后可以是 getreply、catch 和 timeout 中的任何一种。在调用内嵌表达式中，规定了对调用的响应，紧跟在调用操作之后的调用内嵌表达式用虚线进行分隔（参见图 64）。

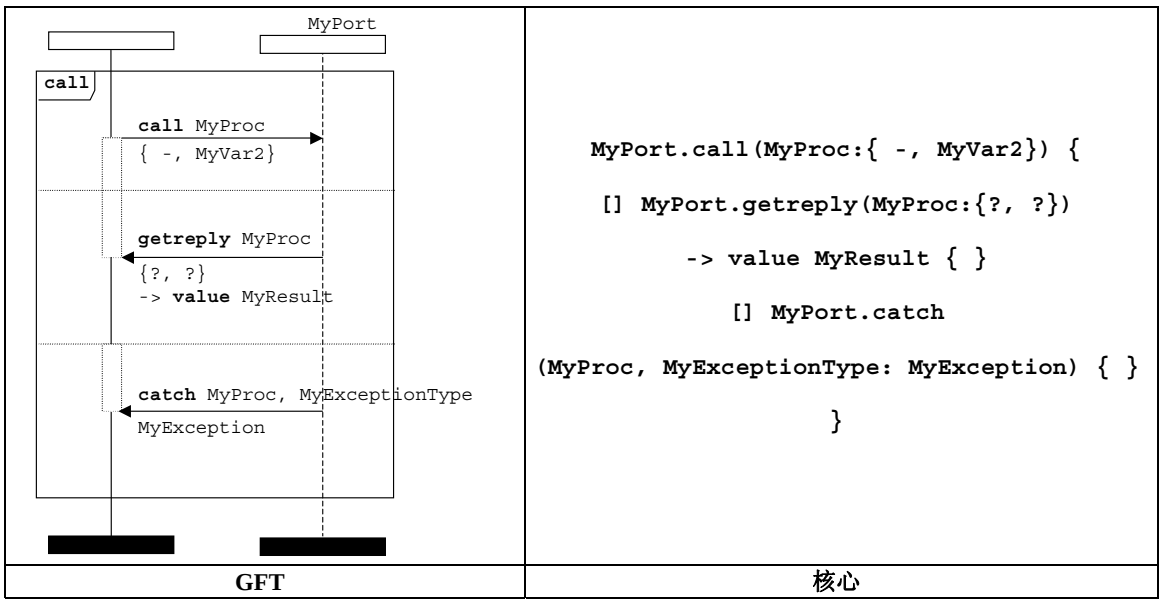


图 64/Z.142－后跟getreply和catch选项的阻塞调用操作

可选地，调用操作可以包括一个超时。对此，启动隐含定时器符号用于启动这一时间期限。超时隐含定时器符号用于表示超时异常（参见图 65）。

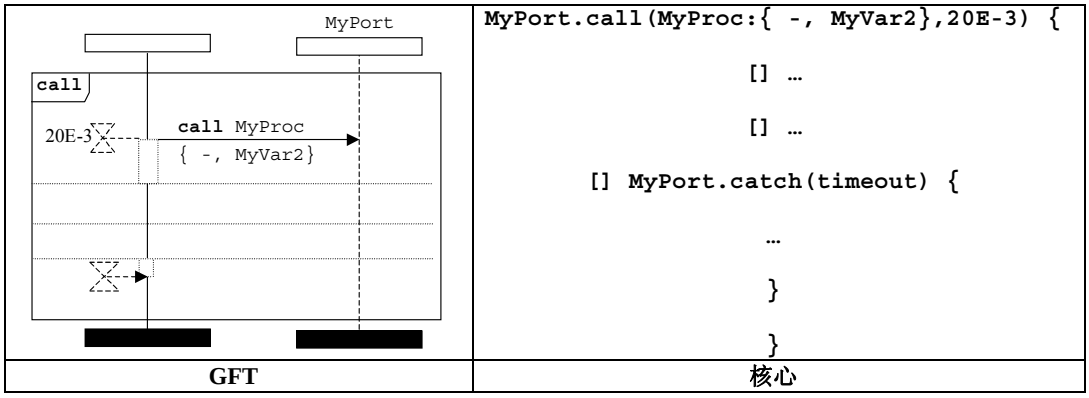


图 65/Z.142－后跟超时异常的阻塞调用操作

11.8.4.1.2 调用非阻塞过程

非阻塞调用操作将通过从测试部件到端口的一个向外消息符号以及特征之前、消息箭头之上的关键字 **call** 来表示。在该消息符号上，不得附有任何暂停区域符号。可选的特征在消息箭头之上进行表示。（内嵌的）模板置于消息箭头之下（参见图 66 和图 67）。

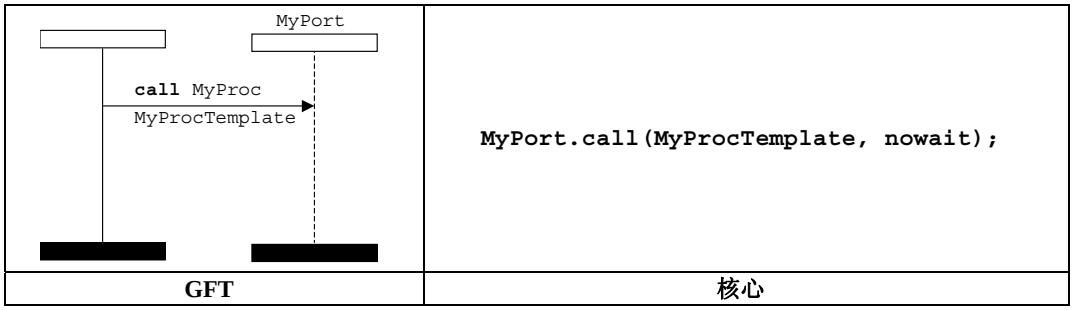


图 66/Z.142－带模板引用的非阻塞调用操作

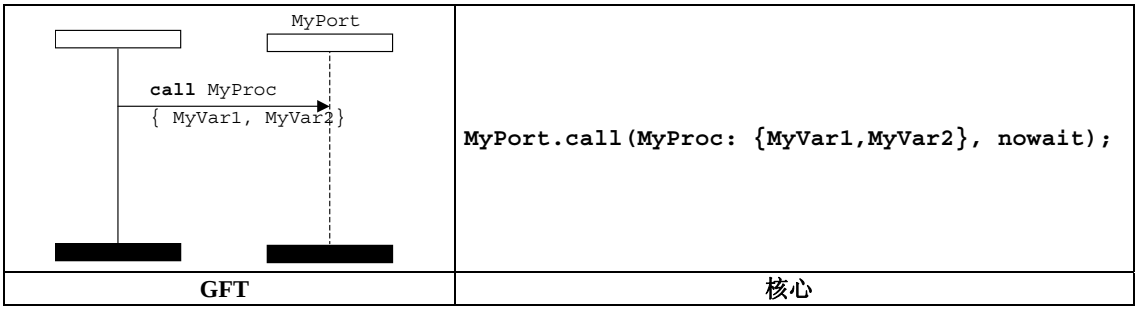


图 67/Z.142－带内嵌模板的非阻塞调用操作

11.8.4.2 Getcall操作

getcall 操作将通过从端口实例到测试部件的一个向内消息箭头以及特征之前、消息箭头之上的关键字 **getcall** 来表示。特征置于消息箭头之上，紧跟在关键字 **getcall** 之后。（内嵌的）模板置于消息箭头之下（参见图 68 和图 69）。

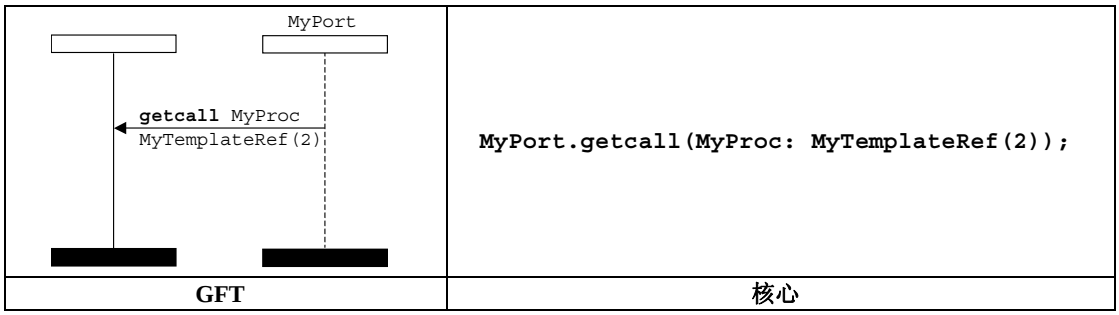


图 68/Z.142－带模板引用的Getcall操作

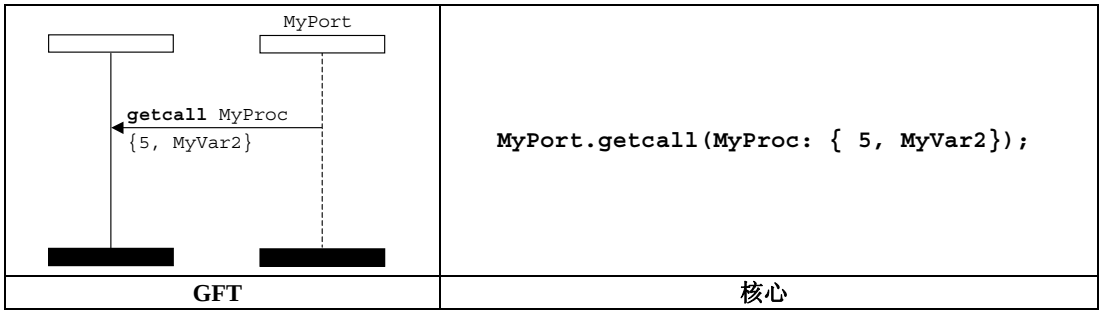


图 69/Z.142－带内嵌模板的Getcall操作

11.8.4.2.1 接受任何调用

接受任何调用操作经通过从端口实例到测试部件的一个向内消息箭头以及消息箭头之上的关键字 **getcall** 来表示。该消息符号上不得附有任何更多信息（参见图 70）。

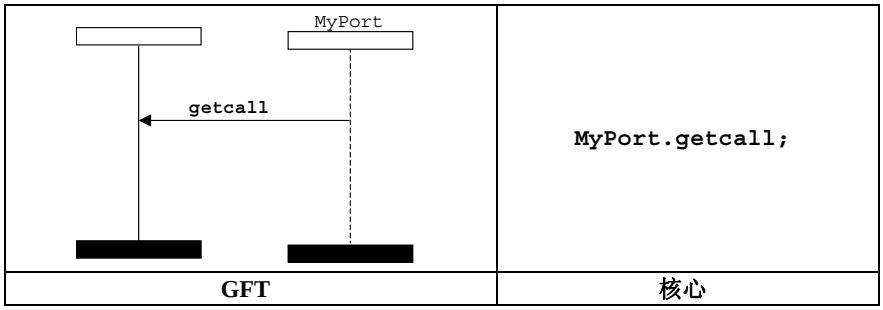


图 70/Z.142－对任何调用操作的Getcall

11.8.4.2.2 对任何端口的Getcall

对任何端口的 **getcall** 操作将通过一个已发现的符号来表示，它表示指向测试部件的任何端口，并且如果存在，通过后跟特征、消息箭头之上的关键字 **getcall** 来表示。如果存在（内嵌的）模板，那么将置于消息箭头之下（参见图 71）。

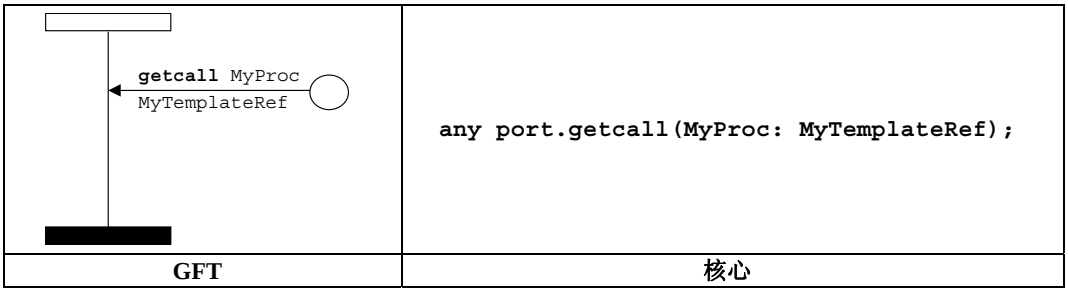


图 71/Z.142－对任何带模板引用的端口操作的Getcall

11.8.4.3 答复操作

回复操作将通过从测试部件到端口的一个向外消息符号以及特征之前、消息箭头之上的关键字 **reply** 来表示。特征将置于消息箭头之上，紧跟在关键字 **reply** 之后。（内嵌的）模板将置于消息箭头之下（参见图 72 和图 73）。

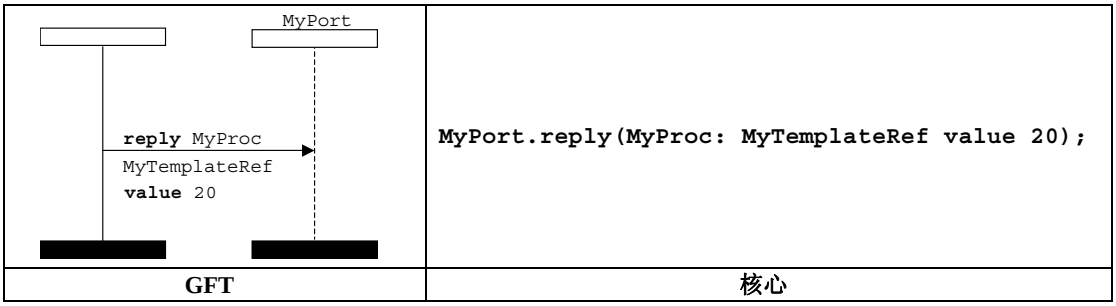


图 72/Z.142－带模板引用的答复操作

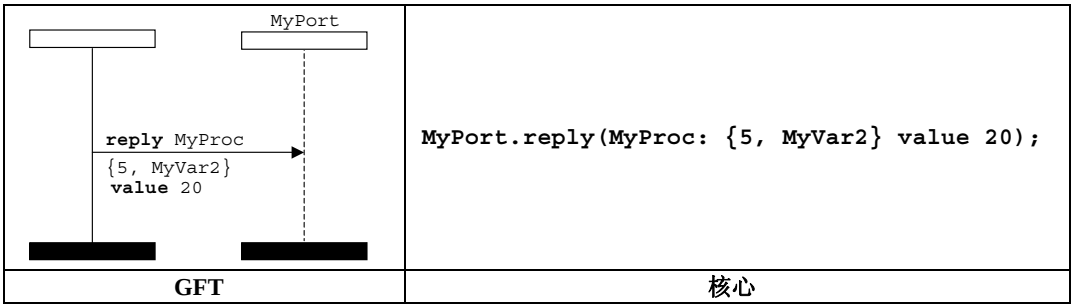


图 73/Z.142－带内嵌模板的答复操作

11.8.4.4 Getreply操作

getreply 操作将通过从端口实例到测试部件的一个向内消息箭头以及特征之前、消息箭头之上的关键字 **getreply** 来表示。在调用符号内，消息箭头头部将附于一个测试部件中之前暂停区域上（参见图 74 和图 75）。在调用符号外，消息箭头头部不得附于一个测试部件中之前暂停区域上（参见图 76 和图 77）。

特征将置于消息箭头之上，紧跟在关键字 **getreply** 之后。（内嵌的）模板将置于消息箭头之下。

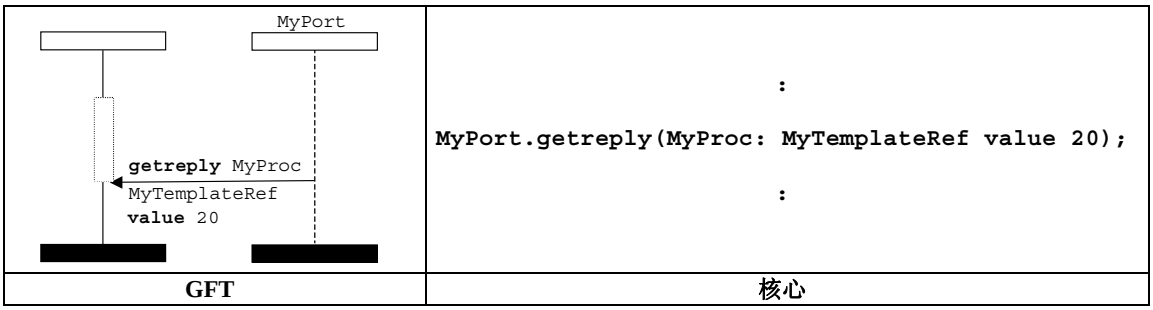


图 74/Z.142－带模板引用的Getreply操作（在一个调用符号内）

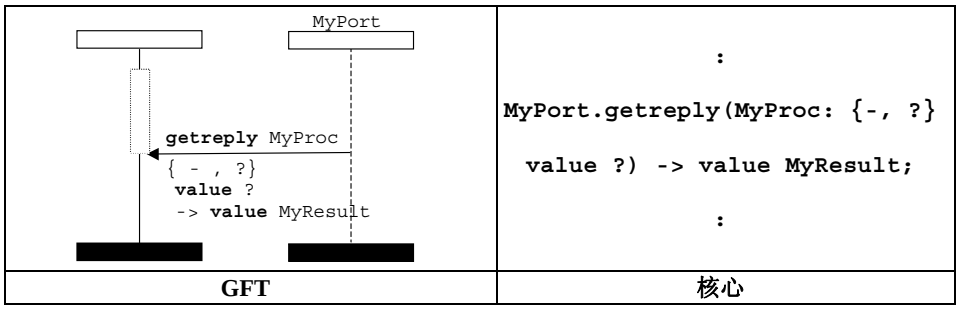


图 75/Z.142－带内嵌模板的Getreply操作（在一个调用符号内）

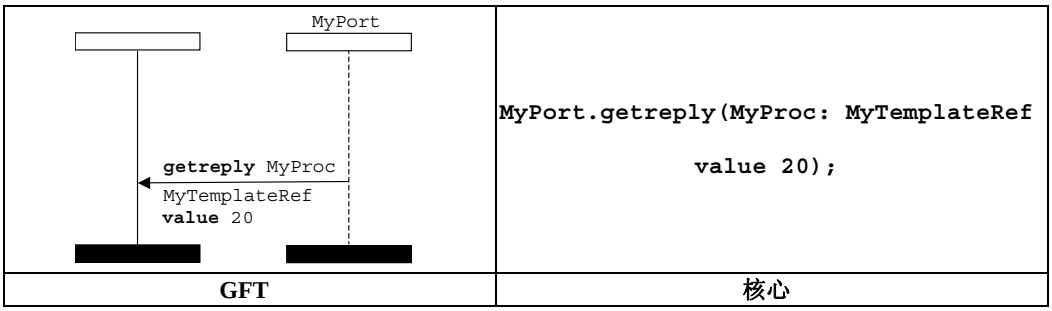


图 76/Z.142－带模板引用的Getreply操作（在一个调用符号外）

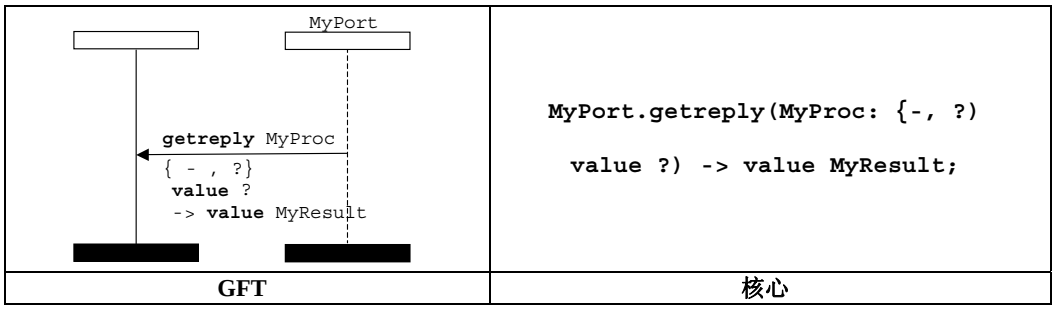


图 77/Z.142－带内嵌模板的Getreply操作（在一个调用符号外）

11.8.4.4.1 获得来自任何调用的任何答复

获得来自任何调用之任何回复的操作将通过从端口实例到测试部件的一个向内消息箭头以及消息之上的关键字 **getreply** 来表示。关键字 **getreply** 之后不得跟随任何特征。在调用符号内，消息箭头头部将附于一个测试部件中之前暂停区域上（参见图 78）。在调用符号外，消息箭头头部不得附于一个测试部件中之前暂停区域上（参见图 79）。

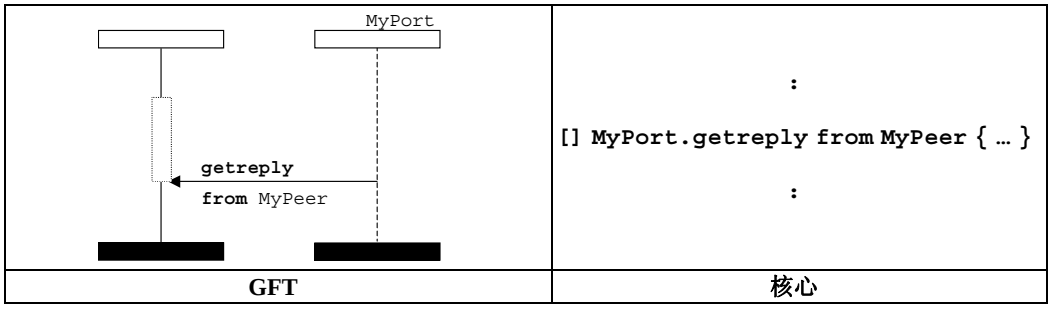


图 78/Z.142－从任何调用获得答复（在一个调用符号内）

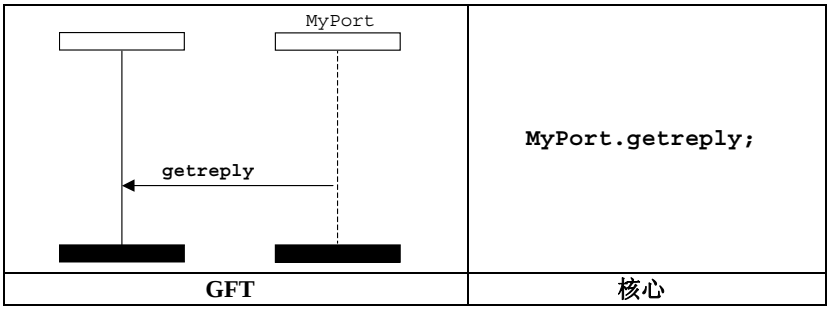


图 79/Z.142－从任何调用获得答复（在一个调用符号外）

11.8.4.4.2 获得来自任何端口的一个答复

获得来自任何端口之一个答复的操作将通过一个已发现的符号来表示，它表示任何指向测试部件的端口。关键字 **getreply** 将置于消息箭头之上，如果存在，随后紧跟特征。在调用符号内，消息箭头头部将附于一个测试部件中之前暂停区域上（参见图 80）。在调用符号外，消息箭头头部不得附于一个测试部件中之前暂停区域上（参见图 81）。

如果存在，那么特征将置于消息箭头之上，紧跟在关键字 **getreply** 之后。可选的（内嵌）模板置于消息箭头之下。

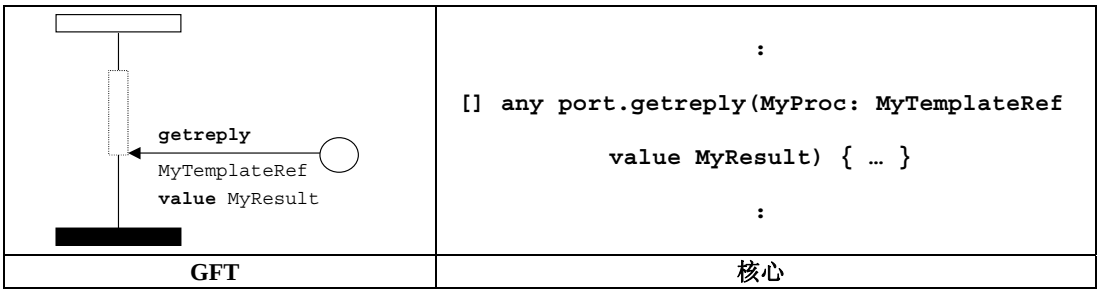


图 80/Z.142－在任何端口上获得一个答复（在一个调用符号内）

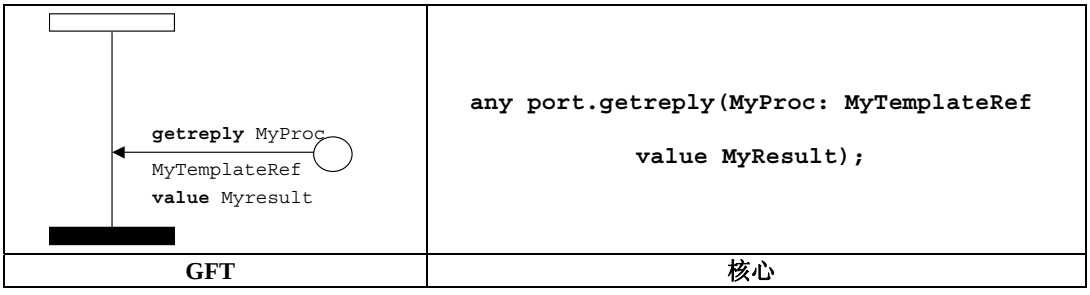


图 81/Z.142－在任何端口上获得一个答复（在一个调用符号外）

11.8.4.5 引起操作

引起操作将通过从测试部件到端口实例的一个向外消息符号来表示。关键字 **raise** 将置于特征和异常类型之前、消息箭头之上，它们用逗号分隔。（内嵌的）模板将置于消息箭头之下（参见图 82 和图 83）。

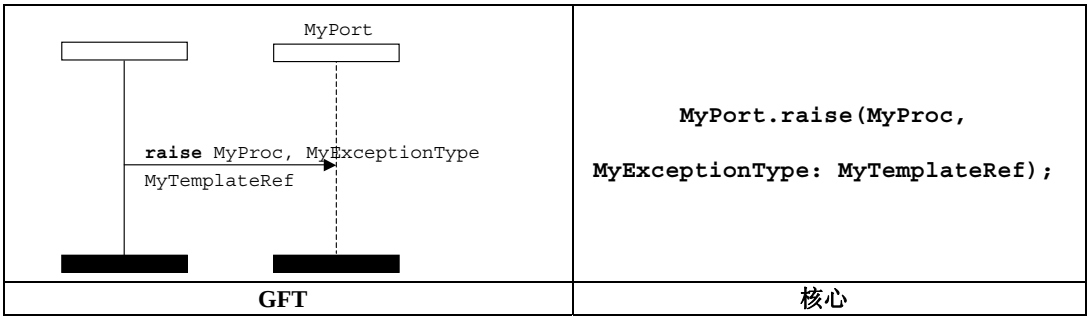


图 82/Z.142－带模板引用的引起操作

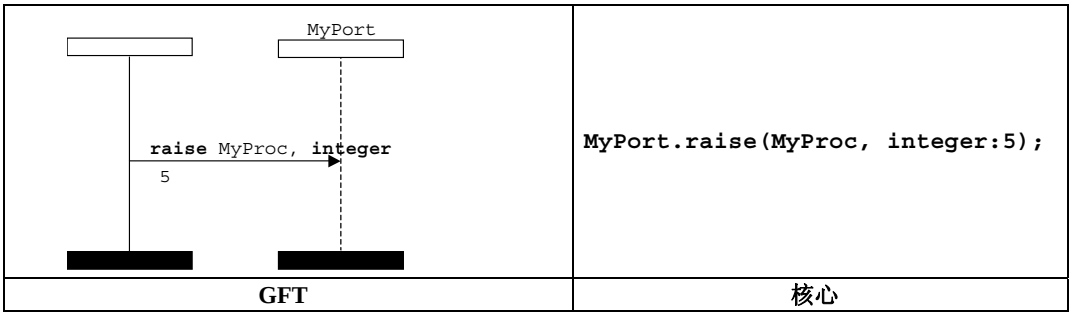


图 83/Z.142－带内嵌模板的引起操作

11.8.4.6 捕获操作

捕获操作将通过从端口实例到测试部件的一个向内消息箭头以及特征和异常类型之前（如果存在）、消息箭头之上的关键字 **catch** 来表示。在调用符号内，消息箭头头部将附于一个测试部件中之前暂停区域上（参见图 84 和图 85）。在调用符号外，消息箭头头部不得附于一个测试部件中之前暂停区域上（参见图 86 和图 87）。

特征和可选的异常类型信息将置于消息箭头之上，紧跟在关键字 **catch** 之后，如果存在异常类型，那么用逗号将它们隔开。（内嵌的）模板将置于消息箭头之下。

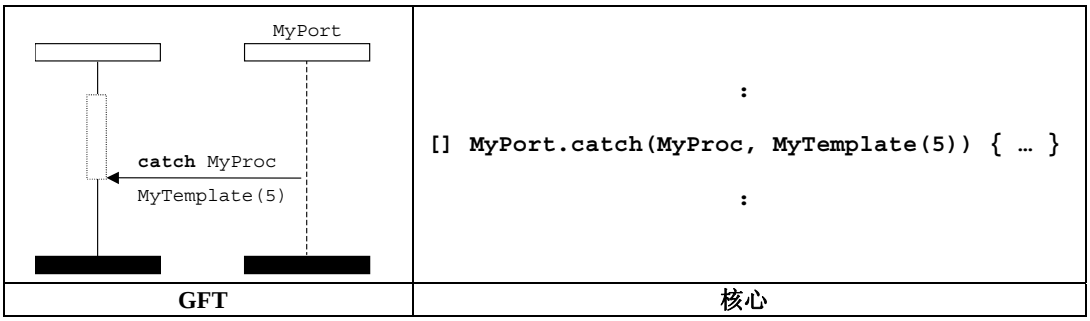
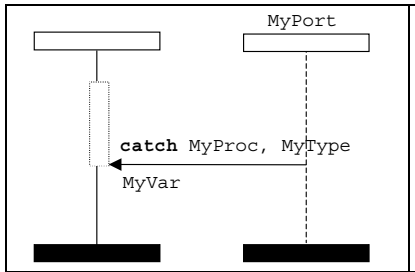


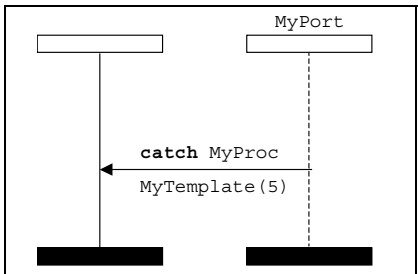
图 84/Z.142－带模板引用的捕获操作（在一个调用符号内）




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

GFT	核心

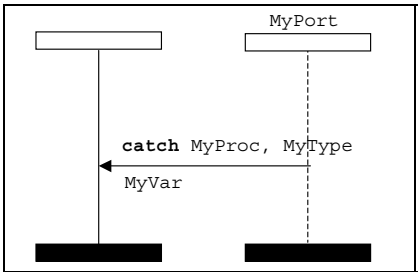
图 85/Z.142－带内嵌模板的捕获操作（在一个调用符号内）




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

GFT	核心

图 86/Z.142－带模板引用的捕获操作（在一个调用符号外）



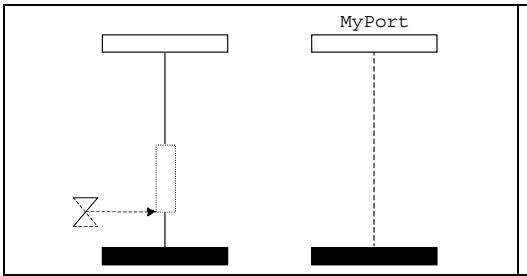

```
MyPort.catch(MyProc, MyType: MyVar);
```

GFT	核心

图 87/Z.142－带内嵌模板的捕获操作（在一个调用符号外）

11.8.4.6.1 超时异常

超时异常操作将通过一个超时符号来表示，它带有一个与测试部件相连的箭头（参见图 88）。该超时符号上不得附有任何更多信息。它将只在调用符号中使用。消息箭头头部将附于一个测试部件中之前暂停区域上。




```
:
[] MyPort.catch(timeout) { … }
:
```

GFT	核心

图 88/Z.142－超时异常（在一个调用符号内）

11.8.4.6.2 捕获任何异常

捕获任何异常的操作将通过从端口实例到测试部件的一个向内消息箭头以及消息箭头之上的关键字 **catch** 来表示。在调用符号内，消息箭头头部将附于一个测试部件中之前暂停区域上（参见图 89）。在调用符号外，消息箭头头部不得附于一个测试部件中之前暂停区域上（参见图 90）。捕获任何异常不得拥有任何模板和任何异常类型。

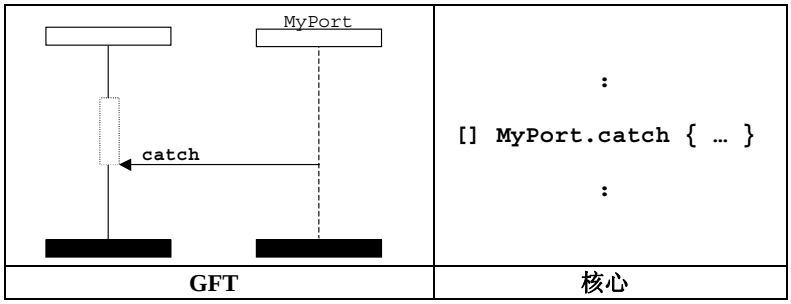


图 89/Z.142－捕获任何异常（在一个调用符号内）

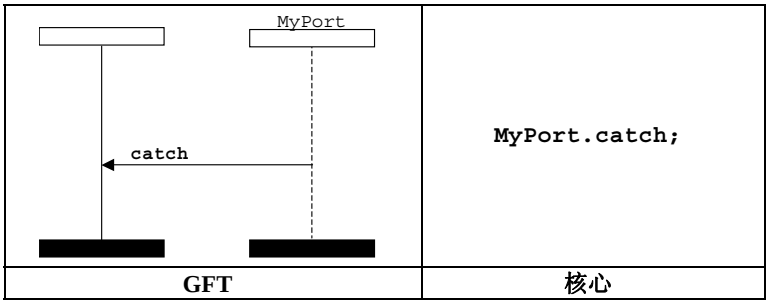


图 90/Z.142－捕获任何异常（在一个调用符号外）

11.8.4.6.3 对任何端口的捕获

对任何端口的捕获操作将通过一个已发现的符号（表示指向测试部件的任何端口）以及消息箭头之上的关键字 **catch** 来表示。在调用符号内，消息箭头头部将附于一个测试部件中之前暂停区域上（参见图 91）。在调用符号外，消息箭头头部不得附于一个测试部件中之前暂停区域上（参见图 92）。如果存在，那么模板置于消息箭头之下。

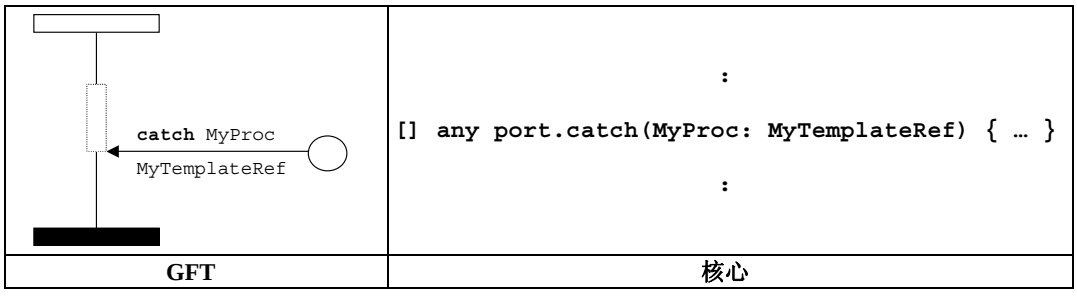


图 91/Z.142－对任何端口的捕获（在一个调用符号内）

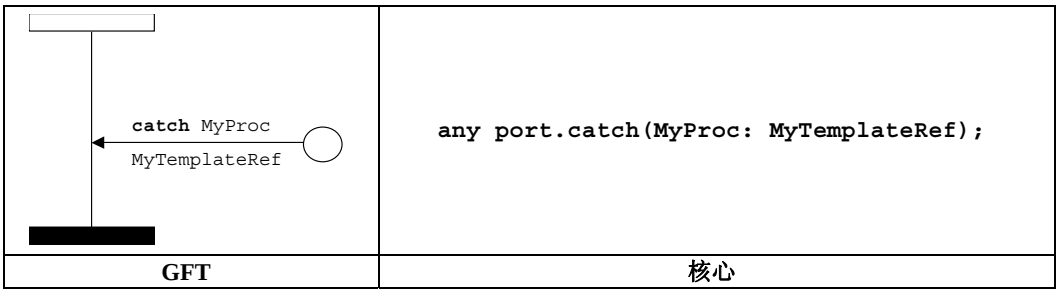


图 92/Z.142－对任何端口的捕获（在一个调用符号外）

11.8.5 检查操作

检查操作将通过从端口实例到测试部件的一个向内消息箭头来表示。关键字 **check** 将置于消息箭头之上。与 **receive**（参见图 93）、**getcall**、**getreply**（参见图 94 和图 95）和 **catch** 相关的信息附件紧跟在关键字 **check** 之后，并遵循用于表示这些操作的规则。

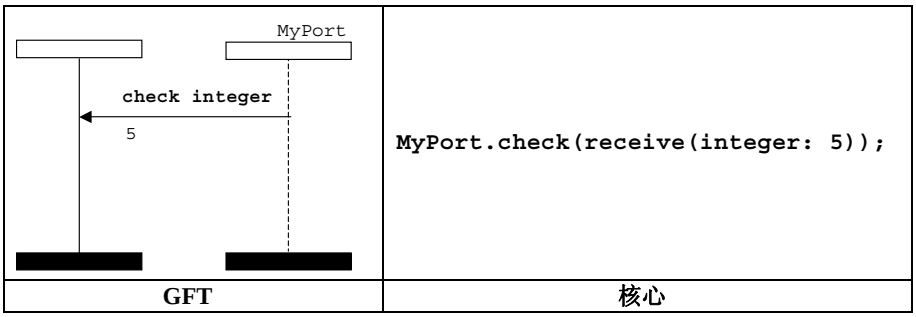


图 93/Z.142－检查带内嵌模板的接收

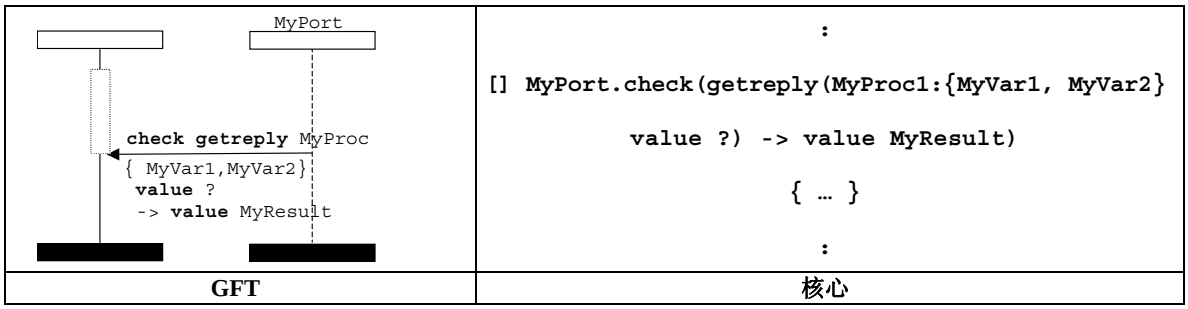


图 94/Z.142－检查getreply（在一个调用符号内）

图 95/Z.142－检查getreply（在一个调用符号外）

11.8.5.1 检查任何操作

检查任何操作将通过从端口实例到测试部件的一个向内消息箭头以及消息箭头之上的关键字 **check** 来表示（参见图 96）。它不得附有任何接收操作关键字、类型和模板。可选地，可以附有地址信息和保存发送方。

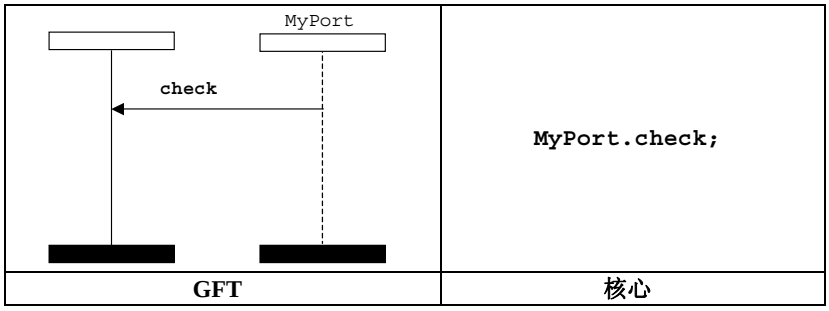


图 96/Z.142－检查任何操作

11.8.5.2 对任何端口的检查

对任何端口的检查操作将通过一个已发现的符号（表示指向测试部件的任何端口）以及消息箭头之上的关键字 **check** 来表示(参见图 97)。与 **receive**、**getcall**、**getreply** 和 **catch** 相关的信息附件紧跟在关键字 **check** 之后，并遵循用于表示这些操作的规则。

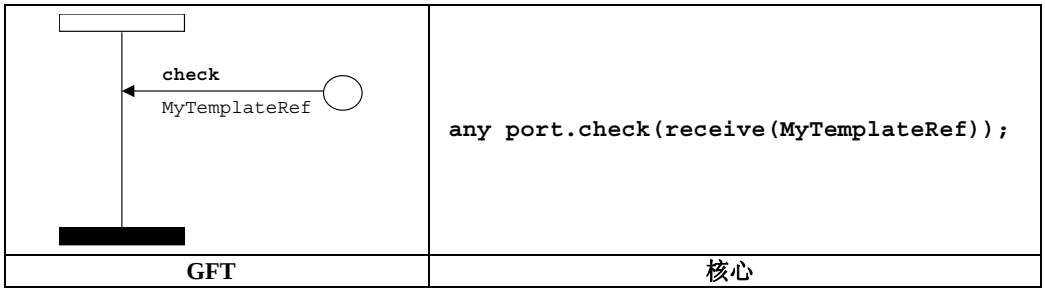


图 97/Z.142－检查任何端口上的接收

11.8.6 控制通信端口

11.8.6.1 清除端口操作

清除端口操作将通过带关键字 **clear** 的条件符号来表示。它附于测试部件实例（执行清除端口操作）和被清除的端口上（参见图 98）。

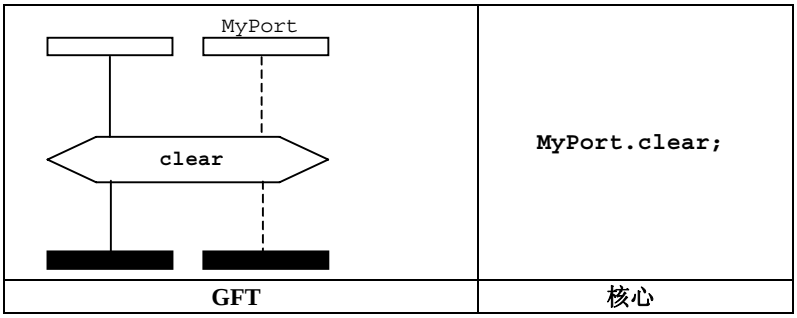


图 98/Z.142－清除端口操作

11.8.6.2 启动端口操作

启动端口操作将通过带关键字 **start** 的条件符号来表示。它附于测试部件实例（执行启动端口操作）和被启动的端口上（参见图 99）。

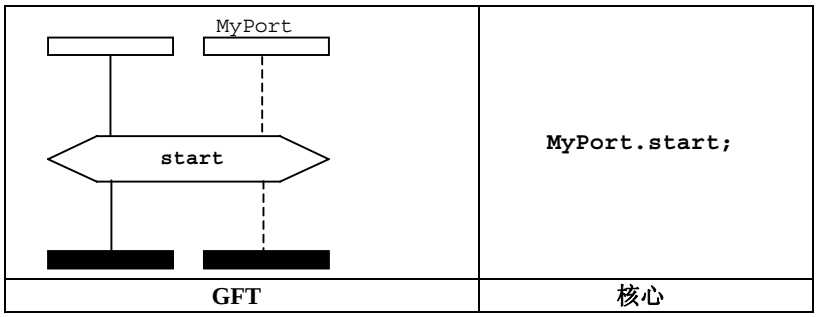


图 99/Z.142－启动端口操作

11.8.6.3 停止端口操作

停止端口操作将通过带关键字 **stop** 的条件符号来表示。它附于测试部件实例（执行停止端口操作）和被停止的端口上（参见图 100）。

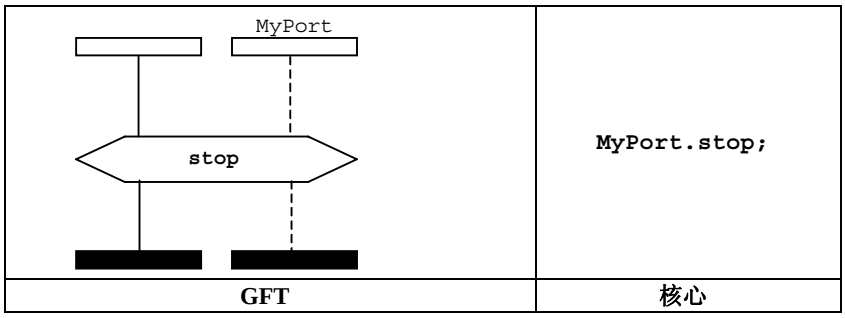


图 100/Z.142－结束端口操作

11.8.6.4 与端口一起使用any和all关键字

在第 11.8 节的各子节中对端口关键字 **any** 的 GFT 表示以及 **receive**、**trigger**、**getcall**、**getreply**、**catch** 和 **check** 操作进行了解释。

端口关键字 **all** 和清除、启动、停止操作一起，通过将包含 **clear**、**start** 或 **stop** 操作的条件符号附于有关测试用例、函数或可选步骤的 GFT 图中所表示的所有端口实例上来表示。

11.9 定时器操作

在 GFT 中，存在两种不同的定时器符号：一种针对的是确定的定时器，另一种针对的是调用定时器（参见图 101）。它们在外观上不同，实线定时器符号用于确定的定时器，虚线定时器符号用于调用定时器。一个确定的定时器将拥有自己的名称，附于其符号上，而一个调用定时器没有名称。本节描述确定的定时器。在第 11.8.4 节中对调用定时器进行描述。

图 101/Z.142－确定的定时器和调用定时器

GFT 不为 **running** 定时器操作（是一个布尔表达式）提供任何图形表示。在其使用处，以文本形式对它进行表示。

11.9.1 启动定时器操作

对于启动定时器操作，启动定时器符号将与部件实例相连。定时器名称和可选的期限值（在括号内）可以是相关的（参见图 102）。

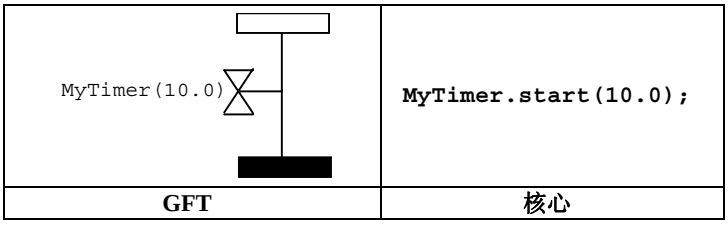


图 102/Z.142－启动定时器操作

11.9.2 停止定时器操作

对停止定时器操作，停止定时器符号将与部件实例相连。可选的定时器名称可以是相关的（参见图 103）。

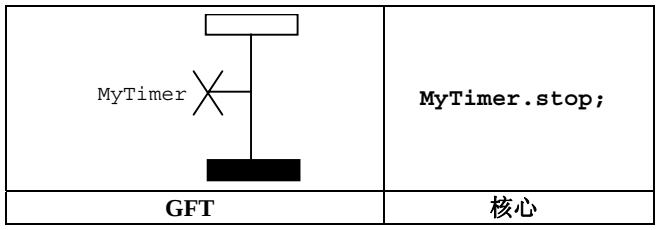


图 103/Z.142－结束定时器操作

启动定时器和停止定时器操作符号可以与一根垂直线相连。这样，定时器标识符只需在启动定时器信号附近进行说明（参见图 104）。

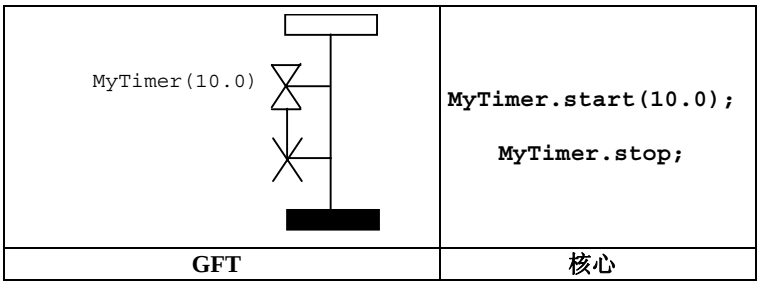


图 104/Z.142－相连的启动和停止定时器符号

11.9.3 超时操作

对超时操作，超时符号将与部件实例相连。可选的定时器名称可以是相关的（参见图 105）。

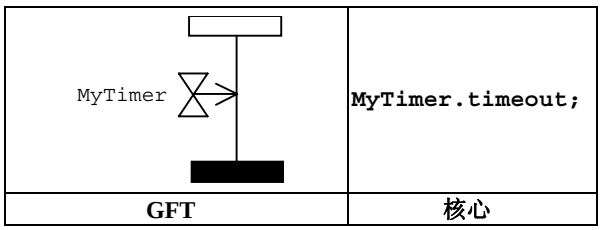


图 105/Z.142－超时操作

启动定时器和超时定时器操作的符号可与一根垂直线相连。这样，定时器标识符只需在启动定时器信号附近进行说明（参见图 106）。

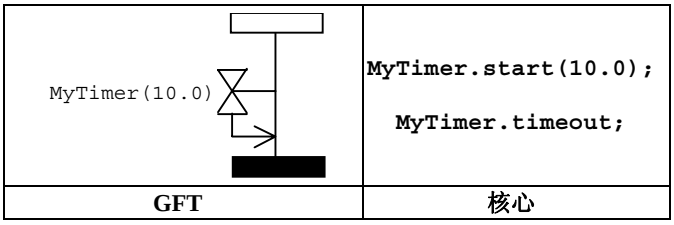


图 106/Z.142－相连的启动和超时定时器符号

11.9.4 读取定时器操作

读取定时器操作将置于动作框中（参见图 107）。

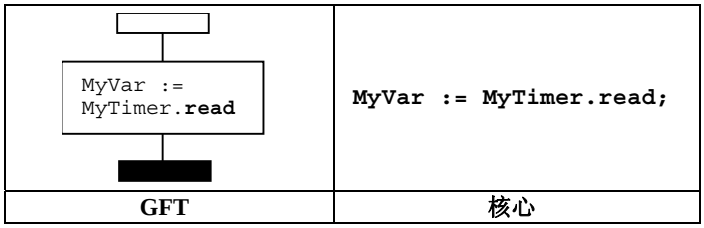


图 107/Z.142－读取定时器操作

11.9.5 与定时器一起使用any和all关键字

停止定时器操作可用于 **all** 定时器（参见图 108）。

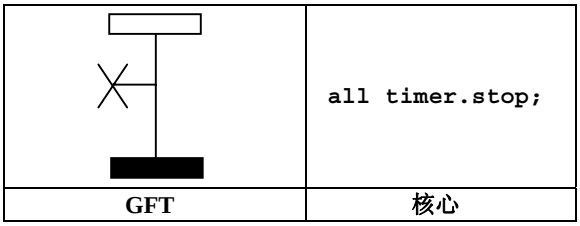


图 108/Z.142－停止所有定时器

暂停操作可适用于 **all** 定时器（参见图 109）。

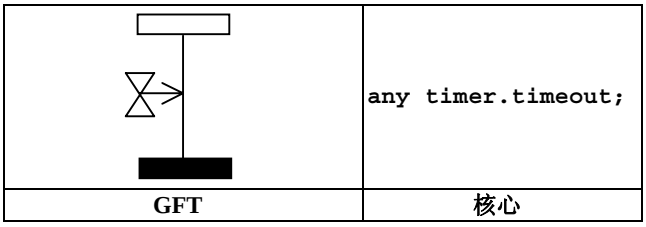


图 109/Z.142－来自任何定时器的超时

11.10 测试判定操作

判定设置操作 **setverdict** 用带条件符号的 GFT 进行表示，**pass**、**fail**、**inconc** 或 **none** 的值在其中进行表示（见图 110）。

注 — 设置一个新判定的规则遵循通常的、有关测试判定的TTCN-3覆盖规则。

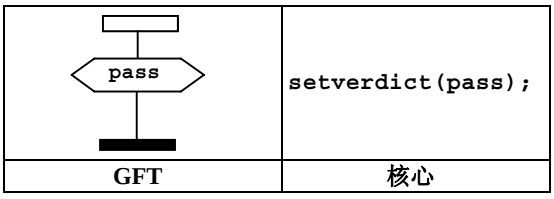


图 110/Z.142－设置局部判定

GFT 不为 **getverdict** 操作（是一个表达式）提供任何图形表示。文本上，它在使用处进行表示。

11.11 外部行为

外部行为在行为框符号中进行表示（见图 111）。外部行为的语法置于该符号内。

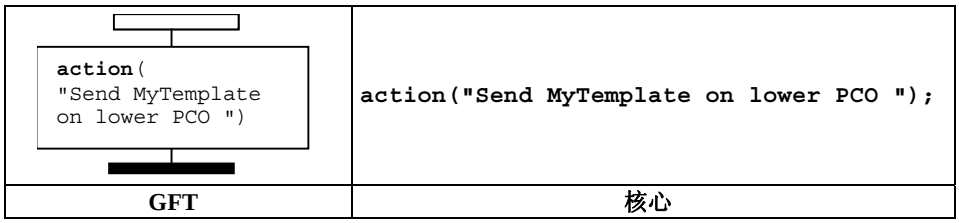


图 111/Z.142－外部行为

11.12 指定属性

为模块控制部分、测试用例、函数和可选步骤定义的属性在文本符号中表示。**with** 语句的语法置于该符号内。在图 112 中给出了一个例子。

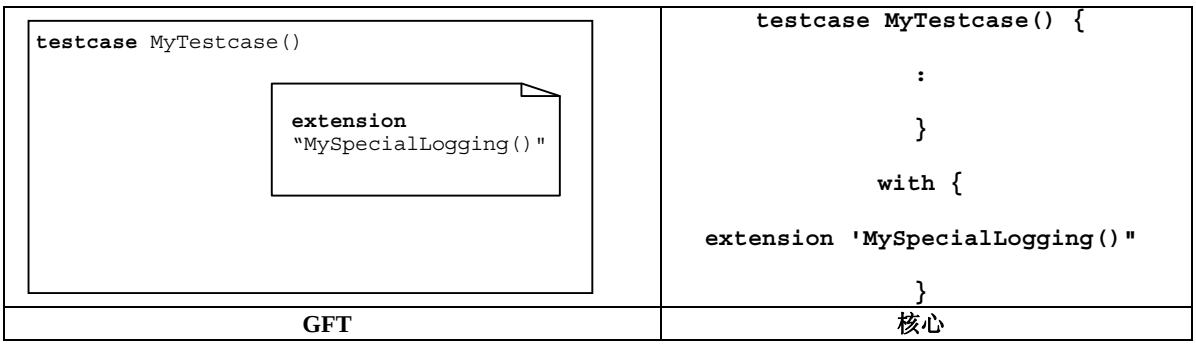


图 112/Z.142－指定属性

附 件 A

GFT BNF

A.1 GFT元语言

基于 MSC 的图形语法（ITU-T Z.120 建议书[3]）来定义 GFT 的图形语法。图形语法定义使用一种元语言，它在 1.4/Z.120[3]中进行解释。

“图形语法的精度不足以描述图形，因此没有任何图形变异。允许图形终端符号的实际形态有小的变异。这些包括，例如，填充符号的阴影、箭头的阴影和图形元素的相对大小。一旦需要，将用非正式的构件外形描述来对图形语法进行补充。元语法由带特殊元构件的类 BNF 记法组成：***contains***（包含）、***is followed by***（后跟）、***is associated with***（关联于）、***is attached to***（附于）、***above***（上面） 和 ***set***（集合）。这些构件的作用像普通的 BNF 产生规则，但它们额外地隐含某些变量间的逻辑或几何关系。***is attached to*** 构件的作用与下面所述有所不同。除了 ***above***，所有构件的左侧都必须是一个符号。一个符号是一个非终端，它在每个产生序列中都准确地产生一个图形终端。我们将认为一个符号附于（***is attached to***）其他区域上或者关联于（***is associated with***）一个也是符号的文本串。解释是非正式的，并且元语言无法准确描述几何从属关系。”

更多细节请参见 ITU-T Z.120 建议书[3]。

A.2 语法描述惯例

表 A.1 定义了用于规范 GFT 语法的元记法。它等同于 TTCN-3 所用的元记法，但区别于 MSC 所用的元记法。为了便于阅读，额外提供了对应的 MSC 元记法，并指出了差别。

表 A.1/Z.142－语法元记法

含　义	TTCN-3	GFT	MSC	差　别
定义为	::=	::=	::=	
Abc 后跟 xyz	abc xyz	abc xyz	abc xyz	
选项	|	|	|	
abc 的 0 或 1 个实例	[abc]	[abc]	[abc]	
abc 的 0 或更多个实例	{abc}	{abc}	{abc}*	X
abc 的 1 或更多个实例	{abc} +	{abc} +	{abc} +	
文本形式的分组	(...)	(...)	{...}	X
非末端符号 abc	abc	abc （针对一个 GFT 非末端 abc）或者 <u>abc</u> （针对一个 TTCN 非末端 abc）	<abc>	X
末端符号 abc	**abc**	**abc**	**abc** 或者 <名称> 或者 <字符串>	X

A.3 GFT语法

A.3.1 图

A.3.1.1 控制图

```
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 测试用例图

```
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 ]

/*  静态语义 – 包含一个布尔表达式的条件区域只能用在内嵌表达式中，即AltArea，以及调用内嵌表达式中，即只有CallArea。*/

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 函数图

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

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

FunctionBodyArea ::=
        TestcaseBodyArea
```

A.3.1.4 可选步骤图

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

AltstepHeading ::=
        AltstepKeyword AltstepIdentifier
        '('[AltstepFormalParList ] ')'
```

```
        [ RunsOnSpec ]
        { LocalDefinition [ SemiColon ] }


AltstepBodyArea ::=
        TestcaseBodyArea


/*  静态语义 – 可选步骤主体区域只能包含一个单个的可选内嵌表达式。*/
```

A.3.1.5 注释

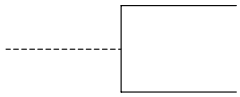
```
TextArea ::=
        TextSymbol
        contains ( { TTCN3Comments } [ MultiWithAttrib ] { TTCN3Comments } )
```

注意：对TTCN-3注释不存在明确的规则，它们在ITU-T Z.140建议书[1]第A.1.4节中进行解释。

/* 静态语义 – 在一个图中，至多只能有一个用于定义with语句的文本符号。*/

```
TextSymbol ::=
```

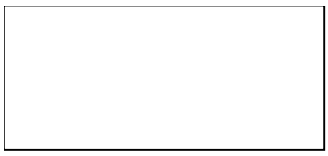
```
CommentArea ::=
        EventCommentSymbol contains TTCN3Comments
EventCommentSymbol ::=
```



/* 静态语义 – 注释符号可以附于GFT形式的任何图形符号上。*/

A.3.1.6 图

```
Frame ::=
```



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

/* 静态语义 – 在LocalDefinition中，不得以文本形式声明带创建、激活和执行语句以及带函数（包括通信函数）的常量与变量，而是必须以图形形式分别在创建、缺省、执行和引用符号中予以声明。*/

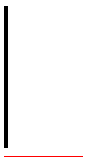
A.3.2 实例

A.3.2.1 部件实例

```
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

/* 静态语义 – 返回符号只能用在函数图中。*/
/* 静态语义 – 重复符号只能结束一个可选步骤图的部件实例。*/
```

A.3.2.2 端口实例

```
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 控制实例

```
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 实例结束

```
InstanceEndSymbol ::=
```

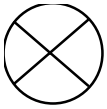
```
StopArea::=
        StopSymbol
        is associated with ( Expression )
```

/* 静态语义 – 表达式指的是**mtc**或其自身。*/

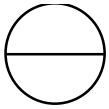
StopSymbol::=

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

ReturnSymbol::=

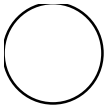


RepeatSymbol::=



```
GotoArea::=
        GotoSymbol
        contains LabelIdentifier
```

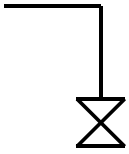
GotoSymbol::=



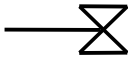
A.3.3 定时器

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


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


TimerStartSymbol ::=
        TimerStartSymbol1 | TimerStartSymbol2
```

TimerStartSymbol1 ::=

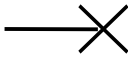


TimerStartSymbol2 ::=

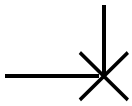


```
InstanceTimerStopArea ::=
        TimerStopArea1 | TimerStopArea2

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

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

TimerStopSymbol1 ::=

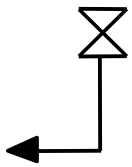


TimerStopSymbol2 ::=

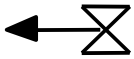


```
InstanceTimeoutArea ::=
        TimeoutArea1 | TimeoutArea2

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

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

TimeoutSymbol ::=
        TimeoutSymbol1 | TimeoutSymbol2
```

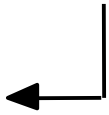
TimeoutSymbol1 ::=



TimeoutSymbol2 ::=



TimeoutSymbol3 ::=



A.3.4 行为

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

ActionSymbol ::=

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

/* 静态语义 – 在一个行为框中，不得以文本形式声明带创建、激活和执行语句以及带用户定义函数函数调用的常量与变量，而是必须以图形形式分别在创建、缺省、执行和引用符号中予以声明。*/

/* 静态语义 – 在一个行为框中，不得以文本形式进行带创建、激活和执行语句以及带用户定义函数函数调用的赋值，而是必须以图形形式分别在创建、缺省、执行和引用符号中进行。*/

/* 静态语义 – 只有那些不涉及通信操作的循环和条件构件，即那些只有“数据函数”的循环和条件构件，才可以包含在各行为框中。*/

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

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

/* 静态语义 – 在一个行为框中，不得以文本形式声明带创建、激活和执行语句以及带用户定义函数函数调用的常量与变量，而是必须以图形形式分别在创建、缺省、执行和引用符号中予以声明。*/

/* 静态语义 – 在一个行为框中，不得以文本形式进行带创建、激活和执行语句以及带用户定义函数函数调用的赋值，而是必须以图形形式分别在创建、缺省、执行和引用符号中进行。*/

A.3.5 调用

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

/* 静态语义 – 如果被调用函数执行规范说明，那么有关它的引用符号必须覆盖所有端口实例，对被调用的可选步骤也如此。*/
/* 静态语义 – 只有那些通过端口参数传入某个函数的端口实例，才必须被有关被调用函数（它不执行规范说明的）的引用符号所覆盖。注意：引用符号可以附于不作为端口参数传入函数的端口实例上。*/

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

ReferenceSymbol ::=

A.3.5.1 有关部件/控制实例的函数和可选步骤调用

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

InstanceInvocationBeginSymbol ::=
        VoidSymbol

InstanceInvocationEndSymbol ::=
        VoidSymbol
```

A.3.5.2 有关端口的函数和可选步骤调用

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

/* 静态语义 – 只有带函数实例和测试步骤实例的调用才需要附于某个端口实例上，在这种情况下，如果被调用函数执行规范说明，那么有关它的引用符号必须覆盖所有端口实例，对被调用的可选步骤也如此。*/

```
PortInvocationBeginSymbol ::=
        VoidSymbol

PortInvocationEndSymbol ::=
        VoidSymbol
```

A.3.5.3 测试用例执行

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

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

/* 静态语义 – 常量和变量声明以及赋值将用作最右侧表达式的一个执行语句。*/

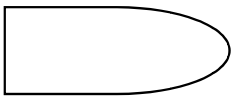
ExecuteSymbol ::=

A.3.6 激活/取消激活缺省

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

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

/* 静态语义 – 常量和变量声明以及赋值将用作最右侧表达式的一个激活语句。*/

DefaultSymbol ::=



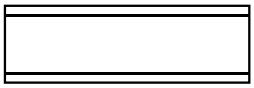
A.3.7 测试部件

A.3.7.1 创建测试部件

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

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

/* 静态语义 – 常量和变量声明以及赋值将用作最右侧表达式的一个创建语句。*/

CreateSymbol ::=



A.3.7.2 启动测试部件

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

StartSymbol ::=

A.3.7.3 停止测试部件

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

/* 静态语义 – 表达式指的是一个部件标识符。*/
/* 静态语义 – 如果部件停止其自身（如self.stop）或者停止测试执行（如mtc.stop），那么实例部件停止区域将用作一个内嵌表达式符号中某操作数的最后事件。*/

A.3.8 内嵌表达式

```
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 ]
```

/* 静态语义 – 如果SeparatorSymbol包含在内嵌表达式符号中，那么使用有关部件和端口实例的InstanceInlineExpressionSeparatorSymbols来将SeparatorSymbol附于各自的实例上。*/

```
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
```

/* 静态语义 – 操作数区域中的事件层不得拥有一个带布尔表达式的条件。*/

```
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 ]
```

/* 静态语义 – 每个部件的InstanceInlineExpressionSeparatorSymbol数量以及端口实例数量必须遵循内嵌表达式符号中所含的SeparatorSymbols数量：使用有关部件和端口实例的InstanceInlineExpressionSeparatorSymbol来将SeparatorSymbols附于各自的实例上。*/

```
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
```

/* 静态语义 – 首先，对alt内嵌表达式的各个操作数，或者为部件实例提供一个InstanceTimeoutArea，或者提供一个GuardOpLayer 。*/

```
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 ]
```

/* 静态语义 – 每个部件的InstanceInlineExpressionSeparatorSymbol数量以及端口实例数量必须遵循内嵌表达式符号中所含的SeparatorSymbols数量：使用有关部件和端口实例的InstanceInlineExpressionSeparatorSymbol来将SeparatorSymbols附于各自的实例上。*/

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

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

/* 静态语义 – 插入内嵌表达式区域中的连接器层不可以包含循环语句、跳转、激活、取消激活、停止、返回或函数调用。*/

```
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 ]
```

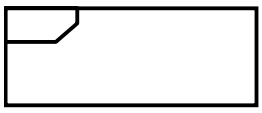
/* 静态语义 – 每个部件的InstanceInlineExpressionSeparatorSymbol数量以及端口实例数量必须遵循内嵌表达式符号中所含的SeparatorSymbols数量：使用有关部件和端口实例的InstanceInlineExpressionSeparatorSymbol来将SeparatorSymbols附于各自的实例上。*/

```
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
```

/* 静态语义 – 首先，对调用内嵌表达式GuardedCallOperandArea中的各个操作数，或者为部件实例提供一个InstanceCatchTimeoutWithinCallEventArea，或者提供一个CallBodyOpsLayer。*/

```
GuardedConditionLayer ::=
        BooleanExpressionConditionArea
        | DoneArea

CallBodyOpsLayer ::=
        GetreplyWithinCallArea
        | CatchWithinCallArea
```

InlineExpressionSymbol ::=



SeparatorSymbol::=

- - - - - - - - - - - -

A.3.8.1 有关部件实例的内嵌表达式

```
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
```

/* 静态语义 – 实例调用区域只能包含一个可选步骤实例。*/

```
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
```

/* 静态语义 – 实例事件区域不可以包含循环语句、跳转、激活、取消激活、停止、返回或函数调用。*/

```
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 有关端口的内嵌表达式

```
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 有关控制实例的内嵌表达式

```
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

/* 静态语义 – 实例调用区域只能包含一个可选步骤实例。*/

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

/* 静态语义 – 实例事件区域不可以包含循环语句、跳转、激活、取消激活、停止、返回或函数调用。*/
```

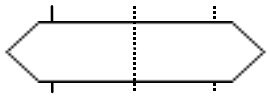
A.3.9 条件

```
ConditionArea ::=
        PortOperationArea

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

/* 静态语义 – 条件中的布尔表达式只能用作alt和调用内嵌表达式的防护。它们只能附于一个单个的测试部件上或附于控制实例上。*/

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

/*  静态语义 – SingleExpression必须解析为一个类型判定值。*/
/*  静态语义 – SetLocalVerdict不得用于指派值错误。*/
/*  静态语义 – 如果使用了关键字pass、fail、inconc，那么不得使用带setverdict关键字的形式。*/
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
```

/* 静态语义 – 条件符号将附于所有端口上或者只附于一个端口上。*/

如果条件符号跨越本端口操作中未涉及之端口的端口轴线符号，那么所画的端口轴线符号穿过：



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

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

ConditionSymbol ::=

A.3.9.1 有关部件实例的条件

```
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 有关端口的条件

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

PortConditionBeginSymbol ::=
        VoidSymbol

PortConditionEndSymbol ::=
        VoidSymbol
```

A.3.10 基于消息的通信

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

/* 静态语义 – 如果存在的话，类型将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，导出的定义将置于消息符号之下。*/
/* 静态语义 – 模板将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，to子句将置于消息符号之下。 */

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

/* 静态语义 – 如果存在的话，类型将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，导出的定义将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，模板将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，from子句将置于消息符号之下。 */
/* 静态语义 – 如果存在的话，端口重定向将置于消息符号之下。*/
```

MessageSymbol ::=

A.3.10.1 有关部件实例的、基于消息的通信

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

MessageOutSymbol ::=
        VoidSymbol
```

VoidSymbol是一个不带空间扩展的集合点。

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

MessageInSymbol ::=
        VoidSymbol
```

A.3.10.2 有关端口实例的、基于消息的通信

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

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

A.3.11 基于签名的通信

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

```
/* 静态语义 – 如果存在的话，特征将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，导出的定义将置于消息符号之下。*/
/* 静态语义 – 模板将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，to子句将置于消息符号之下。 */

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

/* 静态语义 – 如果存在的话，特征将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，导出的定义将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，模板将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，from子句将置于消息符号之下。 */
/* 静态语义 – 如果存在的话，端口重定向将置于消息符号之下。*/

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

/* 静态语义 – 如果存在的话，特征将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，导出的定义将置于消息符号之下。*/
/* 静态语义 – 模板将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，答复值将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，to子句将置于消息符号之下。 */

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

/* 静态语义 – 如果存在的话，特征将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，导出的定义将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，模板将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，值匹配规范说明将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，from子句将置于消息符号之下。 */
/* 静态语义 – 如果存在的话，端口重定向将置于消息符号之下。*/

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

/* 静态语义 – 如果存在的话，特征将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，导出的定义将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，模板将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，值匹配规范说明将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，from子句将置于消息符号之下。 */
/* 静态语义 – 如果存在的话，端口重定向将置于消息符号之下。*/

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

/* 静态语义 – 特征将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，异常类型将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，导出的定义将置于消息符号之下。*/
/* 静态语义 – 模板将置于消息符号之下。*/
```

```
/* 静态语义 – 如果存在的话，to子句将置于消息符号之下。*/

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

/* 静态语义 – 特征将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，异常类型将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，导出的定义将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，模板将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，from子句将置于消息符号之下。 */
/* 静态语义 – 如果存在的话，端口重定向将置于消息符号之下。*/

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

/* 静态语义 – 特征将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，异常类型将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，导出的定义将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，模板将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，from子句将置于消息符号之下。 */
/* 静态语义 – 如果存在的话，端口重定向将置于消息符号之下。*/
```

A.3.11.1 有关部件实例的、基于签名的通信

```
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 有关端口的、基于签名的通信

```
PortGetcallOutEventArea::=
        PortOutMsgEventArea

PortGetreplyOutEventArea::=
        PortOutMsgEventArea

PortCatchOutEventArea::=
        PortOutMsgEventArea

PortCallInEventArea::=
        PortInMsgEventArea

PortReplyInEventArea::=
        PortInMsgEventArea

PortRaiseInEventArea::=
        PortInMsgEventArea
```

A.3.12 触发与检查

A.3.12.1 有关部件实例的触发和检查

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

/* 静态语义 – 触发关键字将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，类型将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，导出的定义将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，模板将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，from子句将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，端口重定向将置于消息符号之下。*/

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

/* 静态语义 – 检查关键字将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，检查操作信息将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，检查数据将置于消息符号之下。*/

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

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

/* 静态语义 – 值匹配规范说明只能与getreply一起使用。*/
/* 静态语义 – 带参数的端口重定向只能与getcall 和 getreply一起使用。*/
```

```
InstanceTriggerEventArea ::=
        InstanceReceiveEventArea

InstanceCheckEventArea ::=
        InstanceReceiveEventArea
```

A.3.12.2 有关端口实例的触发和检查

```
PortTriggerOutEventArea ::=
        PortOutMsgEventArea

PortCheckOutEventArea ::=
        PortOutMsgEventArea
```

A.3.13 对来自任何端口的通信的处理

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

/* 静态语义 – 标签标识符将置于标签符号的圆圈内。*/

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

/* 静态语义 – 如果存在的话，类型将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，导出的定义将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，模板将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，from子句将置于消息符号之下。 */
/* 静态语义 – 如果存在的话，端口重定向将置于消息符号之下。*/

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

/* 静态语义 – 触发将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，类型将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，导出的定义将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，模板将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，from子句将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，端口重定向将置于消息符号之下。*/

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

/* 静态语义 – 如果存在的话，特征将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，导出的定义将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，模板将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，from子句将置于消息符号之下。 */
/* 静态语义 – 如果存在的话，端口重定向将置于消息符号之下。*/
```

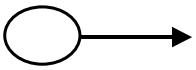
```
FoundGetReply ::=
        FoundSymbol
        is associated with GetreplyKeyword [ Signature ]
        is associated with ( [ [ DerivedDef AssignmentChar ] TemplateBody ]
                             [ ValueMatchSpec ]
                             [ FromClause ] [ PortRedirectWithParam ] )
     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

/* 静态语义 – 如果存在的话，特征将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，导出的定义将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，模板将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，值匹配规范说明将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，from子句将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，端口重定向将置于消息符号之下。*/

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

/* 静态语义 – 特征将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，异常类型将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，导出的定义将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，模板将置于消息符号之下。*/
/* 静态语义 – 如果存在的话，from子句将置于消息符号之下。 */
/* 静态语义 – 如果存在的话，端口重定向将置于消息符号之下。*/

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

/* 静态语义 – 检查符号将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，检查操作信息将置于消息符号之上。*/
/* 静态语义 – 如果存在的话，检查数据将置于消息符号之下。*/
```

FoundSymbol ::=

A.3.14 标签

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

/* 静态语义 – 标签标识符将置于标签符号的圆圈内。 */
```

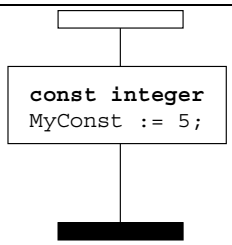
LabellingSymbol ::=

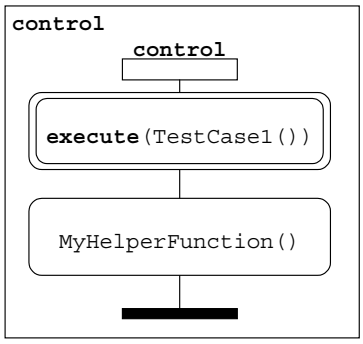
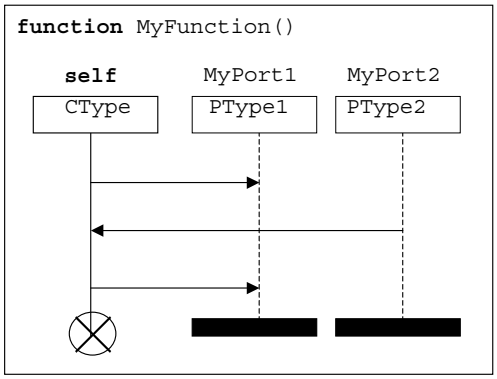
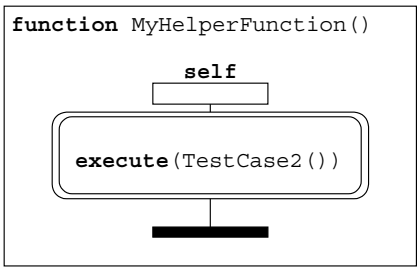
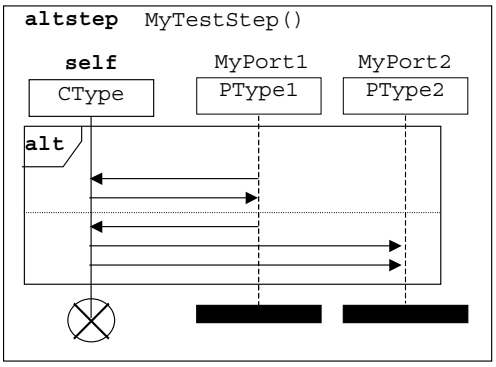


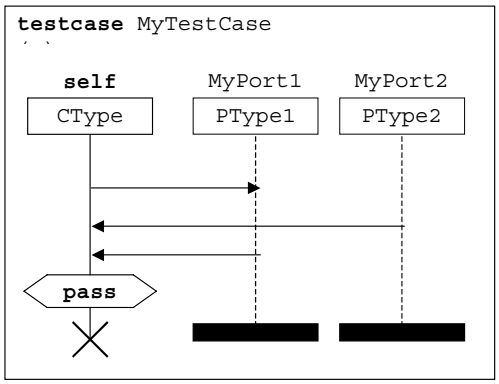
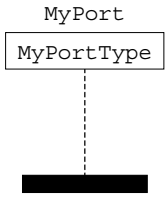
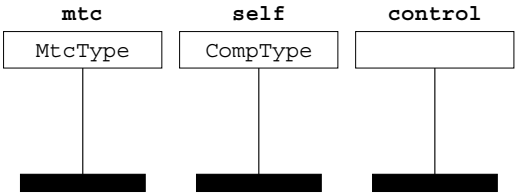
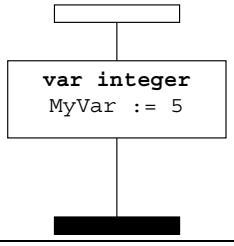
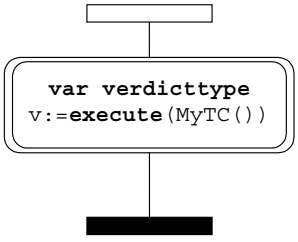
附 件 B

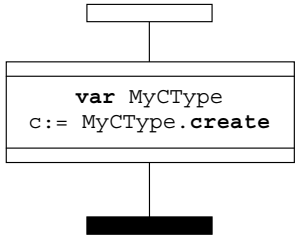
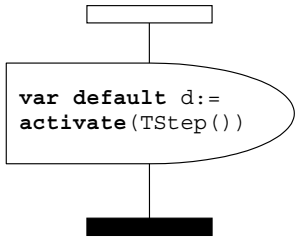
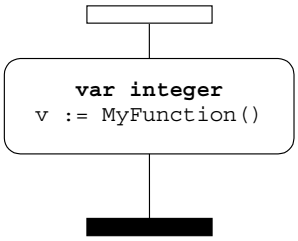
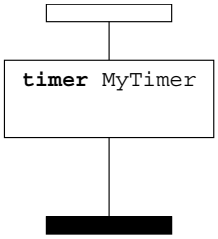
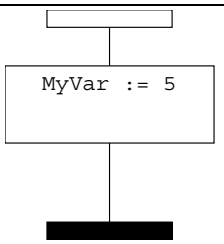
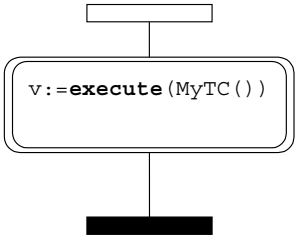
GFT参考指南

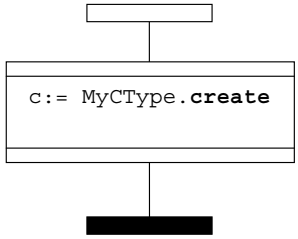
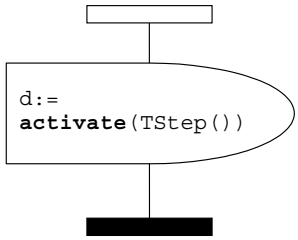
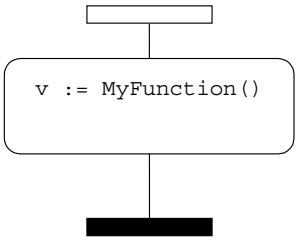
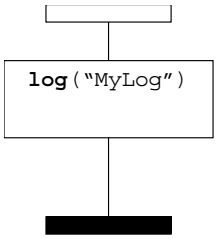
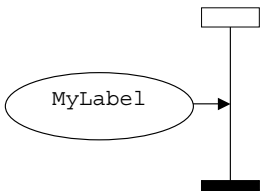
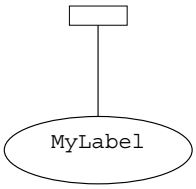
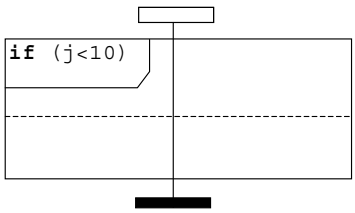
本附件列出了主要的 TTCN-3 语言元素及其 GFT 表示。为了完整描述 GFT 符号及其用法，请参考主文本。

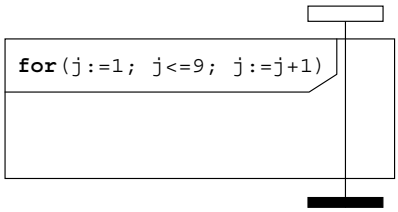
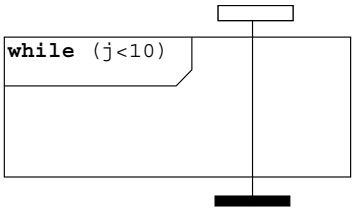
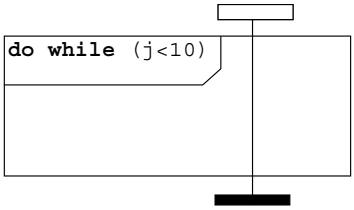
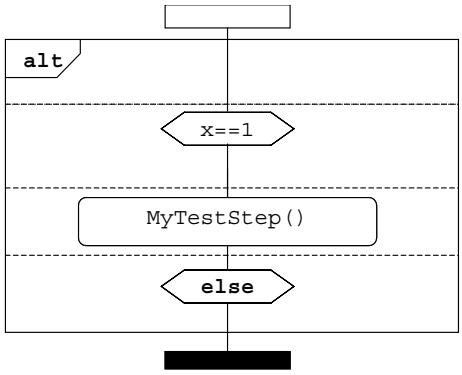
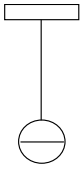
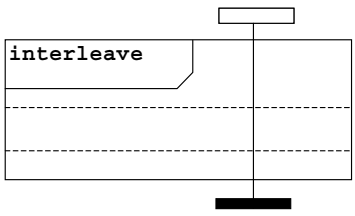
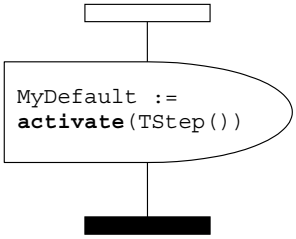
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注 释
模块定义			
TTCN-3 模块定义	**module**		非特殊的 GFT 符号，也就是说，可以使用核心语言或另一种表示格式。
从其他模块输入定义	**import**		非特殊的 GFT 符号，也就是说，可以使用核心语言或另一种表示格式。
定义分组	**group**		非特殊的 GFT 符号，也就是说，可以使用核心语言或另一种表示格式。
数据类型定义	**type**		非特殊的 GFT 符号，也就是说，可以使用核心语言或另一种表示格式。
通信端口定义	**port**		非特殊的 GFT 符号，也就是说，可以使用核心语言或另一种表示格式。
测试部件定义	**component**		非特殊的 GFT 符号，也就是说，可以使用核心语言或另一种表示格式。
特征定义	**signature**		非特殊的 GFT 符号，也就是说，可以使用核心语言或另一种表示格式。
外部函数/常量定义	**external**		非特殊的 GFT 符号，也就是说，可以使用核心语言或另一种表示格式。
常量定义	**const**	`const integer MyConst := 5;`	控制、测试用例、测试步骤或函数图标题中的文本常量声明。
			行为款中的局部常量声明。
数据/特征模板定义	**template**		非特殊的 GFT 符号，也就是说，可以使用核心语言或另一种表示格式。

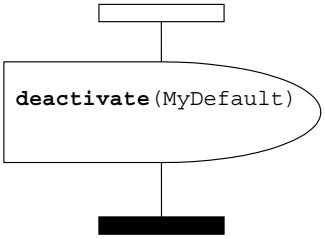
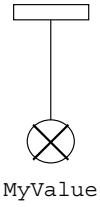
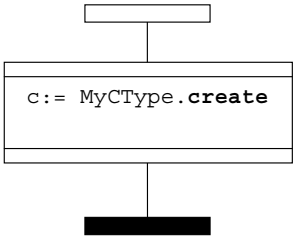
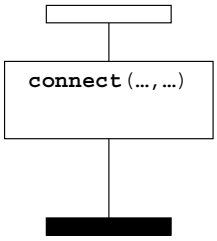
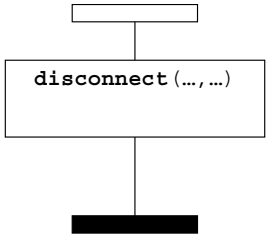
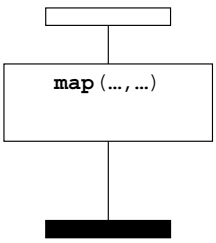
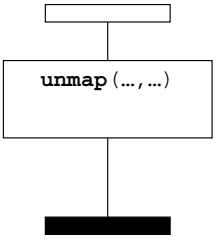
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
控制定义	**control**		表示一个 TTCN-3 模块控制部分的 GFT 控制图。
函数定义	**function**		GFT 函数图用于表示函数。
			可定义 GFT 函数图来构建 TTCN-3 模块控制部分的行为。
可选步骤定义	**altstep**		GFT 可选步骤图用于表示可选步骤。

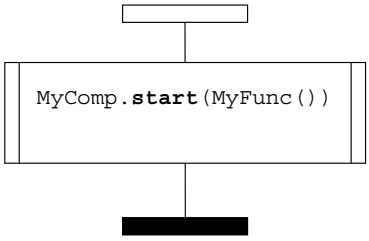
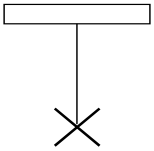
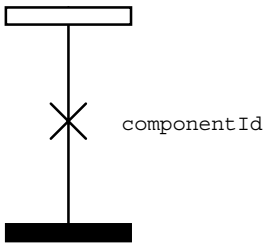
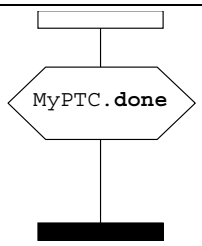
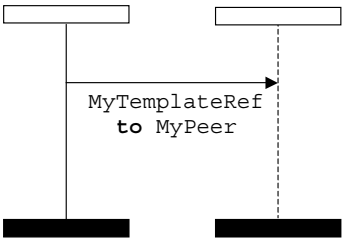
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
测试用例定义	**testcase**		GFT 测试用例图用于表示测试用例。
部件实例和端口用法			
端口实例			通过一个带虚线实例线的实例来表示测试用例、测试步骤和函数图中的端口。端口名称在上面指定，（可选的）端口类型在实例头中进行描述。
测试部件实例			一个 **mtc** 实例表示一个测试实例图中的主测试部件。 一个 **self** 实例表示一个测试步骤或函数图中的一个测试部件。 一个 **control** 实例表示执行一个控制图中模块控制部分的实例。
声明			
变量声明	**var**	**var integer** MyVar := 5	一个控制、测试用例、测试步骤或函数图标题中文本形式的变量声明。
			一个行为框中的变量声明。
			一个测试执行符号中的变量声明。

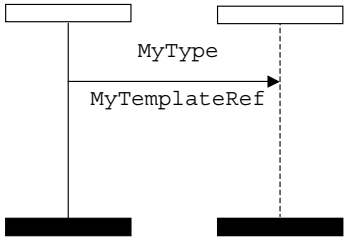
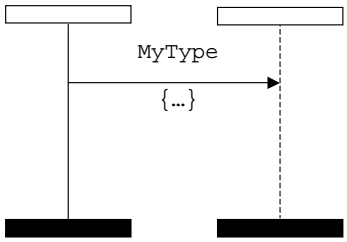
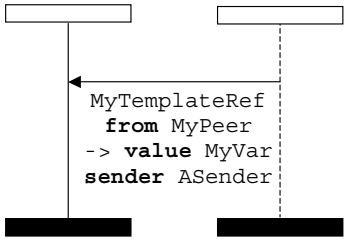
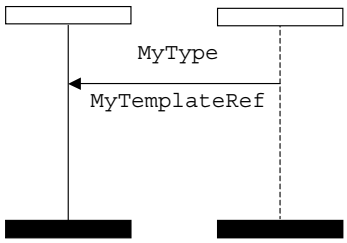
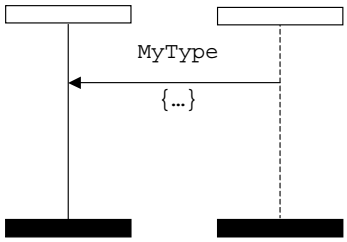
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
			一个测试部件创建符号中的变量声明。
			一个缺省激活符号中的变量声明。
			一个引用符号中的变量声明。
定时器声明	**timer**	**timer** MyTimer	一个控制、测试用例、测试步骤或函数图标题中文本形式的定时器声明。
			一个行为框中的定时器声明。
基本编程语句			
表达式	**(…)**		非特殊的 GFT 符号，也就是说，可以使用核心语言或另一种表示格式。
赋值	**:=**		一个行为框中的赋值。
			一个测试用例执行符号中的赋值。

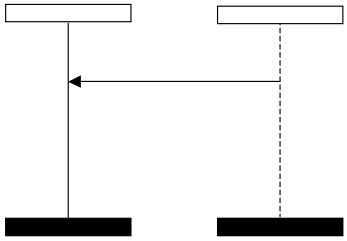
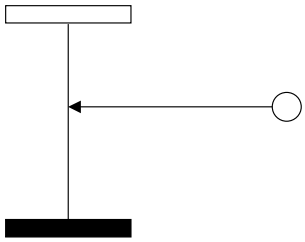
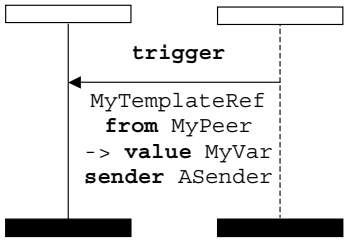
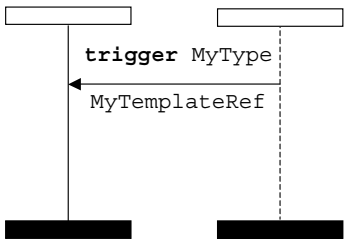
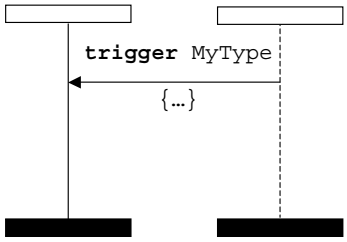
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
			一个测试部件创建符号中的赋值。
			一个缺省激活符号中的赋值。
			一个引用符号中的赋值。
日志	**log**		将日志语句置于一个行为框中。
标签和跳转	**label**		一个标签的定义。
	goto		跳转至标签。
If-else	**if (…) {…}** **else {…}**		

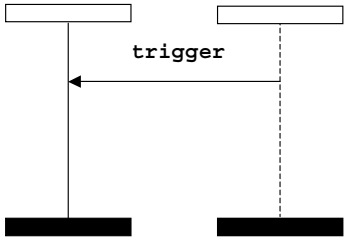
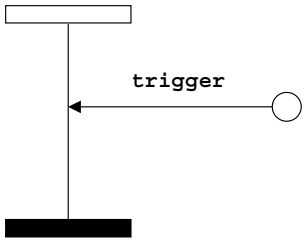
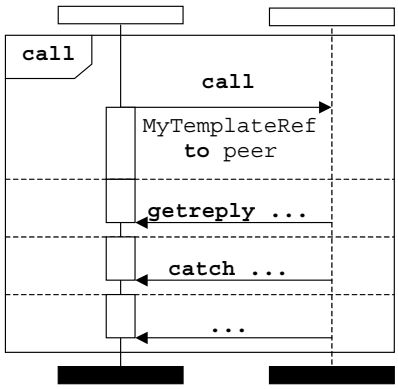
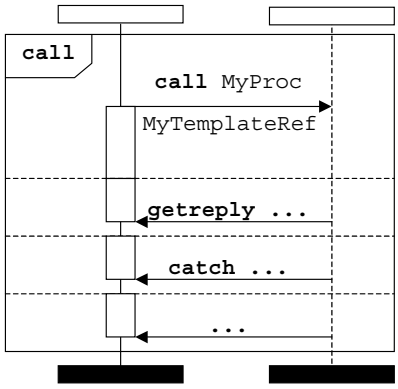
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
For 循环	**for (…) {…}**		
While 循环	**while (…)** **{…}**		
Do while 循环	**do {…}** **while (…)**		
行为编程语句			
可选的行为	**alt {…}**		
重复	**repeat**		在可选行为和测试步骤中使用。
插入的行为	**interleave** **{…}**		
激活缺省	**activate**		将激活语句置于一个缺省符号中。

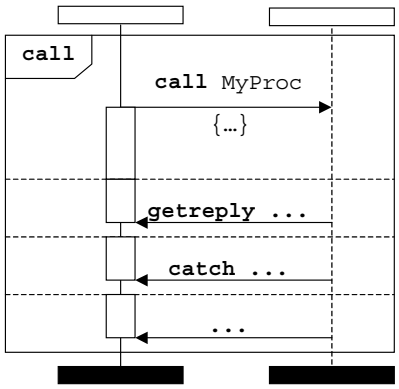
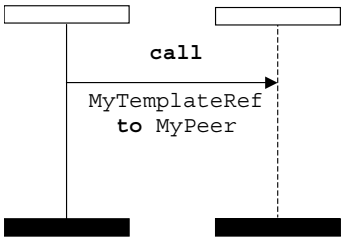
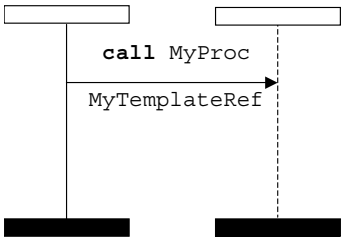
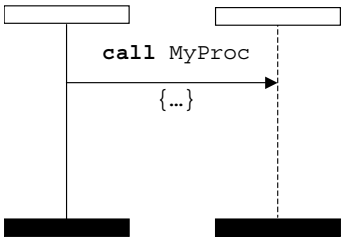
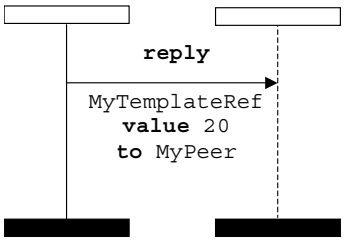
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
取消激活缺省	**deactivate**		将取消激活语句置于一个缺省符号中。
返回控制	**return**		可选的返回值附于返回符号上。
配置操作			
创建并行测试部件	**create**		将创建语句置于一个测试部件创建符号中。
部件与部件连接	**connect**		将连接语句置于一个行为框中。
取消两个部件的连接	**disconnect**		将断开连接语句置于一个行为框中。
将端口映射至测试系统接口	**map**		将映射语句置于一个行为框中。
取消端口与测试系统接口的映射	**unmap**		将取消映射语句置于一个行为框中。

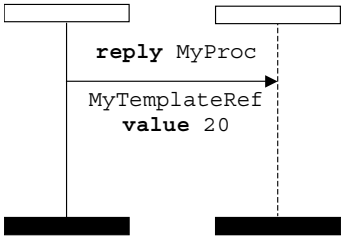
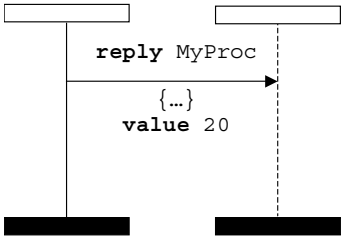
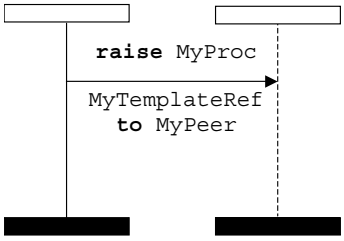
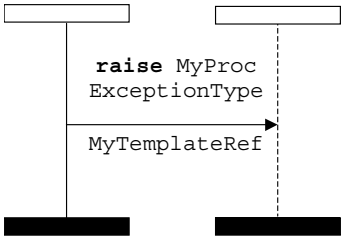
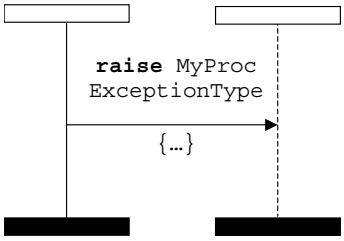
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
获得 MTC 地址	**mtc**		非特殊的 GFT 符号，用在语句、表达式中，或者作为测试部件标识符。
获得测试系统接口地址	**system**		非特殊的 GFT 符号，用在语句、表达式中。
获得自身地址	**self**		非特殊的 GFT 符号，用在语句、表达式中，或者作为测试部件标识符。
启动执行测试部件	**start**		启动语句置于启动符号中。
自身停止执行测试部件	**stop**		终止 mtc 也将终止所有其他测试部件。 不能停止端口实例。
停止执行另一个测试部件			将部件标识符置于停止符号附近。
检查 PTC 的终止情况	**running**		非特殊的 GFT 符号，用在表达式中。
等待 PTC 的终止	**done**		将完成语句置于一个条件符号中。
通信操作			
发送消息	**send**		发送一个由模板引用定义的、但不带类型信息的消息。 （可选的）**to-**指令可用于唯一确定对等实体。

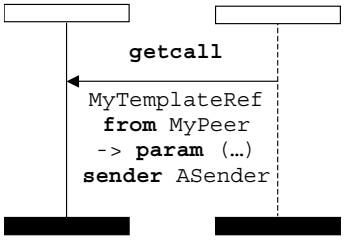
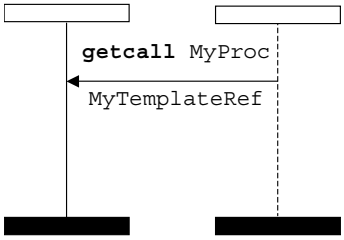
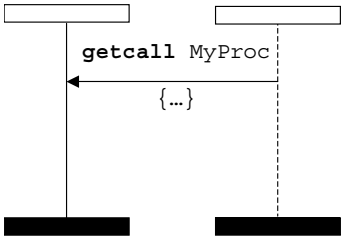
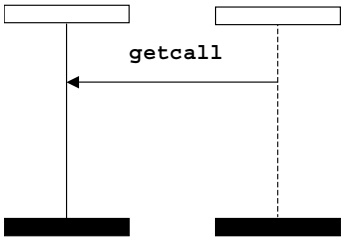
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
			发送一个由模板引用定义的、但带类型信息的消息。 （可选的）**to-**指令可用于唯一确定对等实体。
			发送一个由内嵌模板定义定义的消息。 （可选的）**to-**指令可用于唯一确定对等实体。
接收消息	**receive**		接收一个消息，它带一个由模板引用定义的值，但不带类型信息。 （可选的）**from-**指令表示消息的发送者将由变量 MyPeer 来确定。 （可选的）**value-**指令将收到的消息指派给变量 MyVal。 （可选的）**sender-**指令取得发送者标识符，并将之保存在变量 ASender 中。
			接收一个消息，它带一个由模板引用定义的值以及类型消息。 可选的 **from-**、**value-**和 **sender-**指令可用于确定消息的发送者、将消息指派给变量或者取得对等实体的标识符。
			接收一个消息，它带一个由内嵌模板定义定义的值。 可选的 **from-**、**value-**和 **sender-**指令可用于确定消息的发送者、将消息指派给变量或者取得对等实体的标识符。

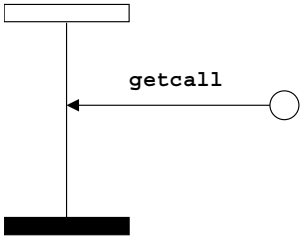
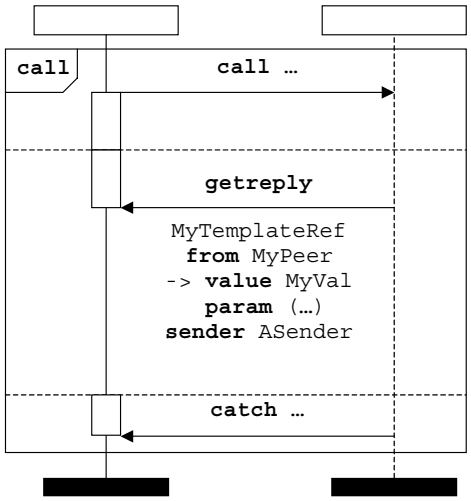
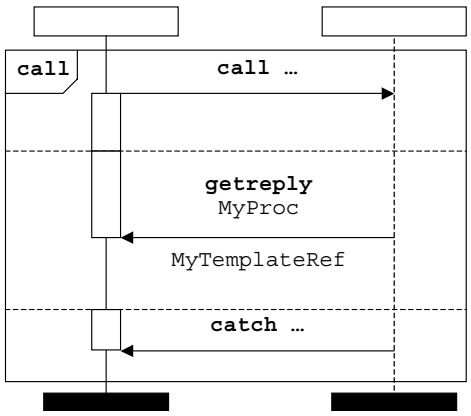
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
			接收任何消息（不指定任何值和任何类型）。 可选的 **from-**、**value-**和 **sender-**指令可用于确定消息的发送者、将消息指派给变量或者取得对等实体的标识符。
			接收来自任何端口的任何消息（不指定任何值和任何类型）。 可以通过引用模板或使用内嵌模板的方式来限制将来自任何端口的消息值。 可选的 **from-**、**value-**和 **sender-**指令可用于确定消息的发送者、将消息指派给变量或者取得对等实体的标识符。
触发消息	**trigger**		触发一个消息，它带一个由模板引用定义的值，但不带类型信息。 （可选的）**from-**指令表示消息的发送者将由变量 MyPeer 来确定。 （可选的）**value-**指令将收到的消息指派给变量 MyVal。 （可选的）**sender-**指令取得发送者标识符，并将之保存在变量 ASender 中。
			触发一个消息，它带一个由模板引用定义的值以及类型信息。 可选的 **from-**、**value-**和 **sender-**指令可用于确定消息的发送者、将消息指派给变量或者取得对等实体的标识符。
			触发一个消息，它带一个由内嵌模板定义规定的值。 可选的 **from-**、**value-**和 **sender-**指令可用于确定消息的发送者、将消息指派给变量或者取得对等实体的标识符。

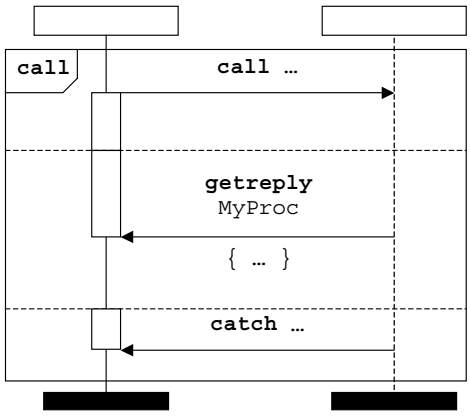
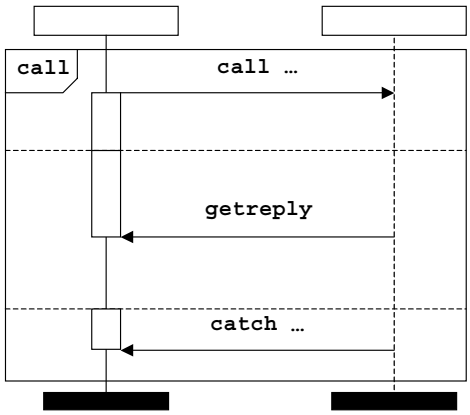
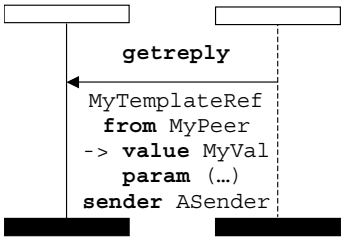
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
			触发任何消息（不指定任何值和任何类型）。 可选的 **from-**、**value-**和 **sender-**指令可用于确定消息的发送者、将消息指派给（**anytype** 类型的）变量并且取得对等实体的标识符。
			触发来自任何端口的任何消息（不指定任何值和任何类型）。 将引起任何端口触发的消息值可以通过引用模板或使用内嵌模板的方式来限制。 可选的 **from-**、**value-**和 **sender-**指令可用于确定消息的发送者、将消息指派给（**anytype** 类型的）变量并且取得对等实体的标识符。
调用阻塞的过程调用	**call**		通过使用一个特征模板来调用一个阻塞过程。 通过（可选的）**to-**指令来唯一确定接收者。 只以示意图形式来显示调用体，即可能的 **getreply** 和 **catch** 操作。
			通过使用一个特征模板和特征信息来调用一个阻塞过程。 （可选的）**to-**指令可用于唯一确定对等实体。 只以示意图形式来显示调用体，即可能的 **getreply** 和 **catch** 操作。

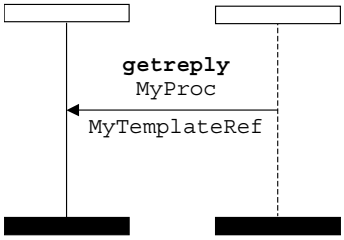
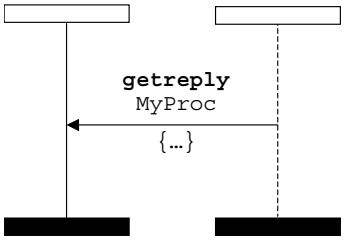
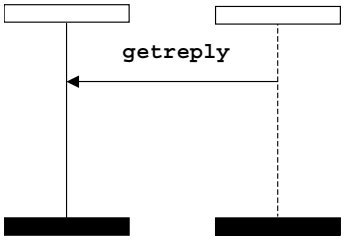
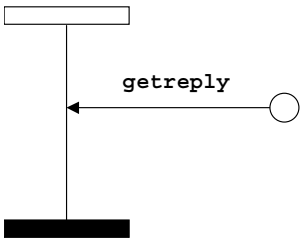
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
			通过使用一个内嵌模板来调用一个阻塞过程。 （可选的）**to-**指令可用于唯一确定对等实体。 只以示意图形式来显示调用体，即可能的 **getreply** 和 **catch** 操作。
调用非阻塞的过程调用	**call**		调用一个远程过程，调用通过一个模板引用来定义，但不带特征信息。 通过（可选的）**to-**指令来唯一确定接收者。
			调用远程过程 MyProc。调用通过一个模板引用来定义。 （可选的）**to-**指令可用于唯一确定对等实体。
			调用远程过程 MyProc。调用通过一个内嵌模板来定义。 （可选的）**to-**指令可用于唯一确定对等实体。
来自远程实体的过程调用答复	**reply**		答复远程过程调用。答复通过一个模板引用和可能的返回值（**value-**指令）来定义。 注 — 特征信息是模板定义的一部分。 通过（可选的）**to-**指令来唯一确定接收者。

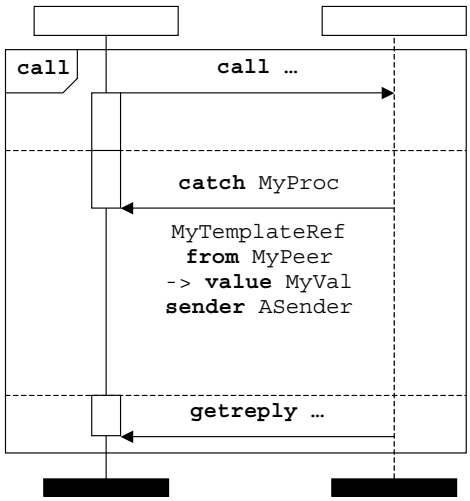
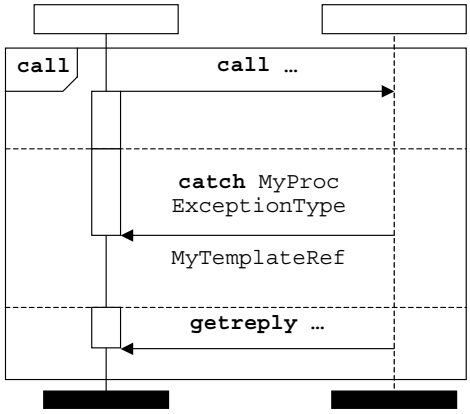
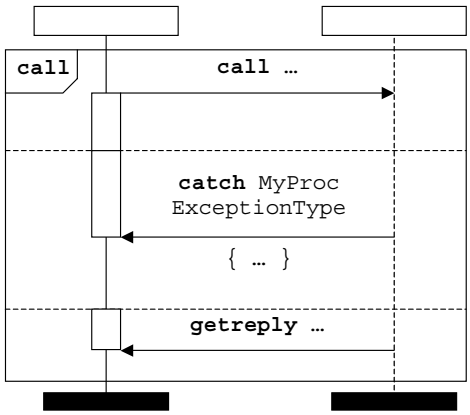
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
			答复 MyProc 的远程过程调用。答复通过一个模板引用和可能的返回值（**value-**指令）来定义。 （可选的）**to-**指令可用于唯一确定对等实体。
			答复 MyProc 的远程过程调用。答复通过一个内嵌模板和可能的返回值（**value-**指令）来定义。 （可选的）**to-**指令可用于唯一确定对等实体。
引起异常（对一个已接受的调用）	**raise**		对一个已接受的 MyProc 调用引起一个异常。异常通过一个模板引用来定义。 注 — 异常类型在模板定义中进行定义。 通过（可选的）**to-**指令来唯一确定接收者。
			对一个已接受的 MyProc 调用引起一个异常。异常通过其（可选的）类型和一个模板引用来定义。 （可选的）**to-**指令可用于唯一确定对等实体。
			对一个已接受的 MyProc 调用引起一个异常。异常通过其类型和一个内嵌模板来定义。 （可选的）**to-**指令可用于唯一确定对等实体。

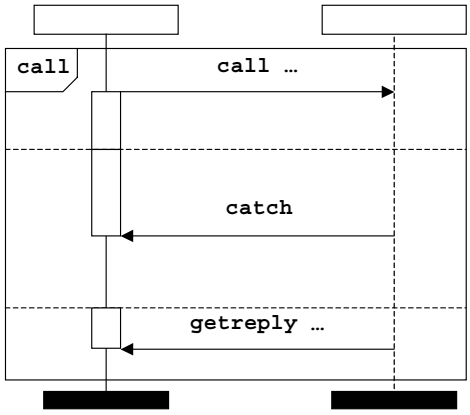
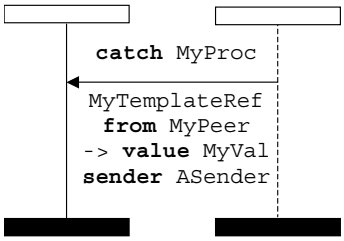
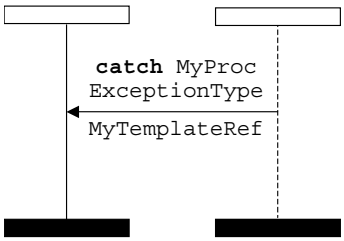
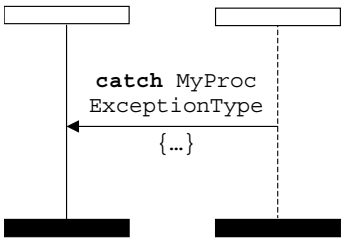
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
接受来自远程实体的过程调用	**getcall**		接受一个来自远程实体的过程调用。调用特征必须匹配模板引用定义的条件。 注 — 特征信息是模板定义的一部分。 （可选的）**from-**指令表示调用的发送者将由变量 MyPeer 来确定。 （可选的）**param-**指令将 in-参数值指派给变量。 （可选的）**sender-**指令取得发送者标识符，并将之保存在变量 ASender 中。
			接受一个来自远程实体的过程调用。调用特征必须匹配特征引用和模板引用定义的条件。 可选的 **from-**、**param-**和 **sender-**指令可用于确定调用的发送者、将 in-参数指派给变量或者取得对等实体的标识符。
			接受一个来自远程实体的过程调用。调用特征必须匹配特征引用和内嵌模板定义规定的条件。 可选的 **from-**、**param-**和 **sender-**指令可用于确定调用的发送者、将 in-参数指派给变量或者取得对等实体的标识符。
			接受来自任何远程实体的任何过程。 可选的 **from-**和 **sender-**指令可用于确定调用的发送者、将 in-参数指派给变量或者取得对等实体的标识符。

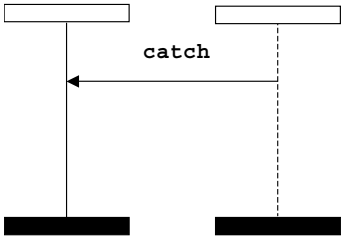
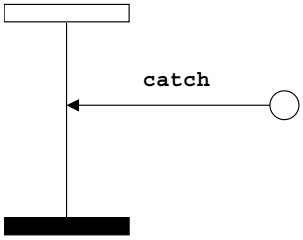
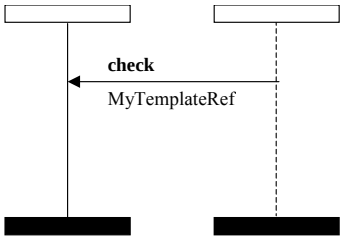
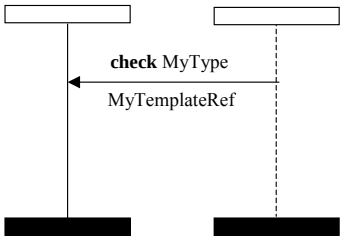
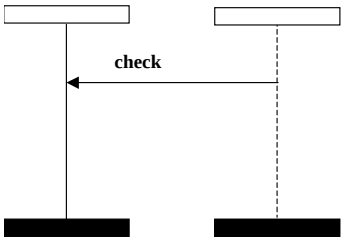
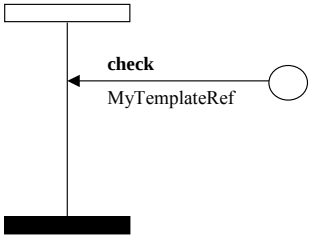
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
			接受任何端口上来自任何远程实体的任何过程。 可以通过引用模板或使用内嵌模板的方式来限制将来自任何端口的调用。 可选的 **from-**、**param-**和 **sender-**指令可用于确定调用的发送者、将 in-参数指派给变量或者取得对等实体的标识符。
处理来自之前阻塞调用的响应	**getreply**		接收一个来自阻塞调用的响应。答复必须匹配模板引用定义的条件。 注 — 特征信息是模板定义的一部分。 （可选的）**from-**指令表示调用的发送者将由变量 MyPeer 来确定。 （可选的）**value-**指令将过程可能的返回值指派给变量 MyVal。 （可选的）**param-**指令将 out-参数值指派给变量。 （可选的）**sender-**指令取得发送者标识符，并将之保存在变量 ASender 中。
			接收一个来自阻塞调用的响应。答复必须匹配特征引用和模板引用定义的条件。 可选的 **from-**、**value-**、**param-**和 **sender-**指令可用于确定答复的发送者、取得过程的返回值并将 in-参数指派给变量或者取得对等实体的标识符。

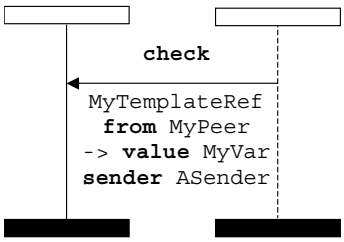
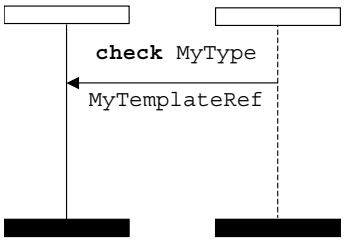
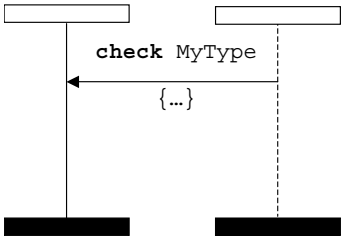
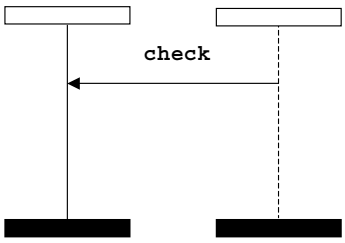
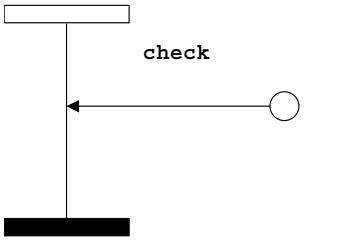
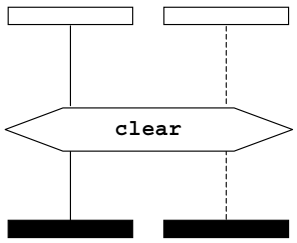
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
			接收一个来自阻塞调用的响应。答复必须匹配特征引用和内嵌模板定义规定的条件。 可选的 **from-**、**value-**、**param-**和 **sender-**指令可用于确定答复的发送者、取得过程的返回值并将 in-参数指派给变量或者取得对等实体的标识符。
			接受来自阻塞调用的任何响应。
处理来自之前非阻塞调用的响应，或者处理独立于调用的响应	**getreply**		接收一个来自之前调用的响应。答复必须匹配特征引用定义的条件。 注 — 特征信息是模板定义的一部分。 （可选的）**from-**指令表示调用的发送者将由变量 MyPeer 来确定。 （可选的）**value-**指令将过程可能的返回值指派给变量 MyVal。 （可选的）**param-**指令将 out-参数值指派给变量。 （可选的）**sender-**指令取得发送者标识符，并将之保存在变量 ASender 中。

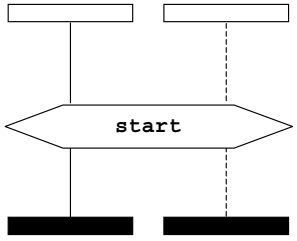
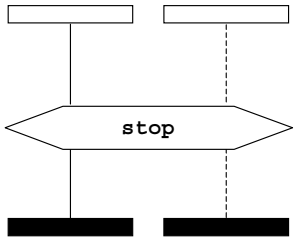
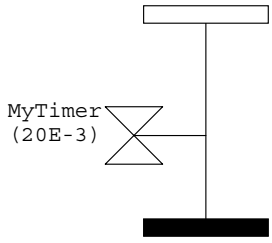
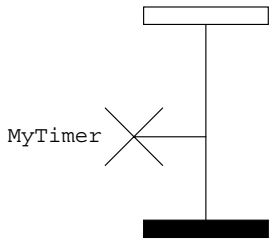
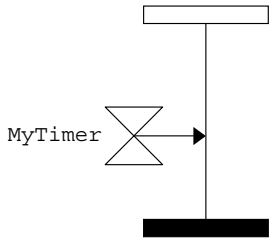
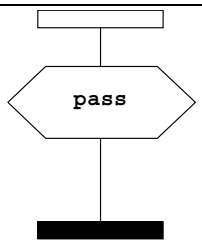
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
			接收一个来自之前调用的响应。答复必须匹配特征引用和模板引用定义的条件。 可选的 **from-**、**value-**、**param-**和 **sender-**指令可用于确定答复的发送者、取得过程的返回值并将 in-参数指派给变量或者取得对等实体的标识符。
			接收一个来自之前调用的响应。答复必须匹配特征引用和内嵌模板定义规定的条件。 可选的 **from-**、**value-**、**param-**和 **sender-**指令可用于确定答复的发送者、取得过程的返回值并将 in-参数指派给变量或者取得对等实体的标识符。
			接受来自任何之前调用的任何响应。 可选的 **from-**和 **sender-**指令可用于确定答复的发送者或者取得对等实体的标识符。
			接受任何端口上来自任何之前调用的任何响应。 可以通过引用模板或使用内嵌模板的方式来限制将来自任何端口的答复。 可选的 **from-**、**value-**、**param-**和 **sender-**指令可用于确定答复的发送者、取得过程的返回值并将 in-参数指派给变量或者取得对等实体的标识符。

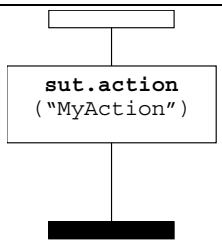
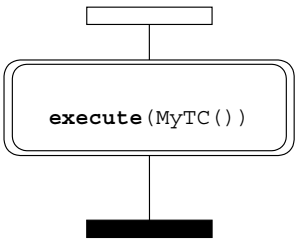
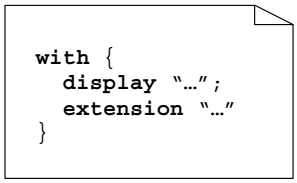
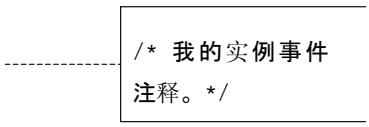
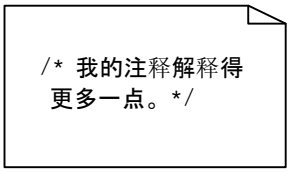
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
捕获来自之前阻塞调用的异常	**catch**		捕获一个来自之前调用的异常。异常必须匹配模板引用定义的条件。 注 — 类型信息是模板定义的一部分。 （可选的）**from-**指令表示异常的发送者将由变量 MyPeer 来确定。 （可选的）**value-**指令将异常值指派给变量 MyVal。 （可选的）**sender-**指令取得发送者标识符，并将之保存在变量 ASender 中。
			捕获一个来自之前调用的异常。异常必须匹配异常类型和模板引用定义的条件。 （可选的）**from-**、**value-**和 **sender-**指令可用于确定异常的发送者、取得异常值或者取得对等实体的标识符。
			捕获一个来自之前调用的异常。异常必须匹配异常类型和内嵌模板定义规定的条件。 可选的 **from-**、**value-**和 **sender-**指令可用于确定异常的发送者、取得异常值或者取得对等实体的标识符。

语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
			接受来自阻塞调用的任何异常。 可选的 **from-**、**value-**和 **sender-**指令可用于确定异常的发送者、取得异常值（并将之指派给一个 **anytype** 类型的变量）或者取得对等实体的标识符。
捕获来自之前阻塞调用的异常，或者捕获独立于调用的异常	**catch**		捕获一个来自之前调用的异常。异常必须匹配模板引用定义的条件。 注 — 类型信息是模板定义的一部分。 （可选的）**from-**指令表示异常的发送者将由变量 MyPeer 来确定。 （可选的）**value-**指令将异常值指派给变量 MyVal。 （可选的）**sender-**指令取得发送者标识符，并将之保存在变量 ASender 中。
			捕获一个来自之前调用的异常。异常必须匹配异常类型和模板引用定义的条件。 可选的 **from-**、**value-**和 **sender-**指令可用于确定异常的发送者、取得异常值或者取得对等实体的标识符。
			捕获一个来自之前调用的异常。异常必须匹配异常类型和内嵌模板定义规定的条件。 可选的 **from-**、**value-**和 **sender-**指令可用于确定异常的发送者、取得异常值或者取得对等实体的标识符。

语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
			捕获来自任何之前调用的任何异常。 可选的 **from-**、**value-**和 **sender-**指令可用于确定异常的发送者、取得异常值（并将之指派给一个 **anytype** 类型的变量）或者取得对等实体的标识符。
			捕获任何端口上来自任何之前调用的任何异常。 可以通过引用模板或使用内嵌模板的方式来限制将来自任何端口的异常。 可选的 **from-**、**value-**和 **sender-**指令可用于确定异常的发送者、取得异常值或者取得对等实体的标识符。
检查（当前）接收的消息/调用	**check**	    也可与 getcall、getreply 和 catch 一起使用。	带模板、不带类型 带模板、带类型 不带模板、不带类型（来自该端口的任何消息） 带模板、不带类型、不带端口（来自该端口的本消息）

语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
检查当前的消息、调用、答复或异常	**check**		检查是否已经收到一个带有模板引用定义的值的消息。 语法遵循有关消息接收的语法。 注 — 检查也可与 **getcall**、**getreply** 和 **catch** 一起使用。
			检查是否已经收到一个带有模板引用定义的值的消息。 语法遵循有关消息接收的语法。 注 — 检查也可与 **getcall**、**getreply** 和 **catch** 一起使用。
			检查是否已经收到一个带有内嵌模板定义规定的值的消息。 语法遵循有关消息接收的语法。 注 — 检查也可与 **getcall**、**getreply** 和 **catch** 一起使用。
			检查是否已经收到任何消息（未指定任何值和任何类型）。 语法遵循有关消息接收的语法。 注 — 检查也可与 **getcall**、**getreply** 和 **catch** 一起使用。
			检查是否已经在任何端口上收到任何消息（未指定任何值和任何类型）。 语法遵循有关消息接收的语法。 注 — 检查也可与 **getcall**、**getreply** 和 **catch** 一起使用。
清除端口	**clear**		将清除端口语句置于一个条件符号中。条件将只覆盖要清除的端口实例。

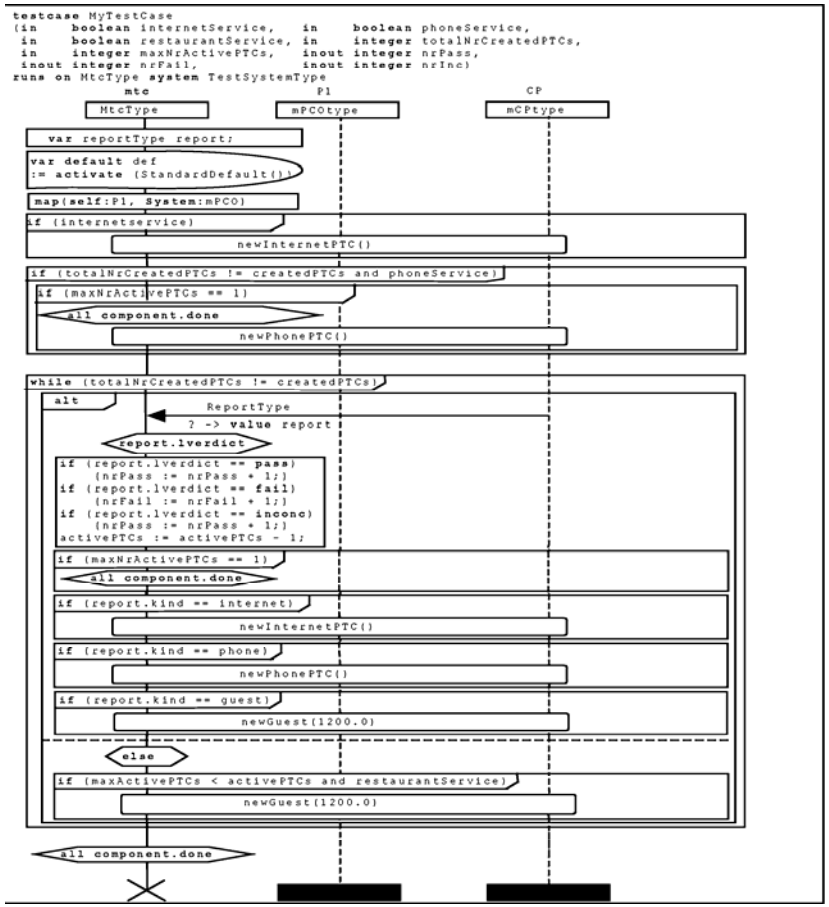
语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
清除并访问端口	**start**		将启动端口语句置于一个条件符号中。条件将只覆盖要启动的端口实例。
停止端口上的访问（接收&发送）	**stop**		将停止端口语句置于一个条件符号中。条件将只覆盖要停止的端口实例。
定时器操作			
启动定时器	**start**		
停止定时器	**stop**		
读取消逝的时间	**read**		非特殊的 GFT 符号，用在语句或表达式中。
检查定时器是否在运行	**running**		非特殊的 GFT 符号，用在语句或表达式中。
超时操作	**timeout**		
设置局部判定	**verdict.set**		将判定置于一个条件符号中。
获得局部判定	**verdict.get**		非特殊的 GFT 符号，用在语句或表达式中。

语 言 元 素	相关的 关键字	GFT符号（如果存在）以及典型用法	注　　释
SUT 操作			
SUT 将要完成的远程行为	**sut.action**		将行为语句置于一个行为框中。
测试用例执行			
执行测试用例	**execute**		将执行语句置于一个测试用例执行符号中。
属性			
控制、测试用例、测试步骤和函数的属性定义	**with**		将 with 语句置于一个文本符号中。
注释			
文本中的注释		/* 若干行注释 */ // 单行注释	可用于任何可放置文本的地方。
实例事件的注释			将附于有关控制、测试部件或端口实例的事件上。
注释控制、测试用例、函数或测试步骤图			将附于有关控制、测试部件或端口实例的事件上。

附 件 C

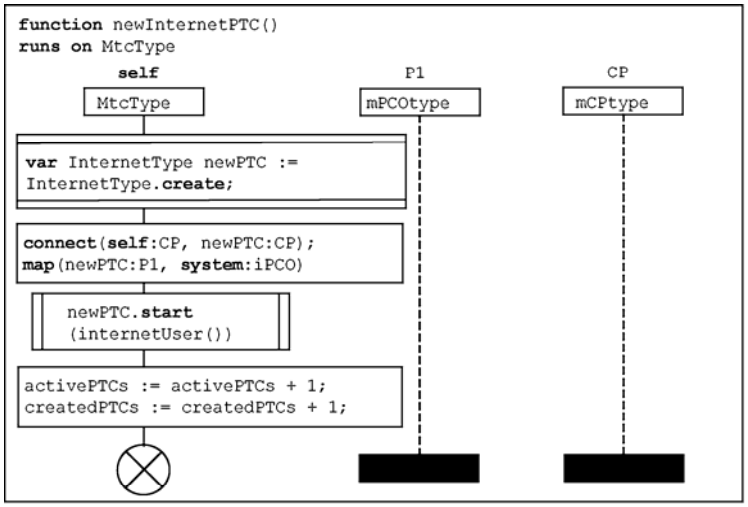
举 例

C.1 饭店例子

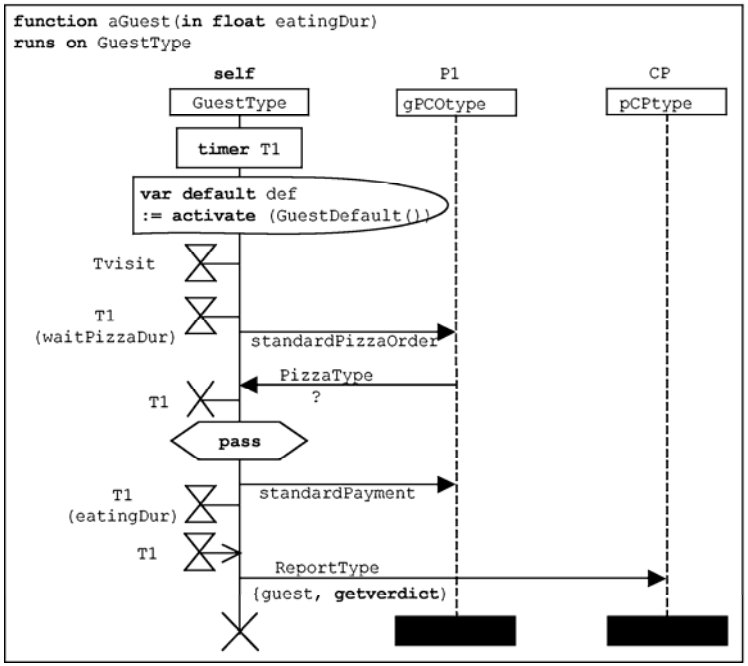



```
testcase MyTestCase (
    in boolean          internetService,     // 服务
    in boolean          phoneService,
    in boolean          restaurantService,
    in integer          totalNrCreatedPTCs,  // 终止
    in integer          maxNrActivePTCs,     // 控制
    inout integer       nrPass,              // 返回
    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;
}
```

图 C.1/Z.142－饭店例子 – MyTestCase测试用例

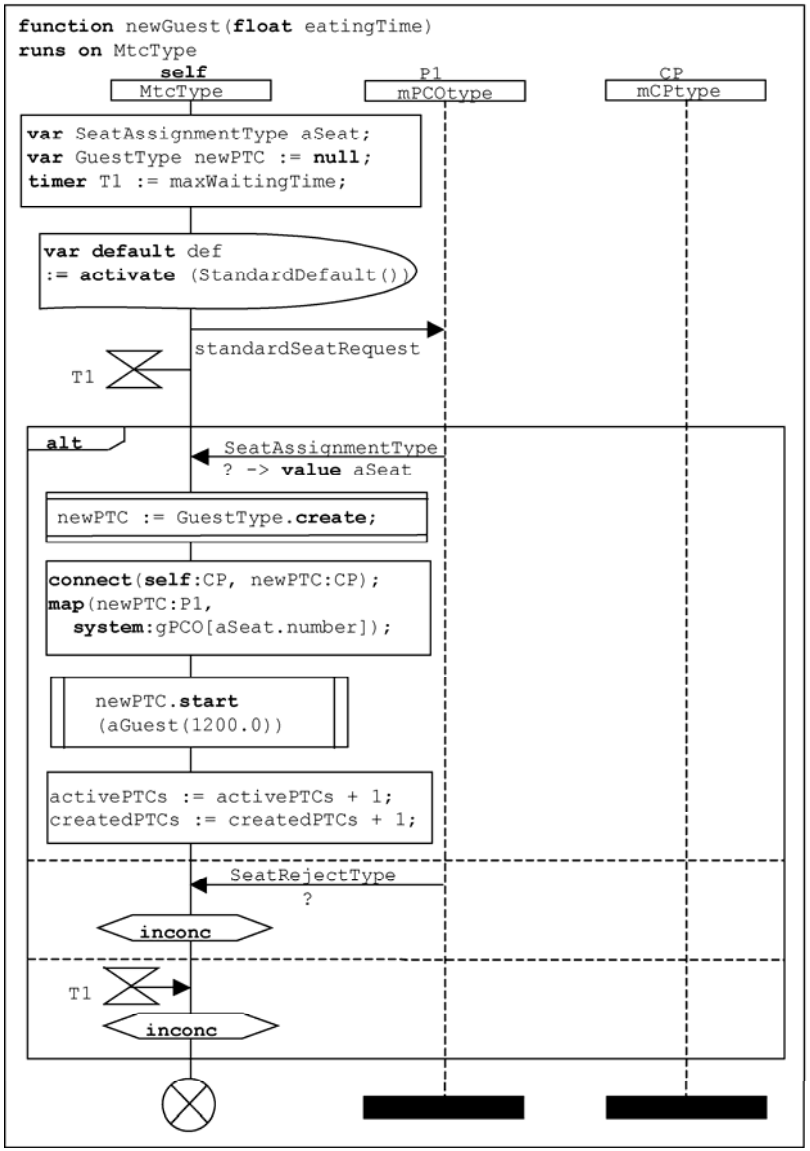



```
function newInternetPTC ()
   runs on MtcType {

   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;

}
```



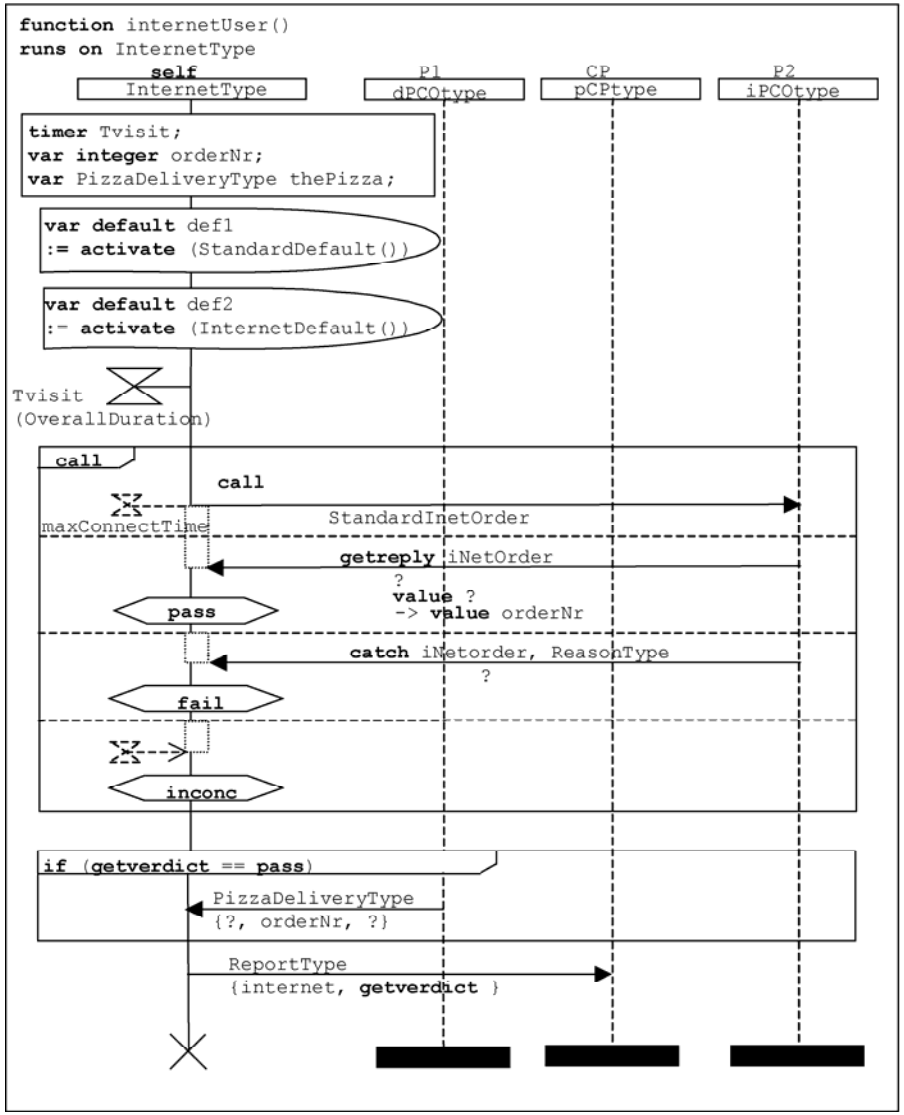

```
function aGuest (in float eatingDur) runs on GuestType {

   timer T1;

   var default def := activate(GuestDefault());
   Tvisit.start; // 部件定时器
   T1.start(waitPizzaDur);
   P1.send(standardPizzaOrder);
   P1.receive(PizzaType : ?);
   T1.stop;
   setverdict(pass);
   P1.send(standardPayment);
   T1.start(eatingDur);  // 吃饭
   T1.timeout;
   CP.send(ReportType : {guest, getverdict});
   stop;
 } // 结束函数aGuest
```

图 C.2/Z.142－饭店例子 – newInternetPTC 和 aGuest 函数



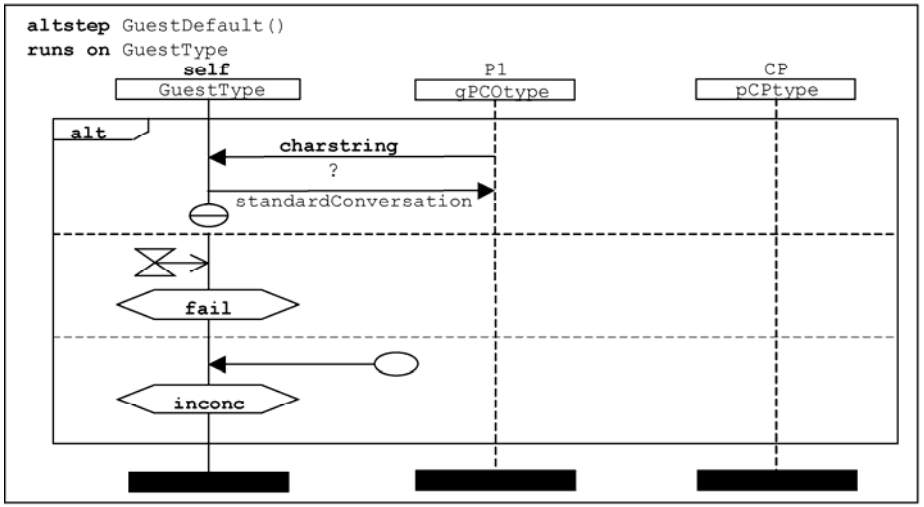

```
function newGuest (float eatingTime) runs on MtcType {

   var SeatAssignmentType aSeat;
   var GuestType newPTC := null;
   timer T1 := maxWaitingTime;

   var default def := activate(StandardDefault());

   // 请求一个座位
   P1.send(standardSeatRequest);
   T1.start;

   alt {
      [] P1.receive(SeatAssignmentType:?) -> value aSeat {
         newPTC := GuestType.create;

         connect(self:CP, newPTC:CP);
         map(newPTC:P1, system:gPCO[aSeat.number]);

         newPTC.start(aGuest(1200.0));

         activePTCs  := activePTCs+1; // 更新MTC变量
         createdPTCs := createdPTCs+1;
      }

      [] P1.receive(SeatRejectType:?) { // 没有指定座位
         setverdict(inconc);
      }

      [] T1.timeout { // 对请求没有答复
         setverdict(inconc);
      }
    }
    return;
}
```

图 C.3/Z.142－饭店例子– newGuest 函数

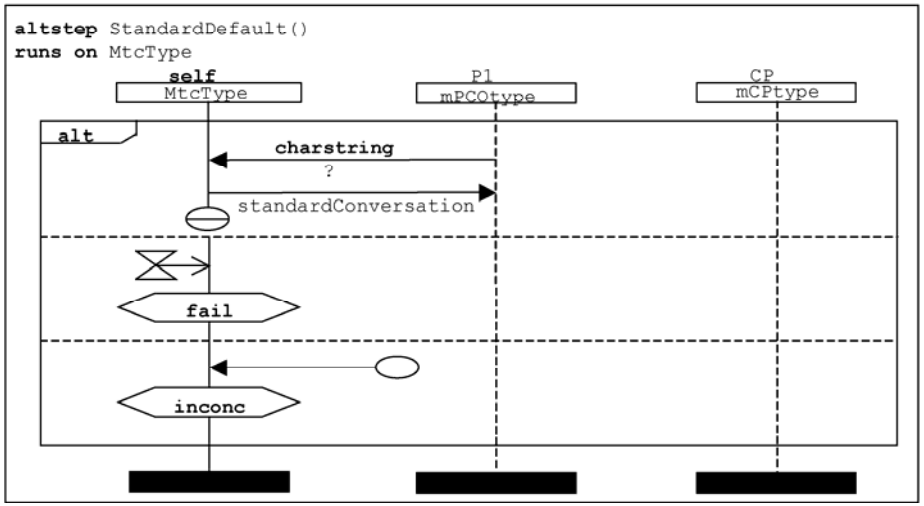



```
function internetUser () runs on InternetType {
  // ***
  // *** 目的：说明一个互联网客户的行为
  // ***

  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;
 }
```

图 C.4/Z.142－饭店例子– internetUser 函数

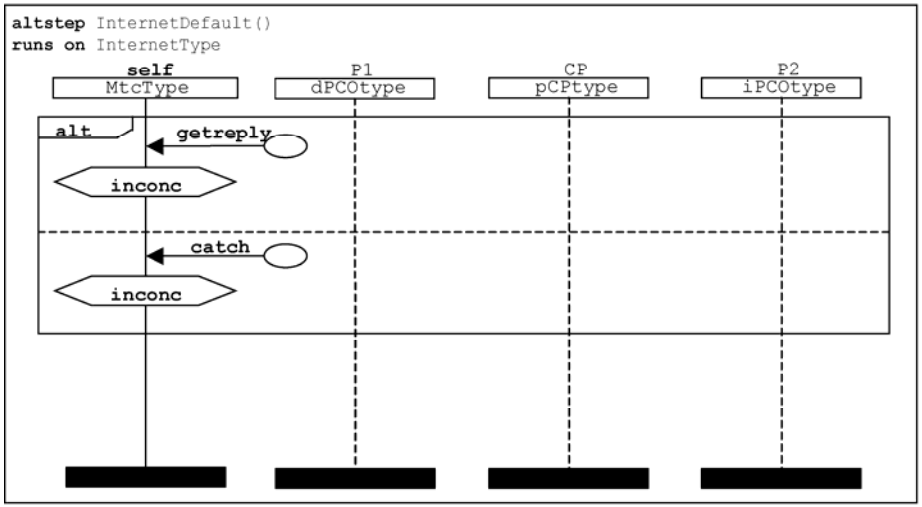



```
altstep GuestDefault() runs on GuestType {
   // ***
   // *** 目的：基于消息的端口的缺省行为
   // ***

   [] P1.receive(charstring : ?) {
      P1.send(standardConversation);
      repeat;
   }

   [] any timer.timeout {
      setverdict(fail);
   }

   [] any port.receive {
      setverdict(inconc);
   }
}
```

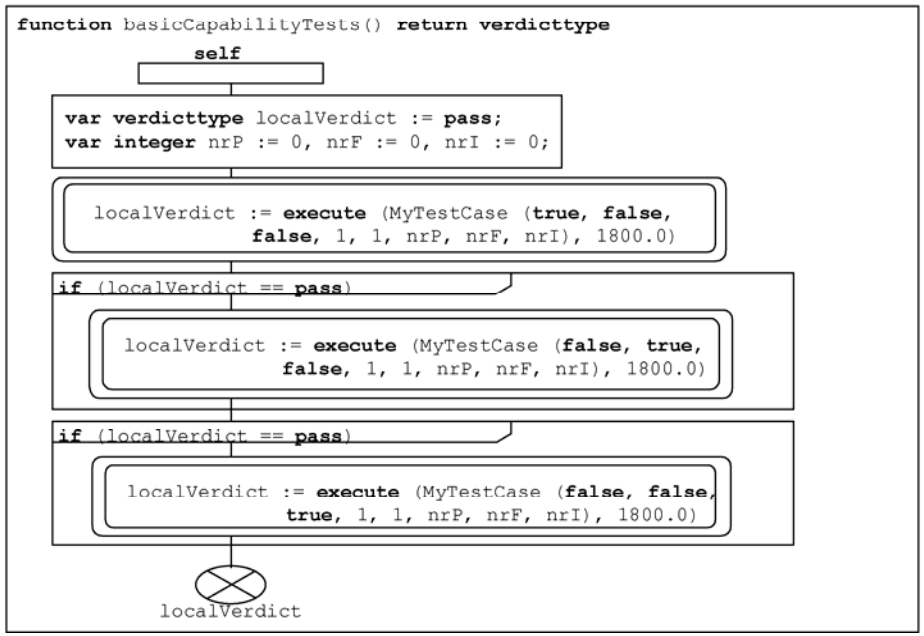



```
altstep StandardDefault() runs on MtcType {
   // ***
   // *** 目的：基于消息的端口的缺省行为
   // ***

   [] P1.receive(charstring : ?) {
      P1.send(standardConversation);
      repeat;
   }

   [] any timer.timeout {
      setverdict(fail);
   }

   [] any port.receive {
      setverdict(inconc);
   }
}
```

图 C.5/Z.142－饭店例子– GuestDefaut 和 StandardDefault 函数

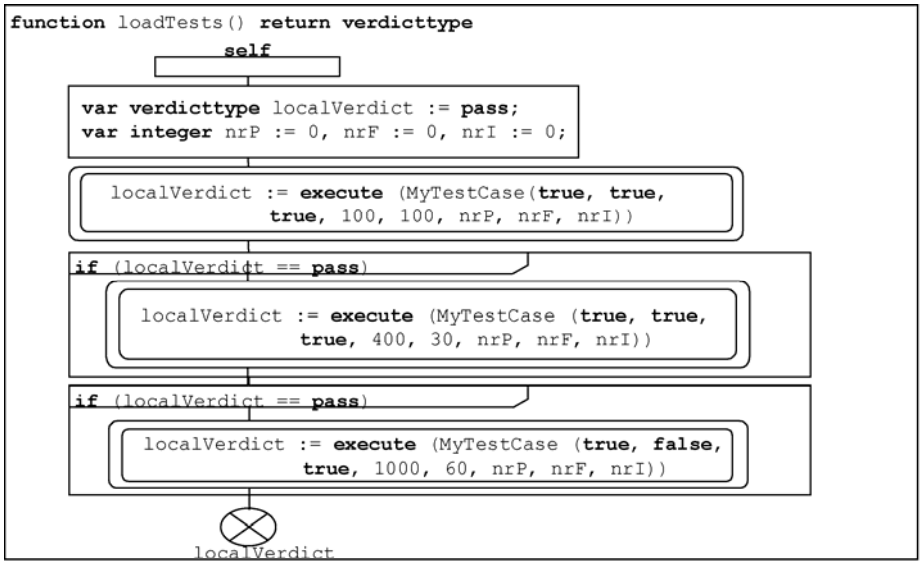



```
altstep InternetDefault()
   runs on InternetType {
   // ***
   // *** 目的：基于过程的端口的缺省行为
   // ***

   [] any port.getreply {
      setverdict(inconc);
   }

   [] any port.catch {
      setverdict(inconc);
   }
}
```

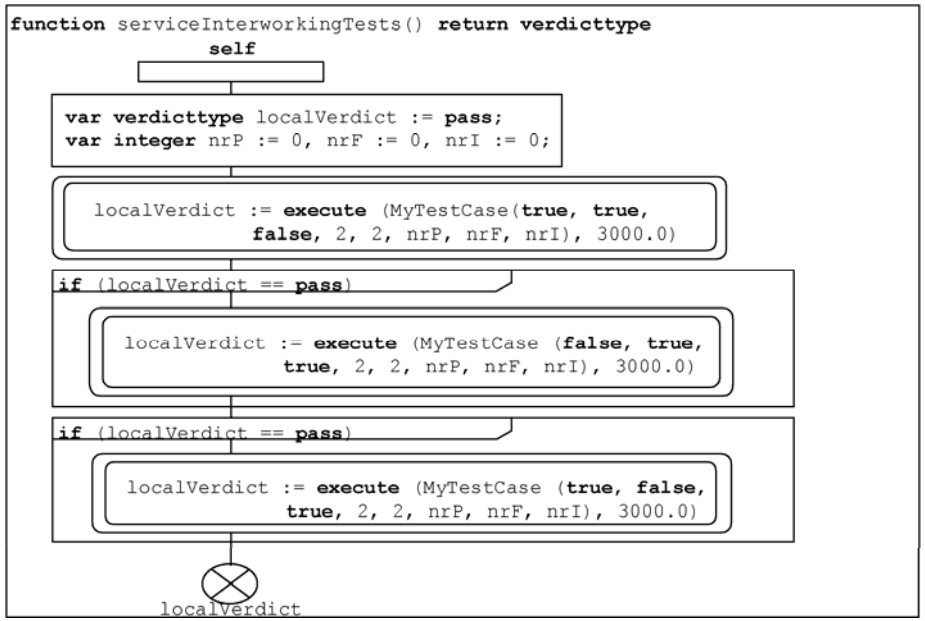



```
function basicCapabilityTests ()
   return verdicttype {
   var verdicttype localVerdict := pass;
   var integer nrP := 0, nrF := 0, nrI  := 0;

   // *** 互联网订购 ***
   localVerdict := execute(MyTestCase (true,false,
         false,1,1,nrP,nrF,nrI),1800.0);

   // *** 电话订购
   if (localVerdict == pass) {
      localVerdict := execute(MyTestCase
      (false,true,false,1,1,nrP,nrF,nrI),1800.0);
   }

   // *** 饭店订购 ***
   if (localVerdict == pass) {
      localVerdict := execute(MyTestCase
      (false,false,true,1,1,nrP,nrF,nrI),1800.0);
   }
   return (localVerdict);
}
```

图 C.6/Z.142－饭店例子– internetDefault 可选步骤和 basicCapabilityTests 函数

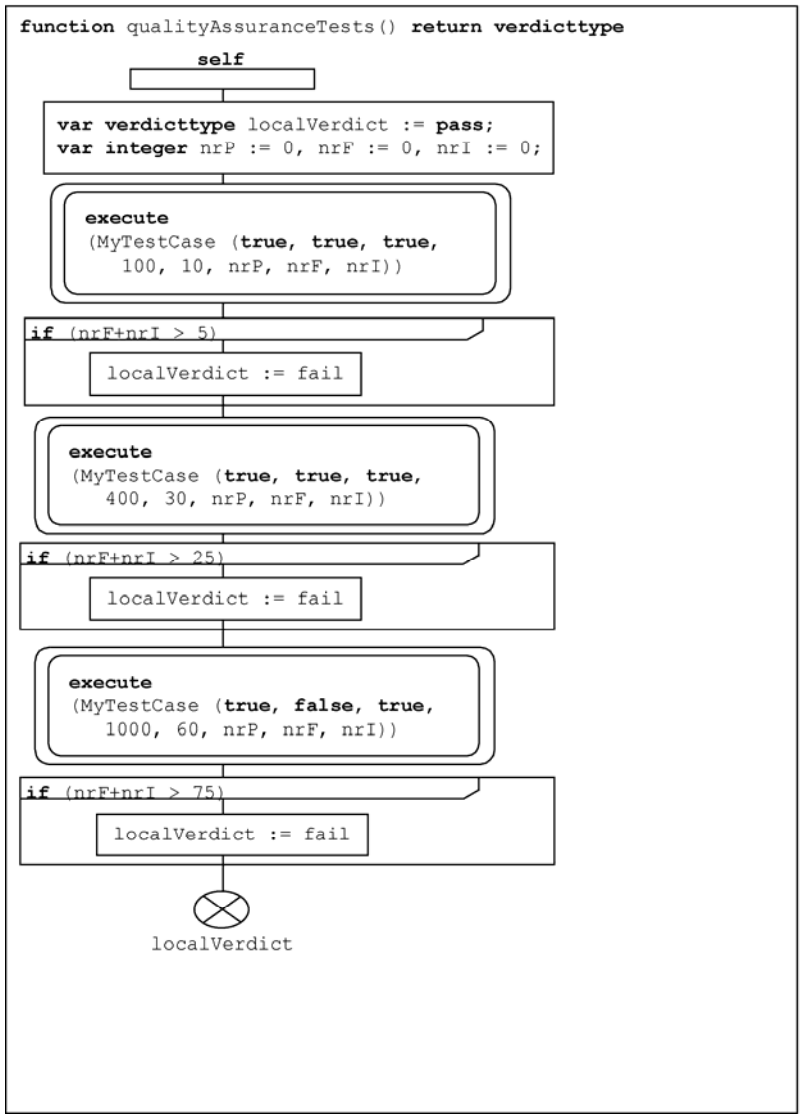



```
function loadTests () return verdicttype {
   var verdicttype localVerdict := pass;
   var integer nrP := 0, nrF := 0, nrI  := 0;

   // *** 最小负载 ***
   localVerdict := execute(MyTestCase(
         true,true,true,100,10,nrP,nrF,nrI));

   // *** 中等负载 ***
   if (localVerdict == pass) {
      localVerdict := execute(MyTestCase(
         true,true,true,400,30,nrP,nrF,nrI));
    }

   // *** 最大负载 ***
   if (localVerdict == pass) {
      localVerdict := execute(MyTestCase(
         true,false,true,1000,60,nrP,nrF,nrI));
    }
    return (localVerdict);
}
```



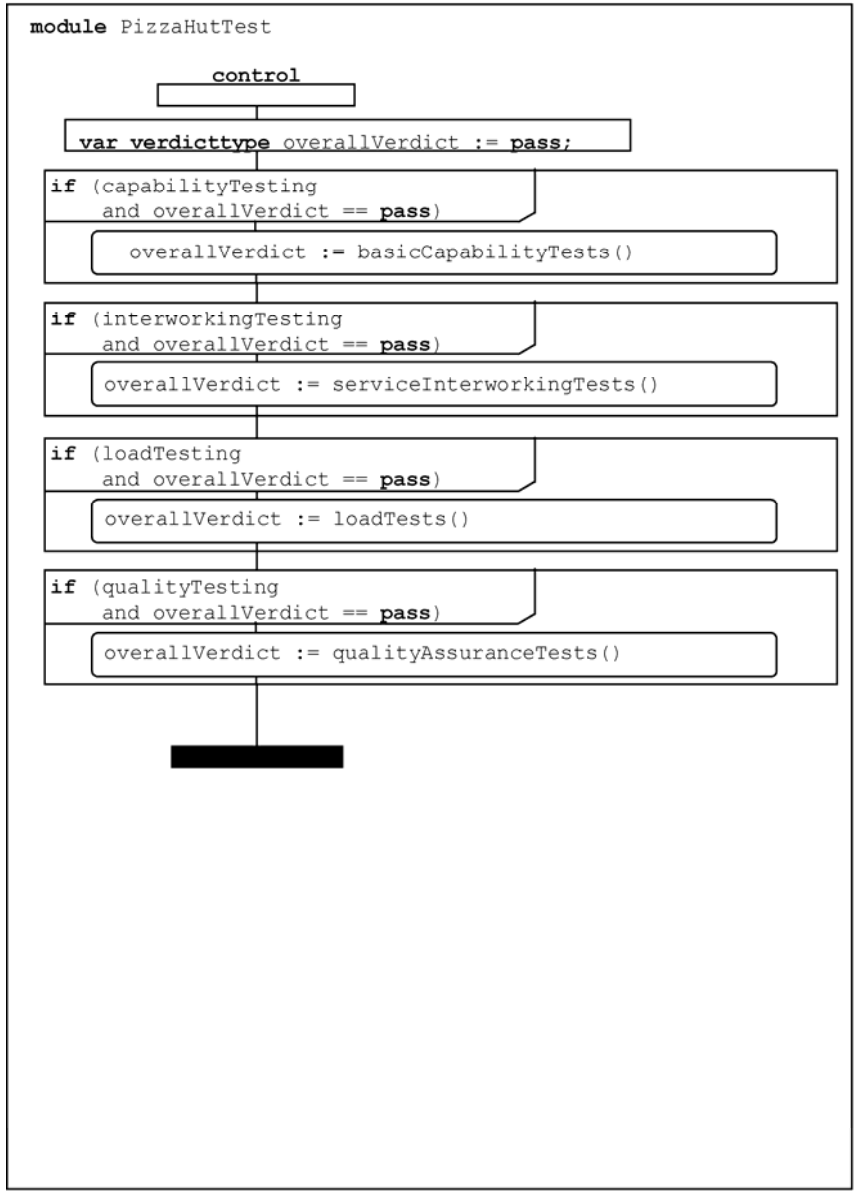

```
function serviceInterworkingTests ()
   return verdicttype {
   var verdicttype localVerdict := pass;
   var integer nrP := 0, nrF := 0, nrI  := 0;

   // *** 互联网订购与电话订购 ***
   localVerdict := execute(MyTestCase(
      true,true,false,2,2,nrP,nrF,nrI),3000.0);

   // *** 电话订购与饭店订购
   if (localVerdict == pass) {
      localVerdict := execute(MyTestCase(
      false,true,true,2,2,nrP,nrF,nrI),3000.0);
   }

   // *** 饭店订购与互联网订购 ***
   if (localVerdict == pass) {
      localVerdict := execute(MyTestCase(
      true,false,true,2,2,nrP,nrF,nrI),3000.0);
   }

   return (localVerdict);
}
```

图 C.7/Z.142－饭店例子– loadTests 和 serviceInterworkingTests 函数




```
function qualityAssuranceTests ()
   return verdicttype {

   var verdicttype localVerdict := pass;
   var integer nrP := 0,
               nrF := 0,
               nrI  := 0;

   // *** 最小负载下的质量 ***
   execute(MyTestCase(true,true,true,100,10,
         nrP,nrF,nrI));

   if ( nrF + nrI > 5 ) {
      localVerdict := fail;
   }

   // *** 中等负载下的质量 ***
   execute(MyTestCase(true,true,true,400,30,
         nrP,nrF,nrI));

   if ( nrF + nrI > 25 ) {
      localVerdict := fail;
   }

   // *** 最大负载下的质量 ***
   execute(MyTestCase(true,false,true,1000,60,
         nrP,nrF,nrI));

   if ( nrF + nrI > 75 ) {
      localVerdict := fail;
   }

   return (localVerdict);

} // 结束函数 qualityAssuranceTests
```

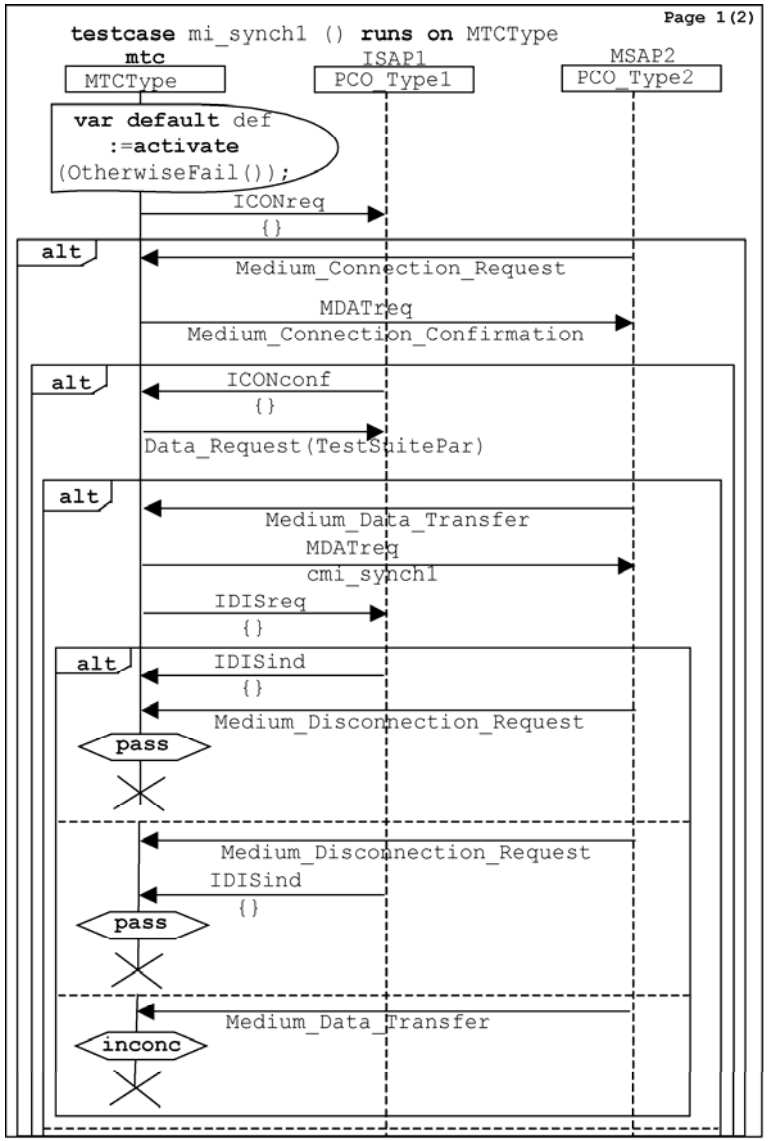
图 C.8/Z.142－饭店例子– qualityAssuranceTests




```
module PizzaHutTest (
      boolean capabilityTesting,
      boolean interworkingTesting,
      boolean loadTesting,
      boolean qualityTesting ) {

   control {

   var verdicttype overallVerdict := pass;

   // 基本性能测试
   if (capabilityTesting and overallVerdict == pass) {
      overallVerdict := basicCapabilityTests();
   }

   // 联网测试
   if (interworkingTesting and overallVerdict == pass) {
      overallVerdict := serviceInterworkingTests();
   }

   // 负载测试
   if (loadTesting and overallVerdict == pass) {
      overallVerdict := loadTests();
   }

   // 质量保证测试
   if (qualityTesting and overallVerdict == pass) {
      overallVerdict := qualityAssuranceTests();
   }

}
```

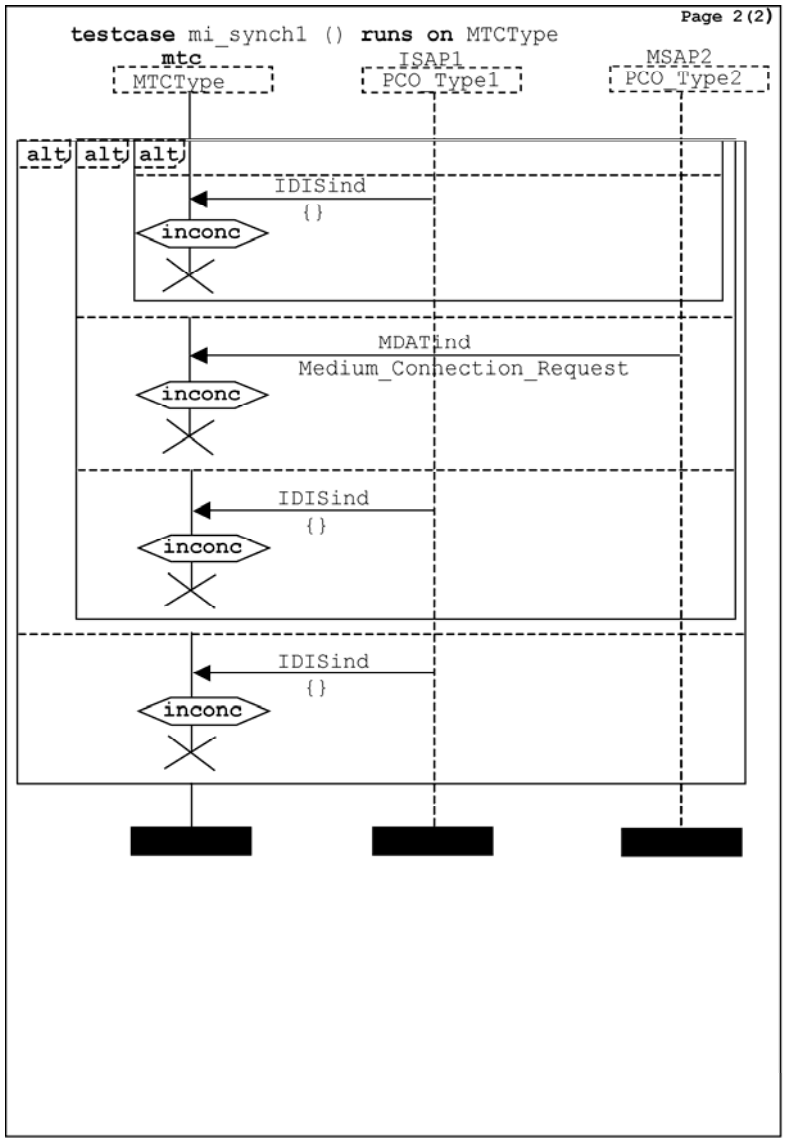
图 C.9/Z.142－饭店例子– PizzaHutTest 模块

C.2 INRES例子



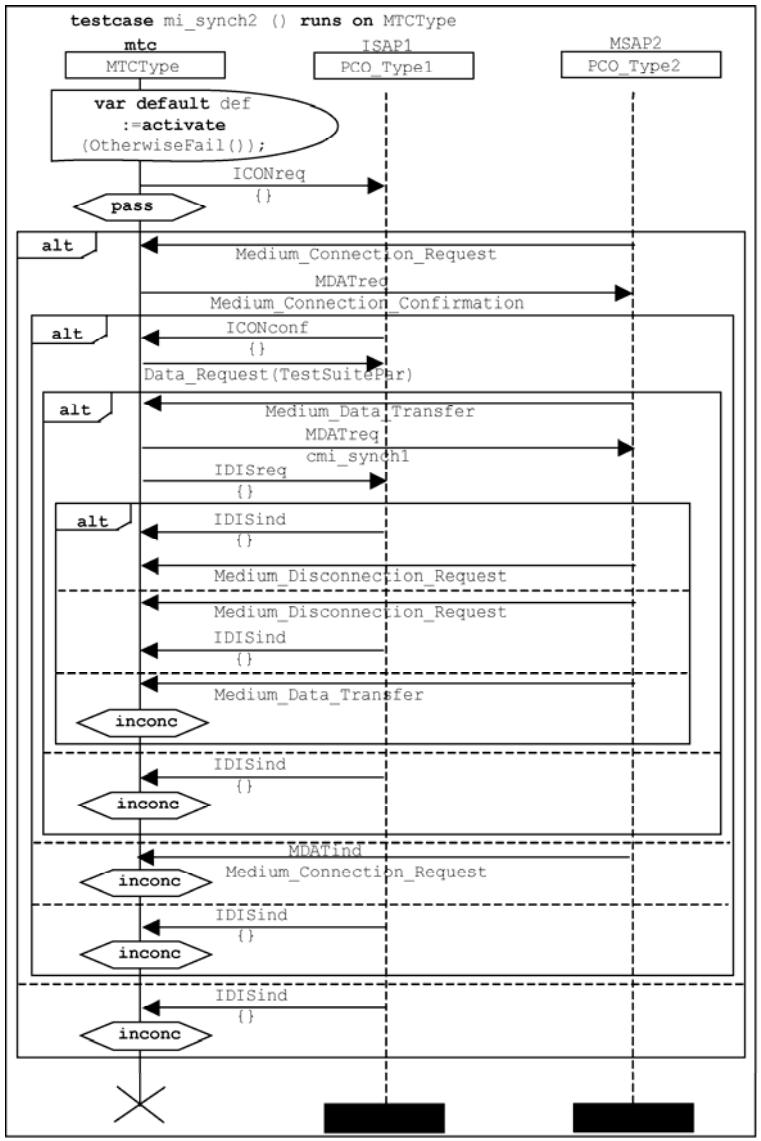

```
testcase mi_synch1 () runs on MTCType {

   /* 缺省激活 */
   var default def := activate(OtherwiseFail());

   /* 内嵌模板定义 */
   ISAP1.send( ICONreq:{} );

   alt { /* alt1 */
      [] MSAP2.receive( Medium_Connection_Request ) {
                          /* 模板使用 */
         MSAP2.send( MDATreq:Medium_Connection_Confirmation );
                          /* 可选的模板类型 */

      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;
                     }
                  } /* 结束alt4 */
               }
```

图 C.10/Z.142－INRES 例子 – mi_synch1 1(2) 测试用例



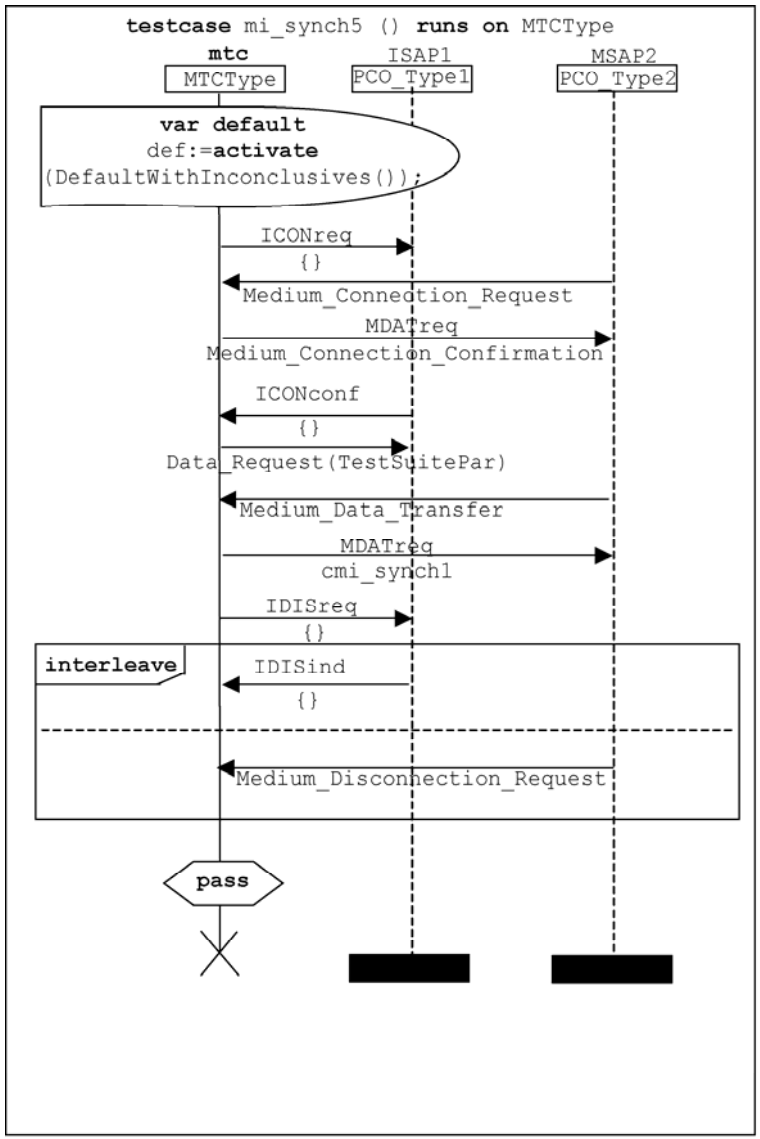

```
/* 测试用例 mi_synch1()继续 */

                [] ISAP1.receive( IDISind:{} ) {
                   setverdict(inconclusive);
                   stop;
                }
             } /* 结束 alt3 */
          }

          [] MSAP2.receive(
                   MDATind:Medium_Connection_Request) {
             setverdict(inconclusive);
          stop;
          }

          [] ISAP1.receive( IDISind:{} ) {
          setverdict(inconclusive);
          stop;
          }
       }
    } /* 结束 alt2 */


[] ISAP1.receive( IDISind:{} ) {
      setverdict(inconclusive);
      stop;
    }
   } /* 结束 alt1 */
 } /* 结束测试用例 mi_synch1 */
```

图 C.11/Z.142－INRES 例子– mi_synch1 2(2) 测试用例



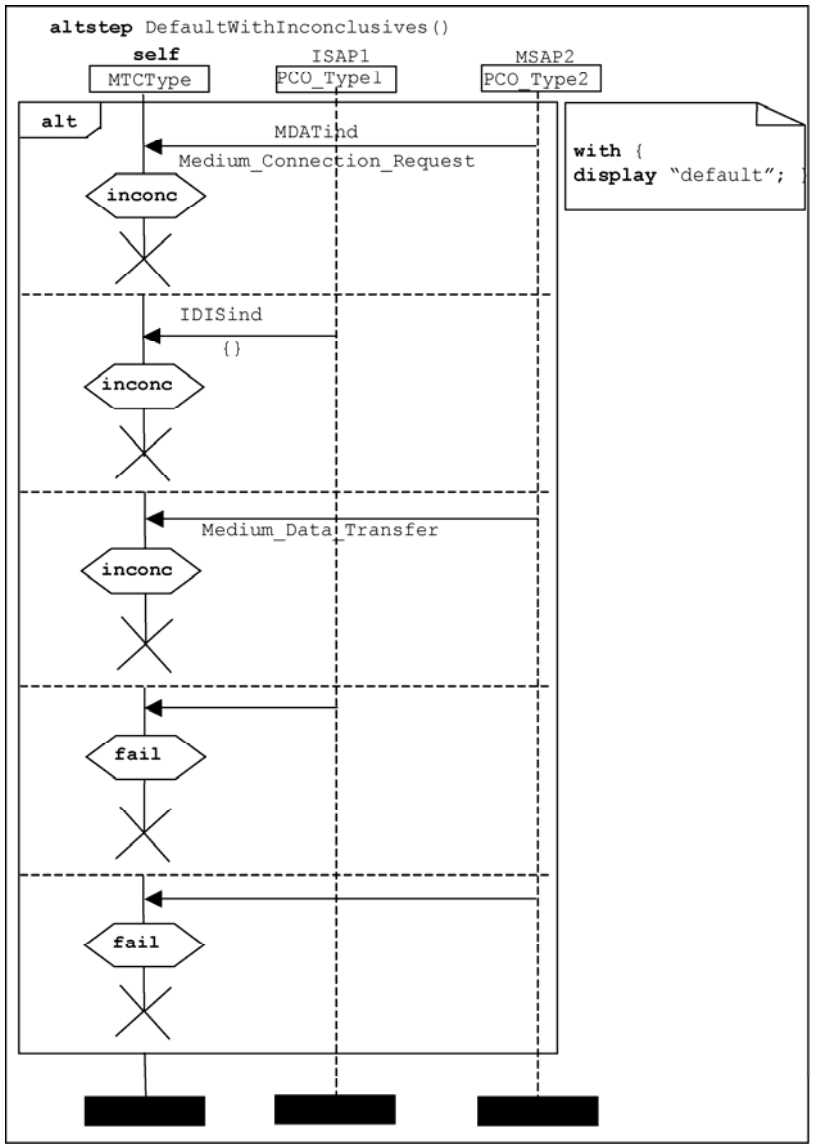

```
testcase mi_synch2 () runs on MTCType {


   var default def := activate(OtherwiseFail());
                      /* 缺省激活 */

   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:{} ) { /* 通过 */
                           MSAP2.receive(
                              Medium_Disconnection_Request );
                        }
                        [] MSAP2.receive(
                              Medium_Disconnection_Request ){
                           ISAP1.receive( IDISind:{} ); /* 通过 */
                        }

                        [] 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; } /* 结束测试用例 mi_synch2 */
```

图 C.12/Z.142－INRES 例子– mi_synch2 测试用例



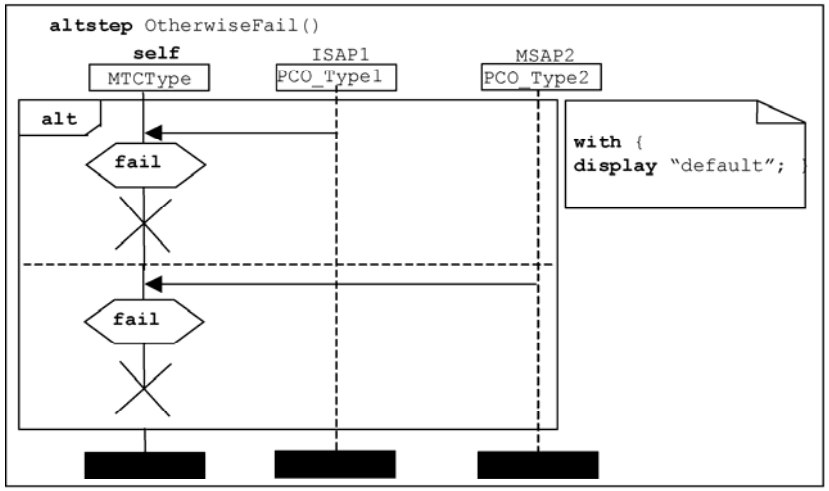

```
testcase mi_synch5 () runs on MTCType {



  var default
       def := activate(DefaultWithInconclusives );
                        /* 缺省激活 */
  /* 消息1以及消息1的响应 */
  ISAP1.send( ICONreq:{} );
  MSAP2.receive(Medium_Connection_Request );

  /* 消息2以及消息2的响应 */
  MSAP2.send(
          MDATreq:Medium_Connection_Confirmation );
  ISAP1.receive ( ICONconf:{} );

  /* 消息3以及消息3的响应 */
  ISAP1.send ( Data_Request(TestSuitePar) );
  MSAP2.receive ( Medium_Data_Transfer );

  /* 消息4和消息5 */
  MSAP2.send ( MDATreq:cmi_synch1 );
  ISAP1.send ( IDISreq:{} );

  interleave {
     /* 消息4和消息5的两个响应可以以任何次序到达 */
     [] ISAP1.receive(IDISind:{}) {};
     [] MSAP2.receive(
           Medium_Disconnection_Request ) {};
   }

   setverdict(pass);

   stop;

} /* 结束测试用例 mi_synch5 */
```

图 C.13/Z.142－INRES 例子– mi_synch5 测试用例

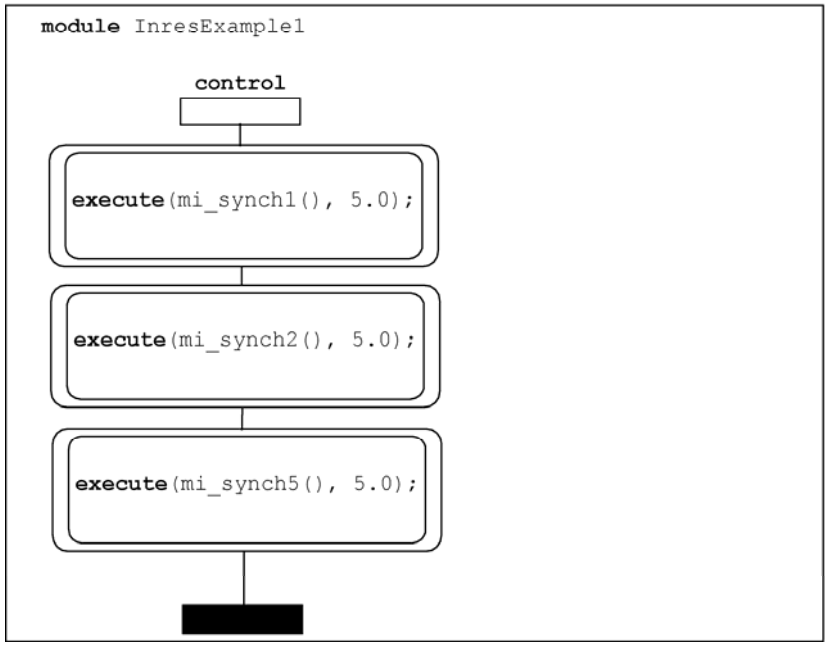



```
altstep DefaultWithInconclusives() {

    /* 不确定的情况 */

   [] MSAP2.receive( MDATind:Medium_Connection_Request) {

      setverdict(inconclusive);
      stop;
   }


   [] ISAP1.receive ( IDISind:{} ) {

      setverdict(inconclusive);
      stop;
   }



   [] MSAP2.receive ( Medium_Data_Transfer ) {

      setverdict(inconclusive);
      stop;
   }


  /* 失败的情况 */

   [] ISAP1.receive {

      setverdict(fail);
      stop;

   }


   [] MSAP2.receive {

      setverdict(fail);
      stop;
   }


} with { display "default"; }
```

图 C.14/Z.142－INRES 例子– DefaultWithInconclusives 可选步骤




```
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);

   } // 结束控制部分

}
```

图 C.15/Z.142－INRES 例子– OtherwiseFail可选步骤和InresExample1模块定义

ITU-T 系列建议书

A系列	ITU-T工作的组织
D系列	一般资费原则
E系列	综合网络运行、电话业务、业务运行和人为因素
F系列	非话电信业务
G系列	传输系统和媒质、数字系统和网络
H系列	视听及多媒体系统
I系列	综合业务数字网
J系列	有线网络和电视、声音节目及其它多媒体信号的传输
K系列	干扰的防护
L系列	电缆和外部设备其它组件的结构、安装和保护
M系列	电信管理，包括TMN和网络维护
N系列	维护：国际声音节目和电视传输电路
O系列	测量设备的技术规范
P系列	电话传输质量、电话设施及本地线路网络
Q系列	交换和信令
R系列	电报传输
S系列	电报业务终端设备
T系列	远程信息处理业务的终端设备
U系列	电报交换
V系列	电话网上的数据通信
X系列	数据网、开放系统通信和安全性
Y系列	全球信息基础设施、互联网协议问题和下一代网络
Z系列	**用于电信系统的语言和一般软件问题**

瑞士印刷
2006年，日内瓦