国 际 电 信 联 盟

ITU-T Z.144

国际电信联盟
电信标准化部门

(03/2006)

Z系列：用于电信系统的语言和一般软件问题

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3版（TTCN-3）：运行时间接口（TRI）

ITU-T Z.144建议书

国际电信联盟

ITU-T Z系列建议书

用于电信系统的语言和一般软件问题

正式描述技巧（FDT）	
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
基本句法和对话程序	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	
分布式处理环境	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.144建议书

测试和测试控制记法第3版（TTCN-3）：运行时间接口（TRI）

摘　要

本建议书提供了 TTCN-3（*测试和测试控制记法第 3 版*）测试系统实现的运行时间接口规范。TTCN-3 运行时间接口为一个测试系统的定时和通信提供了分别对特殊处理平台和被测系统的建议适配。本建议书将接口定义为与目标语言无关的操作集。

接口被定义为遵循 ITU-T Z.140 建议书。这一建议书完全使用 CORBA 接口定义语言(IDL)来规定 TRI。第 6 和 7 节说明了抽象规范到目标语言 Java 和 ANSI-C 的语言映射。基于 IDL 的接口规范的摘要在附件 A 中提供。

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.144 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

本建议书包含两个不同的部分，第一部分描述 TTCN-3 测试系统实现的结构，第二部分阐述 TTCN-3 运行时间接口规范。

第一部分将 TTCN-3 测试系统分成 4 个主要的实体：

- 测试管理实体（TM）；
- TTCN-3 执行实体（TE）；
- SUT 适配实体（SA）；和
- 平台适配实体（PA）。

另外，定义了这些实体之间的交互，也即对应的接口。

本建议书的第二部分规定了 TTCN-3 运行时间接口（TRI）。接口按照操作定义，这些操作作为一个实体的部分实现，并由测试系统的其他实体调用。对于每个操作，接口规范定义了相关的数据结构、预计对测试系统的影响以及对操作使用的约束。注意接口规范仅定义 TSI 和 SUT 以及定时器操作之间的交互。

ITU-T Z.144建议书

测试和测试控制记法第3版（TTCN-3）：运行时间接口（TRI）

1 范围

本建议书提供了 TTCN-3 测试系统实现的运行时间接口规范。TTCN-3 运行时间接口为一个测试系统的定时和通信提供了分别对特殊处理平台和被测系统的标准化适配。本建议书将接口定义为与目标语言无关的操作集。

接口被定义为遵循 TTCN-3 标准（见以下参考文献）。这一建议书完全使用 CORBA 接口定义语言（IDL）来规定 TRI。第 6 和 7 节说明了抽象规范到目标语言 Java 和 ANSI-C 的语言映射。基于 IDL 的接口规范的摘要在附件 A 中提供。

1.1 依从性

要求 TTCN-3 测试系统依从 TRI，要坚持本建议书中规定的接口规范以及所包含的目标语言映射之一。

例子：如果一个供货商支持 Java，则 TRI 操作调用和实现（它们是 TTCN-3 执行实体的一部分）务必遵循本建议书中规定的 IDL 到 Java 的映射。

2 参考文献

下列 ITU-T 建议书和其他参考文献的条款，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 X.290 (1995), *OSI conformance testing methodology and framework for protocol Recommendations for ITU-T applications – General concepts*.
ISO/IEC 9646-1:1994, *Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 1: General concepts*.

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

[3] 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)*.

[4] ITU-T Recommendation Z.143 (2006), *Testing and Test Control Notation version 3 (TTCN-3): Operational Semantics*.

3 定义和缩略语

3.1 定义

就本建议书而言，在 ITU-T Z.140 建议书[2]中给出的和下列的术语和定义适用：

3.1.1 abstract test suite (ATS) 抽象测试套（ATS）：见 ITU-T X.290 建议书[1]。

3.1.2 communication port 通信端口：便于测试组件之间通信的抽象装置。

注 — 在接收方向，通信端口被建模为一个 FIFO 队列。端口可以是基于消息的、基于过程的或者是这二者的混合。

3.1.3 executable test suite (ETS) 可执行测试套（ETC）：见 ITU-T X.290 建议书[1]。

3.1.4 explicit timer 显式定时器：TTCN-3 ATS 中声明的定时器，它可通过 TTCN-3 定时器操作访问。

3.1.5 implementation extra information for testing (IXIT) 测试实现附加信息（IXIT）：见 ITU-T X.290 建议书[1]。

3.1.6 implicit timer 隐式定时器：由 TTCN-3 执行实体创建用来保护 TTCN-3 调用或执行操作的系统定时器。

注 — TTCN-3 用户不能访问隐式定时器。

3.1.7 platform adapter (PA) 平台适配实体：使 TTCN-3 执行实体适配于特殊执行平台的实体。

注 — 平台适配实体为测试系统创建了一个单独的时间符号、实现外部函数以及显式和隐式定时器。

3.1.8 SUT adapter (SA) SUT 适配实体：适配于具有基于抽象测试系统接口的 SUT 的 TTCN-3 的通信操作并实现实际测试系统接口的实体。

3.1.9 system under test (SUT) 被测系统（SUT）：见 ITU-T X.290 建议书[1]。

注 — 所有类型都在编译时间已知，即在静态范围内。

3.1.10 test case 测试案例：见 ITU-T X.290 建议书[1]。

3.1.11 test event 测试事件：在一个通信端口（测试系统接口的一部分）上的发送或接收的测试数据（消息或过程调用）。

3.1.12 test management (TM) 测试管理：提供用户接口和管理 TTCN-3 测试系统的实体。

3.1.13 test system 测试系统：见 ITU-T X.290 建议书[1]。

3.1.14 test system interface 测试系统接口：系统的组成部分，它提供从（抽象的）TTCN-3 测试系统中可用的端口到由实际测试系统提供的那些可用端口的映射。

3.1.15 timer identification (TID) 定时器标识：由 TTCN-3 执行实体生成的显式或隐式定时器实例的惟一标识。

3.1.16 TTCN-3 control interface (TCI) TTCN-3 控制接口（TCI）：通常在一个测试系统中规定测试管理和执行实体之间交互的专用接口。

3.1.17 TTCN-3 executable (TE) TTCN-3 执行实体（TE）：测试系统的一部分，它处理 TTCN-3 ETS 的解释或执行。

3.1.18 TTCN-3 runtime interface (TRI) TTCN-3 运行时间接口（TRI）：在一个测试系统中定义 TTCN-3 执行实体与 SUT 以及平台适配器之间的交互的接口。

3.2 缩略语

本建议书使用下列缩略语：

ATS	抽象测试套
CH	组件处理器
ECD	外部编解码器
EDS	编码/解码系统
ETS	可执行测试套
IDL	接口定义语言
IXIT	测试实现附加信息
MSC	消息序列图
MTC	主测试组件
OMG	对象管理组
PA	平台适配实体
SA	SUT 适配实体
SUT	被测系统
T3RTS	TTCN-3 运行时间系统
TC	测试控制
TCI	TTCN-3 控制接口

TE	TTCN-3 执行实体
TID	定时器标识
TL	测试记录
TM	测试管理
TRI	TTCN-3 运行时间接口
TSI	测试系统接口
TTCN	测试和测试控制记法
TTCN-3	树表组合符号第 3 版

4 TTCN-3测试系统的一般结构

TTCN-3 测试系统在概念上可以被认为是交互实体集，其中每个实体对应于测试系统实现内的功能性的一个特定方面。这些实体管理测试执行、解释或执行编译的 TTCN-3 代码、实现与 SUT 的正确通信并处理定时器操作。（见图 1）。

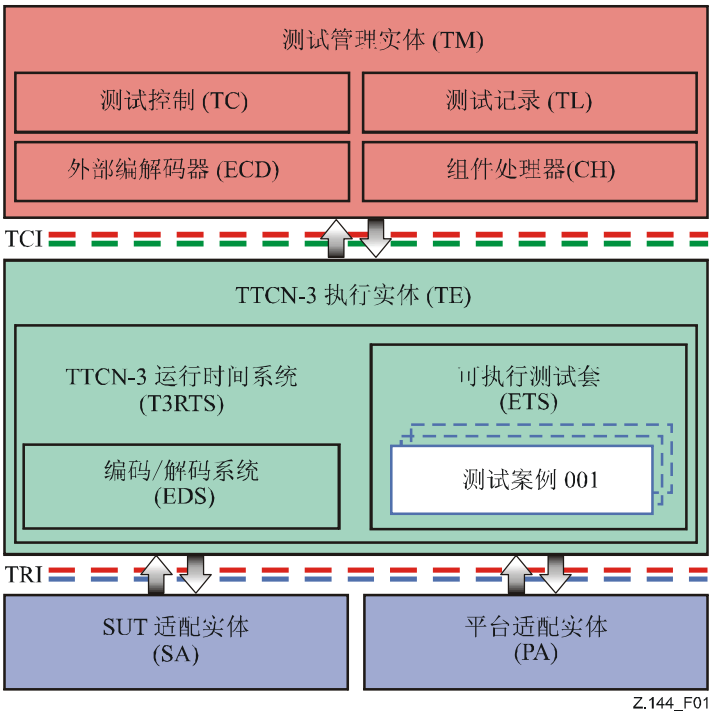

图 1/Z.144－TTCN-3 测试系统的一般结构

4.1 TTCN-3 测试系统中的实体

TTCN-3 测试系统实现的结构在图 1 中阐述。应注意，如图 1 所示并在本建议书的下列各节中使用的，进一步将 TM 细分为更小的实体，纯粹是为了帮助定义 TTCN-3 测试系统接口。

处理 TTCN-3 模块解释和执行的测试系统的部分，即可执行测试套（ETS），是 TTCN-3 执行实体的一部分。这对应于由测试系统实现中的 TTCN-3 编译程序或 TTCN-3 解释程序生成的可执行编码。假定测试系统实现包括从 TTCN-3 ATS 中得到的 ETS。

TTCN-3 测试系统的剩余部分（处理不能包括的、来自在最初的 ATS 中单独存在的信息的任何方面）可被分解成测试管理实体（TM）、SUT 适配实体（SA）和平台适配实体（PA）。通常，这些实体包含一个测试系统用户接口、测试执行控制、测试事件记录以及与 SUT 的通信和定时器实现。

4.1.1 测试管理实体（TM）

在 TM 实体中，我们可以区分与测试执行控制和测试事件记录相关的功能性。

4.1.1.1 测试控制（TC）

TC 实体负责测试系统的整个管理。在测试系统已经初始化之后，测试执行在 TC 实体内开始。实体负责 TTCN-3 模块的正确调用，即如果必需，传播模块参数和/或 IXIT 信息给 TE。典型地，这一实体也实现测试系统用户接口。

4.1.1.2 测试记录（TL）

TL 实体负责维护测试记录。由 TE 明确地通报记录测试事件。TL 实体具有一个单向接口，其中 TE 的任何实体部分可发送一个记录请求给 TL 实体。TM 内部接口也可用于记录由 TC 生成的测试管理信息。

4.1.1.3 外部编解码器（ECD）

可选地，外部编解码器实体负责编码和解码在 TE 内与基于消息或基于过程的通信相关联的数据。外部编解码器可和与 TE 相关联的内部编解码器一起使用，或替代其使用。与内部编解码器不同，外部编解码器具有标准的接口，这使其能在不同的 TTCN-3 系统和工具之间可携带。

4.1.1.4 组件处理器（CH）

CH 实体负责分配类似的测试组件。这一分配可能经过一个或多个物理系统。CH 实体允许测试管理实体以透明和独立于 TE 的方式创建和控制分配的测试系统。

4.1.2 TTCN-3执行实体（TE）

TE 实体负责 TTCN-3 ATS 的解释或执行。概念上，TE 可分解成 3 个交互的实体：ETS、TTCN-3 运行时间系统（T3RTS）和编码/解码系统（EDS）实体。注意将 TM 细分为更小的实体，纯粹是为了帮助定义 TTCN-3 测试系统接口 — 对于此细分，不需要反映在 TRI 实现中。

下列各节定义了每个实体的职责，也讨论了 TRI 中定时器的处理。

4.1.2.1 可执行测试套（ETS）

ETS 实体处理测试案例的执行或解释以及测试事件的排序和匹配，如在相应 TTCN-3 模块 ITU-T Z.140 建议书[2]中定义的。它与 T3RTS 实体交互，在测试案例执行期间发送、意图接收（或匹配）和记录测试事件，创建和移除 TTCN-3 测试组件以及处理外部函数调用、动作操作和定时器。注意 ETS 实体不直接经由 TRI 与 SA 交互。

4.1.2.2 TTCN-3 运行时间系统（T3RTS）

T3RTS 实体与 TM、SA 和 PA 实体经由 TCI 和 TRI 交互，并管理 ETS 和 EDS 实体。T3RTS 初始化适配器和 ETS 以及 EDS 实体。这一实体执行在 ETS 实体中正确开始测试案例或具有参数的函数的执行所必需的所有动作。它向 TA 实体询问 ETS 所需的模块参数值，并发送记录信息给它。它也收集与 ETS 实体返回的结论相关的信息，如 ITU-T Z.140 建议书[2]中定义的。

T3RTS 实体实现 TTCN-3 测试组件、TTCN-3 基于消息和基于过程的通信的语义、外部函数调用、动作操作和定时器的创建和移除。这包括向 SUT 适配实体（SA）通报将发送哪一消息或过程调用给 SUT，或向平台适配器（PA）通报 即将执行哪个外部函数或将启动、停止、询问或读取哪个定时器。类似地，T3RTS 通知 ETS 实体来自 SUT 的入网消息或过程调用以及超时事件。

在向 SA 发送或从 SA 接收消息和过程调用或在 PA 中为 ETS 实体处理函数调用和动作操作之前，T3RTS 调用 EDS 实体来进行编解码。T3RTS 实体应实现测试组件之间所有基于消息和基于过程的通信操作，而且也应实现与 SUT 基于过程的通信的 TTCN-3 语义，即测试组件执行的可能的模块化和去模块化、保护隐式定时器、处理这样的通信操作所造成的超时异常。所有与 SUT 的基于过程的通信操作将在 SA 中实现和标

识（在接收操作的情况下），因为它们在特定平台的方式中能最有效地实现。注意任何过程调用操作的定时器，即隐式定时器，在平台适配实体（PA）中实现。

要求 TTCN-3 执行实体对于输入测试事件维持其自己的端口队列（区别于那些可能在 SA 或 PA 中可获得的队列）来执行接收操作的快照，如 ITU-T Z.140 建议书[2]中定义的。由 TTCN-3 定时器、调用定时器或测试案例定时器实现生成的超时事件在超时列表中保存，如 ITU-T X.292 建议书[3]中规定的。在图 2 中，所有这一功能性已经被分配给 T3RTS 实体。它负责存储 SA 或 PA 已经通知 TE 实体但尚未处理的事件。

4.1.2.3 编码/解码系统（EDS）

EDS 实体负责测试数据的编码和解码，这些数据包括在与 SUT 的通信操作中使用的数据，如在 TTCN-3 模块的执行中规定的。如果对 TTCN-3 模块尚未规定编码，则数据值的编码是工具特定的。这一实体由 T3RTS 实体调用和返回。注意 EDS 实体不直接经由 TRL 与 SA 交互。

4.1.2.4 TTCN-3 执行实体中的定时器

在 TTCN-3 ATS 中已经声明和命名的定时器在概念上可以被归类为 TE 中的显式定时器。TE 为保护过程或执行操作而创建的定时器在 TE 中被认为是隐式定时器。显式和隐式定时器都在 TE 内创建，但是由平台适配实体（PA）实现。这可以通过为 TE 中创建的任何一个定时器生成惟一的定时器标识（TID）来完成。这个惟一的 TID 应使得 TE 在不同的定时器之间存在差异。TID 由 TE 用于与 PA 中对应的定时器实现交互。

注意，TE 的责任是正确地实现显式和隐式定时器的不同 TTCN-3 语义，如 ITU-T Z.140 建议书[2]中定义的，例如定时器关键词 `any` 和 `all` 的使用仅适用于显式定时器。在 PA 中，所有定时器，即显式和隐式定时器都以同一方法对待。

4.1.3 SUT适配实体（SA）

SA 使 TTCN-3 测试系统和 SUT 的基于消息和基于过程的通信适配于测试系统的特殊执行平台。已经意识到将 TTCN-3 测试组件通信端口映射到测试系统接口端口并实现实际测试系统接口，如 ITU-T Z.140 建议书[2]中定义的。它负责从执行实体（TE）传播发送请求和 SUT 动作操作到 SUT，并通过将 TE 任何接收的测试事件添加到 TE 的端口队列来通知 TE。

与 SUT 的基于过程的通信操作在 SA 中实现。SA 负责在基于过程的通信内区分不同消息（即 call、reply 和 exception），并以适当的方式将它们传播给 SUT 或 TE。TTCN-3 基于规程的通信语义，即这样的操作对 TTCN-3 测试组件执行的作用，将在 TE 中处理。

SA 具有一个与 TE 的接口，它将被用于发送 SUT 消息（在 TTCN-3 SUT 动作操作中发布）给 SA，并在与 SUT 的通信操作中的两个实体之间交换编码的测试数据。

4.1.4 平台适配实体（PA）

PA 实现 TTCN-3 外部函数，提供给 TTCN-3 测试系统一个单独的时间符号。在这一实体中，外部函数和所有定时器都将实现。注意，定时器实例在 TE 中创建。PA 中的定时器仅能由其定时器标识（TID）区分。因此，PA 以相同的方式对待显式和隐式定时器。

与 TE 的接口使得可以调用外部函数、定时器的启动、读取和停止以及使用它们的定时器 ID 的状态的查询。PA 向 TE 通报期满的定时器。

4.2 TTCN-3测试系统中的接口

如之前图 1 中所描述的，一个 TTCN-3 测试系统具有两个接口，即 TTCN-3 控制接口（TCI）和 TTCN-3 运行时间接口（TRI），它们分别规定测试管理（TM）和 TTCN-3 执行实体（TE）实体之间的接口以及 TE、SUT 适配实体（SA）和平台适配实体（PA）之间的接口。

本建议书定义了 TRI。TE 与 SA 和 PA 之间的交互将按照 TRI 操作在此定义。尽管必须为一个 TTCN-3 测试系统的完整实现两个接口，即 TRI 和 TCI，TCI 的规范和实现目前被认为是专用的。

4.3 TTCN-3测试系统的执行要求

每个 TRI 操作调用应被当做在调用实体中的原子操作对待。实现 TRI 操作的被调用实体只要其预期影响已经实现，或如果操作不能成功完成，就必须返回控制给调用实体。被调用实体不得阻塞基于过程的通信的实现。然而，在外部函数实现调用之后，被调用实体必须阻塞其实现，并等待它的返回值。注意，依据测试系统实现，从外部函数实现返回失败可能导致测试组件执行、TTCN-3 执行实体、平台适配实体或甚至整个测试系统的无限阻塞。

上述规定的执行要求可以在一个紧密整合的测试系统实现中实行。这里，整个 TTCN-3 测试系统在一个单独的执行实体或进程中实现，其中每个测试系统实体在至少一路执行线程中分配。这里 TRI 操作可做为过程调用实现。

注意，测试系统实现的松整合仍是可能的，例如在分布式计算环境下具有多个 SUT 适配实体的 TTCN-3 测试系统的实现。在这一情况下，仅 SUT 适配实体的一小部分是与 TTCN-3 测试系统的剩余部分紧密整合的，但是实际的 SA 适配实体可能在单独的进程中实现。然后 SA 的那一小部分可能仅实现由 TRI 提供给期望的 SUT 适配实体进程（可能在远端主机上执行）的一路信息，反之亦然。

5 TTCN-3运行时间接口和操作

本节按照将在何时使用 TRI 操作和在 TTCN-3 测试系统实现中它们将有何作用定义了 TRI 操作。同样，也定义了用于 TRI 操作的定义的抽象数据类型集。这一定义也包括每个 TRI 操作调用所需的输入参数及其返回值的更详细的描述。

5.1 TRI的概述

TRI 定义了在 TTCN-3 测试系统实现内的 TTCN-3 执行实体（TE）、SUT 适配实体（SA）和平台适配实体（PA）之间的交互。在概念上，它为 TE 发送测试数据给 SUT 或操作定时器提供了方法，类似地，也为通知 TE 接收的测试数据和超时提供了方法。

TRI 可被看做是由两个子接口组成，一个是 triCommunication 接口，另一个是 triPlatform 接口。triCommunication 接口进行 TTCN-3 ETS 与 SUT 的通信，这在 SA 中实现。triPlatform 接口表示操作集，它将 ETS 适配于一个特殊的执行平台。

两个接口都是双向的，所以调用和被调用部分都位于测试系统的 TE、SA 和 PA 实体内。表 1 更详细地示出了各个实体之间调用方/被调用方的关系。注意该表仅示出在 TRI 可见的交互。不反映出同一实体部分之间的内部通信，因为在 TTCN-3 测试系统实现中，TE、SA 和 PA 的内部结构可能不同。

表 1/Z.144－接口概述

接　　口	方向（调用实体→被调用实体）	
名称	TE → SA 或 PA	SA 或 PA → TE
triCommunication	TE → SA	SA → TE
triPlatform	TE → PA	PA → TE

5.1.1 triCommunication 接口

这一接口由实现 TTCN-3 ETS 与 SUT 的通信所必需的操作组成。它包括这些操作，即初始化测试系统接口（TSI）、建立与 SUT 的连接和处理与 SUT 的基于消息和基于过程的通信。另外，triCommunication 接口提供了重置 SUT 适配实体（SA）的操作。

5.1.2 triPlatform 接口

这一接口包括将 TTCN-3 执行实体适配于一个特殊的执行平台所必需的所有操作。triPlatform 接口提供开始、停止、读取定时器、查询其状态的方法以及将超时事件增加到期满定时器列表中的方法。另外，它还提供了调用 TTCN-3 外部函数和重置平台适配实体（PA）的操作。注意，在 triPlatform 接口处要求的显式和隐式定时器之间没有区别。替代地，每个定时器必须一律按照其定时器标识符（TID）称呼。

5.1.3 TTCN-3与TRI操作调用之间的相关性

对于某些 TTCN-3 操作，存在与一个 TRI 操作调用（或者在 TTCN-3 `execute` 和 `call` 操作的情况下也可能是两个 TRI 操作调用）的直接相关性，这在表 2 中示出。对于所有其他 TRI 操作调用，可能没有直接的相关性。

如果在映射到 TSI 端口上的测试组件端口被调用，所示出的相关性仅适用于 TTCN-3 通信操作（即 `send`、`call`、`reply` 和 `raise`）。然而，如果对于测试案例尚未规定系统组件，即仅创建 MTC 测试组件用于测试案例且无其他测试组件，则这一相关性仅适用于所有这样的操作调用。

表 2/Z.144－TTCN-3与TRI操作调用之间的相关性（* = 如果适用）

TTCN-3 操作名称	TRI操作名称	TRI 接口名称
execute	triExecuteTestCase triStartTimer*	TriCommunication TriPlatform
map	triMap	TriCommunication
unmap	triUnmap	TriCommunication
send	triSend （见注 1）	TriCommunication
	triSendBC （见注 2）	
	triSendMC （见注 3）	
call	triCall （见注 1）	TriCommunication
	triCallBC （见注 2）	
	triCallMC （见注 3）	
	triStartTimer*	TriPlatform
reply	triReply （见注 1）	TriCommunication
	triReply （见注 2）	
	triReply （见注 3）	
raise	triRaise （见注 1）	TriCommunication
	triRaise （见注 2）	
	triRaise （见注 3）	
action	triSUTactionInformal	TriCommunication
start （定时器）	triStartTimer	TriPlatform
stop （定时器）	triStopTimer	TriPlatform
read （定时器）	triReadTimer	TriPlatform
running （定时器）	triTimerRunning	TriPlatform
TTCN-3 外部功能	triExternalFunction	TriPlatform
注 1 — 用于单播通信。 注 2 — 用于广播通信。 注 3 — 用于多播通信。		

注意，在表 2 中列出的所有 TRI 操作由 TE 使用，当在 TTCN-3 ETS 内评估 TTCN 快照时，TE 可能以不同的方式实现这些操作的调用。

5.2 错误处理

显式错误处理仅规定用于 TTCN-3 执行实体(TE)调用的 TRI 操作。SA 或 PA 在 TRI 操作的返回值中报告 TRI 操作的状态。状态值可指示 TRI 操作的局部成功（***TRI_OK***）或失败（***TRI_Error***）。因此，TE 可能对发生在 SA 或者 PA 和事件内的错误（例如一个测试案例错误）起反应。

对于由 SA 或 PA 调用的 TRI 操作，不需要显式错误处理，因为这些操作在 PE 内实现。这里，在这样的 TRI 操作中发生一个错误的情况下，TE 控制测试执行。

注意，在任何测试系统实体中，特定的错误编码和错误的检测以及处理都超出了当前 TRI 规范的范围。

5.3 数据接口

在 TRI 操作中，必需仅传送编码的测试数据。TTCN-3 执行实体（TE）负责在各个 TRI 操作中编码将发送的测试数据和解码接收到的测试数据，因为可能为 TTCN-3 模块规定或在其内规定编码规则。注意，即使在 TTCN-3 ATS 内尚未提供编码信息，也要求 TE 编码测试数据。在这一情况下，工具供货商必需定义一个编码。

取代为 TTCN-3 和 ASN.1 数据类型定义一个显式数据接口，TRI 标准定义了抽象数据类型集。这些数据类型在下列 TRI 操作的定义中使用，以指示将从调用实体向被调用实体传送哪一信息，反之亦然。这些抽象数据类型的具体表述以及基本数据类型的定义以各自的语言定义，对应于第 6 和 7 节。

注意任何标识符数据类型的值在测试系统实现中必需是惟一的，其中惟一性被定义为在任何地方任何时间都是全球独一的。

定义了下列抽象数据类型，用于 TRI 操作的定义。

5.3.1 连接

`TriComponentIdType` 类型 `TriComponentIdType` 的值包括一个标识符、一个名称和组件类型。后者的惟一值是组件类型名称，如 TTCN-3 ATS 中规定的。这一抽象类型主要用于解决映射到许多测试组件端口的 TSI 端口上的 TRI 通信操作。

`TriComponentIdListType` 类型 `TriComponentIdListType` 的值是 `TriComponentIdType` 的列表。这一抽象类型用于 TCL 中的多播通信。

`TriPortIdType` 类型 `TriPortIdType` 的值包括 `TriComponentIdType` 类型的值以表示端口所属的组件、端口索引（如果有的话）和端口名称，如 TTCN-3 ATS 中所规定的。`TriPortIdType` 主要必需用于从 TE 传送有关 TSI 的信息和到 TSI 的连接给 SA。

`TriPortIdListType` 类型 `TriPortIdListType` 的值是 `TriPortIdType` 的列表。在测试案例调用之后，这一抽象类型用于初始化。

5.3.2 通信

`TriMessageType` 类型 `TriMessageType` 的值是一个编码测试数据，它将发送给 SUT 或者已从 SUT 接收。

`TriAddressType` 类型 `TriAddressType` 的值指示 SUT 内的源或目的地地址。这一抽象类型可在 TRI 通信操作中使用，它是对 TE 不透明的开放类型。

`TriAddressListType` 类型 `TriAddressListType` 的值是 `TriAddressType` 的列表。这一抽象类型用于 TRI 中的多播通信。

`TriSignatureIdType` 类型 `TriSignatureIdType` 的值是过程特征的名称，如 TTCN-3 ATS 中规定的。这一抽象类型在基于过程的 TRI 通信操作中使用。

`TriParameterType` 类型 `TriParameterType` 的值包括一个编码参数和 `TriParameterPassingModeType` 的值来表示在 TTCN-3 ATS 中为参数规定的标示模式。

`TriParameterPassingModeType` 类型 `TriParameterPassingModeType` 的值是 *in*、*inout* 或 *out*。这一抽象类型在基于过程的 TRI 通信操作中使用，并用于外部函数调用。

`TriParameterListType` 类型 `TriParameterListType` 的值是 `TriParameterType` 的列表。这一抽象类型在基于过程的 TRI 通信操作中使用，并用于外部函数调用。

`TriExceptionType` 类型 `TriExceptionType` 的值是一个异常的编码类型和值，它将发送给 SUT 或者已从 SUT 接收。这一抽象类型在基于过程的 TRI 通信操作中使用。

5.3.3 定时器

`TriTimerIdType` 类型 `TriTimerIdType` 的值为一个定时器规定一个标识符。所有 TRI 定时器操作都需要这一抽象类型。

`TriTimerDurationType` 类型 `TriTimerDurationType` 的值规定定时器的时长，以秒计。

5.3.4 杂项

`TriTestCaseIdType` 类型 `TriTestCaseIdType` 的值是测试案例的名称，如 TTCN-3 ATS 中规定的。

`TriFunctionIdType` 类型 `TriFunctionIdType` 的值是外部函数的名称，如 TTCN-3 ATS 中规定的。

`TriStatusType` 类型 `TriStatusType` 的值是 ***TRI_OK*** 或 ***TRI_Error***，指示 TRI 操作的成功或失败。

5.4 操作描述

所有操作定义使用接口定义语言(IDL)来定义。具体的语言映射在第 6 和 7 节中定义。

对于每个 TRI 操作调用，在特殊操作定义中列出的所有 *in*、*inout* 和 *out* 参数都是强制的。*in* 参数的值由调用实体规定。类似地，*out* 参数的值由被调用实体规定，在 *inout* 参数的情况下，值首先由调用实体规定，但是可以由被调用实体用一个新的值替换。注意，尽管 TTCN-3 也使用 *in*、*inout* 和 *out* 参数来进行特征定义，但是在 TRI IDL 规范中使用的指示与 TTCN-3 规范中的那些指示不相关。

操作调用应使用保留的值来指示不存在在相应 TRI 参数描述中作为可选项定义的参数。这些类型的保留值在每个语言映射中定义，随后将作为 `null` 值引用。

在接口内的所有函数使用下列模板描述:

F.n.m	**操作名称** 调用实体 → 被调用实体
特征	IDL 特征。
In 参数	对作为操作的参数从调用实体向被调用实体传送的数据的描述。
Out 参数	对作为操作的参数从被调用实体向调用实体传送的数据的描述。
InOut 参数	对作为操作的参数从调用实体向被调用实体传送和从被调用实体向调用实体传送的数据的描述。
返回值	对从操作返回给调用实体的数据的描述。
约束	对适用于调用操作的任何约束的描述。
作用	在操作可能返回之前被调用实体所需的行为。

5.5 通信接口操作

5.5.1 triSAReset (TE → SA)

特征	`TriStatusType triSAReset()`
In 参数	n.a.
Out 参数	n.a.
返回值	`triSAReset` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	TE 可在任何时间调用这一操作来重置 SA。
作用	SA 必须重置其正维持的所有通信方式，例如重置到 SUT 的静态连接、关闭到 SUT 的动态连接、丢弃任何悬而未决的消息或过程调用。 在操作已经成功地执行的情况下，`triResetSA` 操作返回 ***TRI_OK***，否则返回 ***TRI_Error***。

5.5.2 连接处理操作

5.5.2.1 triExecuteTestCase (TE → SA)

特征	`TriStatusType triExecuteTestCase( in TriTestCaseIdType testCaseId,` `in TriPortIdListType tsiPortList)`
In 参数	`testCaseId` 将要执行的测试案例的标识符 `tsiPortList` 为测试系统定义的测试系统接口端口列表
Out 参数	n.a.
返回值	`triExecuteTestCase` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	在任何测试案例执行之前由 TE 立即调用这一操作。将要被执行的测试案例由 `testCaseId` 指示。`tsiPortList` 包含对于测试案例在系统组件的定义中已经被声明的所有端口，即 TSI 端口。如果在 TTCN-3 ATS 中尚未为测试案例明确定义一个系统组件，则 `tsiPortList` 包含所有 MTC 测试组件的通信端口。`tsiPortList` 中的端口按照它们在各个 TTCN-3 组件声明中的出现排序。
作用	SA 可建立与 SUT 的任何静态连接，并初始化 TSI 端口的任何通信方法。 在操作已经成功地执行的情况下，`triExecuteTestCase` 操作返回 ***TRI_OK***，否则返回 ***TRI_Error***。

5.5.2.2 triMap (TE → SA)

特征	`TriStatusType triMap( in TriPortIdType compPortId,` `in TriPortIdType tsiPortId)`
In 参数	`compPortId` 将被映射的测试组件端口的标识符 `tsiPortId` 将被映射的测试系统接口的标识符
Out 参数	n.a.
返回值	`triMap` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当 TE 执行 TTCN-3 映射操作时调用这一操作。
作用	对于引用的 TSI 端口，SA 可建立与 SUT 的动态连接。 在连接不能成功建立的情况下，`triMap` 操作返回 ***TRI_Error***，否则返回 ***TRI_OK***。在测试系统不需要建立动态连接的情况下，操作应返回 ***TRI_OK***。

5.5.2.3 triUnmap (TE → SA)

特征	`TriStatusType triUnmap( in TriPortIdType compPortId,` `in TriPortIdType tsiPortId)`
In 参数	`compPortId` 将被去映射的测试组件端口的标识符 `tsiPortId` 将被去映射的测试系统接口的标识符
Out 参数	n.a.
返回值	`triUnmap` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当 TE 执行任何 TTCN-3 去映射操作时调用这一操作。
作用	对于引用的 TSI 端口，SA 必须关闭与 SUT 的动态连接。 在连接不能成功关闭或之前尚未建立这样的连接的情况下，`triUnmap` 操作返回 ***TRI_Error***，否则返回 ***TRI_OK***。在测试系统尚未建立动态连接的情况下，操作应返回 ***TRI_OK***。

5.5.3 基于消息的通信操作

5.5.3.1 triSend (TE → SA)

特征	`TriStatusType triSend( in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriAddressType SUTaddress,`
	`in TriMessageType sendMessage)`

In 参数	`componentId` 发送测试组件的标识符 `tsiPortId` 消息经由其发送给 SUT 适配实体的测试系统接口端口的标识符 `SUTaddress` SUT 内的（任选的）目的地地址 `sendMessage` 将发送的编码消息
Out 参数	n.a.
返回值	`triSend` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当 TE 在一个已经映射到 TSI 端口的组件端口上执行 TTCN-3 单播发送操作时调用这一操作。如果尚未为测试案例规定系统组件，即为测试案例仅创建一个 MTC 测试组件，则对于所有 TTCN-3 发送操作，由 TE 调用这一操作。 `sendMessage` 的编码必须在这一 TRI 操作调用之前在 TE 中进行。
作用	SA 可发送消息给 SUT。 `triSend` 操作在其已经成功完成的情况下返回 ***TRI_OK***。否则必须返回 ***TRI_Error*** 。注意，返回值 ***TRI_OK*** 并不意味着 SUT 已经接收 `sendMessage`。

5.5.3.2 triSendBC (TE → SA)

特征	`TriStatusType triSendBC( in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriMessageType sendMessage)`

In 参数	`componentId` 发送测试组件的标识符 `tsiPortId` 消息经由其发送给 SUT 适配实体的测试系统端口接口的标识符 `sendMessage` 将发送的编码消息
Out 参数	n.a.
返回值	`triSend` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当 TE 在一个已经映射到 TSI 端口的组件端口上执行 TTCN-3 广播发送操作时调用这一操作。如果尚未为测试案例规定系统组件，即为测试案例仅创建一个 MTC 测试组件，则对于所有 TTCN-3 发送操作，由 TE 调用这一操作。 `sendMessage` 的编码必须在这一 TRI 操作调用之前在 TE 中进行。
作用	SA 可广播消息给 SUT。 `triSend` 操作在其已经成功完成的情况下返回 ***TRI_OK***。否则必须返回 ***TRI_Error*** 。注意，返回值 ***TRI_OK*** 并不意味着 SUT 已经接收 `sendMessage`。

5.5.3.3 triSendMC (TE → SA)

特征	`TriStatusType triSendMC( in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriAddressListType SUTaddresses,`
	`in TriMessageType sendMessage)`

In 参数	`componentId` 发送测试组件的标识符 `tsiPortId` 消息经由其发送给 SUT 适配实体的测试系统端口接口的标识符 `SUTaddresses` SUT 内的目的地地址 `sendMessage` 将发送的编码消息
Out 参数	n.a.
返回值	`triSend` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当 TE 在一个已经映射到 TSI 端口的组件端口上执行 TTCN-3 多播发送操作时调用这一操作。如果尚未为测试案例规定系统组件，即为测试案例仅创建一个 MTC 测试组件，则对于所有 TTCN-3 发送操作，由 TE 调用这一操作。 `sendMessage` 的编码必须在这一 TRI 操作调用之前在 TE 中进行。
作用	SA 可多播消息给 SUT。 `triSend` 操作在其已经成功完成的情况下返回 ***TRI_OK***。否则必须返回 ***TRI_Error*** 。注意，返回值 ***TRI_OK*** 并不意味着 SUT 已经接收 `sendMessage`。

5.5.3.4 triEnqueueMsg (SA → TE)

特征	`void triEnqueueMsg( in TriPortIdType tsiPortId,`
	`in TriAddressType SUTaddress,`
	`in TriComponentIdType componentId,`
	`in TriMessageType receivedMessage)`

In 参数	`tsiPortId` 消息经由其由 SUT 适配实体排队的测试系统接口端口的标识符 `SUTaddress` SUT 内的（任选的）源地址 `componentId` 接收测试组件的标识符 `receivedMessage` 编码的接收消息
Out 参数	n.a.
返回值	空
约束	SA 在其已经从 SUT 接收消息后调用这一操作。仅当之前 tsiPortId 已经映射到一个 componentId 端口或者在前一个 triExecuteTestCase 声明中已经被引用时可使用它。 在 `triEnqueueMsg` 操作的调用中，`receivedMessage` 必须包含一个编码值。
作用	这一操作必须传送消息给 TE，指示 TSI 端口 tsiPortId 被映射到的组件 componentId。 receivedMessage 的解码必须在 TE 中进行。

5.5.4 基于过程的通信操作

5.5.4.1 triCall (TE → SA)

特征	`TriStatusType triCall( in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriAddressType SUTaddress,`
	`in TriSignatureIdType signatureId,`
	`in TriParameterListType parameterList)`

In 参数	`componentId` 发布过程调用的测试组件的标识符 `tsiPortId` 过程调用经由其发送给 SUT 适配实体的测试系统端口接口的标识符 `SUTaddress` SUT 内的（任选的）目的地地址 `signatureId` 过程调用的特征的标识符 `parameterList` 编码参数（指示的特征的一部分）的列表。`parameterList` 中的参数按照它们在 TTCN-3 特征声明中的出现排序
Out 参数	n.a.
返回值	`triCall` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当 TE 在一个已经映射到 TSI 端口的组件端口上执行 TTCN-3 单播调用操作时调用这一操作。如果尚未为测试案例规定系统组件，即为测试案例仅创建一个 MTC 测试组件，则对于所有 TTCN-3 调用操作，由 TE 调用这一操作。 所有的 *in* 和 *inout* 过程参数包含编码的值。 这些过程参数是在 TTCN-3 特征模板中规定的参数。它们的编码必须在这一 TRI 操作调用之前在 TE 中进行。
作用	在这一操作的调用中，SA 可初始化对应于特征标识符 `signatureId` 和 TSI 端口 `tsiPortId` 的过程调用。 `triCall` 操作必须返回而无需等待发布的过程调用的返回（见注）。这一 TRI 操作当过程调用成功初始化时调用 *TRI_OK*，否则返回 *TRI_Error*。在任何 *out* 参数的值非空的情况下，SA 必须指示无错误。注意这一 TRI 操作的返回值未做出任何有关过程调用成功或失败的声明。 注意可在 TTCN-3 ATS 中为调用操作规定的一个任选超时值，不包括在 `triCall` 操作特征中。TE 负责用一个单独的 TRI 操作调用，即 `triStartTimer`，为 PA 内的 TTCN-3 调用操作启动一个定时器来进行这一发布。
注 — 这可能通过例如产生一个新的线程或进程来实现。然而这一过程调用的处理取决于 TE 的实现。	

5.5.4.2 triCallBC (TE → SA)

特征	`TriStatusType triCallBC( in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriSignatureIdType signatureId, in TriParameterListType parameterList)`
In 参数	`componentId` 发布过程调用的测试组件的标识符 `tsiPortId` 过程调用经由其发送给 SUT 适配实体的测试系统端口接口的标识符 `signatureId` 过程调用的特征的标识符 `parameterList` 编码参数（指示的特征的一部分）的列表。`parameterList` 中的参数按照它们在 TTCN-3 特征声明中的出现排序
Out 参数	n.a.
返回值	`triCall` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当 TE 在一个已经映射到 TSI 端口的组件端口上执行 TTCN-3 广播调用操作时调用这一操作。如果尚未为测试案例规定系统组件，即为测试案例仅创建一个 MTC 测试组件，则对于所有 TTCN-3 调用操作，由 TE 调用这一操作。 所有的 *in* 和 *inout* 过程参数包含编码的值。 这些过程参数是在 TTCN-3 特征模板中规定的参数。它们的编码必须在这一 TRI 操作调用之前在 TE 中进行。
作用	在这一操作的调用中，SA 可初始化和广播对应于特征标识符 `signatureId` 和 TSI 端口 `tsiPortId` 的过程调用。 `triCall` 操作必须返回而无需等待发布的过程调用的返回（见注）。这一 TRI 操作当过程调用成功初始化时调用 *TRI_OK*，否则返回 *TRI_Error*。在任何 *out* 参数的值非空的情况下，SA 必须指示无错误。注意这一 TRI 操作的返回值未做出任何有关过程调用成功或失败的声明。 注意可在 TTCN-3 ATS 中为调用操作规定的一个任选超时值，不包括在 `triCall` 操作特征中。TE 负责用一个单独的 TRI 操作调用，即 `triStartTimer`，为 PA 内的 TTCN-3 调用操作启动一个定时器来进行这一发布。
注 — 这可能通过例如产生一个新的线程或进程来实现。然而这一过程调用的处理取决于 TE 的实现。	

5.5.4.3 triCallMC (TE → SA)

特征	`TriStatusType triCallMC( in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriAddressListType SUTaddresses,`
	`in TriSignatureIdType signatureId,`
	`in TriParameterListType parameterList)`

In 参数	`componentId` 发布过程调用的测试组件的标识符 `tsiPortId` 过程调用经由其发送给 SUT 适配实体的测试系统端口接口的标识符 `SUTaddresses` SUT 内的目的地地址 `signatureId` 过程调用的特征的标识符 `parameterList` 编码参数（指示的特征的一部分）的列表。`parameterList` 中的参数按照它们在 TTCN-3 特征声明中的出现排序。
Out 参数	n.a.
返回值	`triCall` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当 TE 在一个已经映射到 TSI 端口的组件端口上执行 TTCN-3 多播调用操作时调用这一操作。如果尚未为测试案例规定系统组件，即为测试案例仅创建一个 MTC 测试组件，则对于所有 TTCN-3 调用操作，由 TE 调用这一操作。 所有的 *in* 和 *inout* 过程参数包含编码的值。 这些过程参数是在 TTCN-3 特征模板中规定的参数。它们的编码必须在这一 TRI 操作调用之前在 TE 中进行。
作用	在这一操作的调用中，SA 可初始化和多播对应于特征标识符 `signatureId` 和 TSI 端口 `tsiPortId` 的过程调用。 `triCall` 操作必须返回而无需等待发布的过程调用的返回（见注）。这一 TRI 操作当过程调用成功初始化时调用 *TRI_OK*，否则返回 *TRI_Error* 。在任何 *out* 参数的值非空的情况下，SA 必须指示无错误。注意这一 TRI 操作的返回值未做出任何有关过程调用成功或失败的声明。 注意可在 TTCN-3 ATS 中为调用操作规定的一个任选超时值，不包括在 `triCall` 操作特征中。TE 负责用一个单独的 TRI 操作调用，即 `triStartTimer`，为 PA 内的 TTCN-3 调用操作启动一个定时器来进行这一发布。
注 — 这可能通过例如产生一个新的线程或进程来实现。然而这一过程调用的处理取决于 TE 的实现。	

5.5.4.4 triReply (TE → SA)

特征	`TriStatusType triReply( in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriAddressType SUTaddress,`
	`in TriSignatureIdType signatureId,`
	`in TriParameterListType parameterList,`
	`in TriParameterType returnValue)`

In 参数	`componentId` 回答测试组件的标识符 `tsiPortId` 回答经由其发送给 SUT 适配实体的测试系统端口接口的标识符 `SUTaddress` SUT 内的（任选的）目的地地址 `signatureId` 过程调用的特征的标识符 `parameterList` 编码参数（指示的特征的一部分）的列表。`parameterList` 中的参数按照它们在 TTCN-3 特征声明中的出现排序 `returnValue` 过程调用（任选的）编码返回值
Out 参数	n.a.
返回值	`triReply` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当 TE 在一个已经映射到 TSI 端口的组件端口上执行 TTCN-3 单播回答操作时调用一这操作。如果尚未为测试案例规定系统组件，即为测试案例仅创建一个 MTC 测试组件，则对于所有 TTCN-3 回答操作，由 TE 调用这一操作。 所有的 *in* 和 *inout* 过程参数和返回值包含编码的值。 `parameterList` 包含过程调用参数。这些过程参数是在 TTCN-3 特征模板中规定的参数。它们的编码必须在这一 TRI 操作调用之前在 TE 中进行。 如果在 TTCN-3 ATS 内尚未为过程特征定义返回类型，则惟一值 `null` 必须作为返回值传送。
作用	在这一操作的调用中，SA 可发布对应于特征标识符 `signatureId` 和 TSI 端口 `tsiPortId` 的过程调用的回答。 对于这一操作的成功执行，`triReply` 操作将返回 ***TRI_OK***，否则返回 ***TRI_Error***。在任何 *in* 参数的值非空或者未定义的返回值与 null 不同的情况下，SA 必须指示无错误。

5.5.4.5 triReplyBC (TE → SA)

特征	`TriStatusType triReplyBC( in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriSignatureIdType signatureId,`
	`in TriParameterListType parameterList,`
	`in TriParameterType returnValue)`

In 参数	`componentId` 回答测试组件的标识符 `tsiPortId` 回答经由其发送给 SUT 适配实体的测试系统端口接口的标识符 `signatureId` 过程调用的特征的标识符 `parameterList` 编码参数（指示的特征的一部分）的列表。`parameterList` 中的参数按照它们在 TTCN-3 特征声明中的出现排序 `returnValue` 过程调用（任选的）编码返回值
Out 参数	n.a.
返回值	`triReply` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当 TE 在一个已经映射到 TSI 端口的组件端口上执行 TTCN-3 广播回答操作时调用这一操作。如果尚未为测试案例规定系统组件，即为测试案例仅创建一个 MTC 测试组件，则对于所有 TTCN-3 回答操作，由 TE 调用这一操作。 所有的 *in* 和 *inout* 过程参数和返回值包含编码的值。 `parameterList` 包含过程调用参数。这些过程参数是在 TTCN-3 特征模板中规定的参数。它们的编码必须在这一 TRI 操作调用之前在 TE 中进行。 如果在 TTCN-3 ATS 内尚未为过程特征定义返回类型，则惟一值 `null` 必须作为返回值传送。
作用	在这一操作的调用中，SA 可广播对应于特征标识符 `signatureId` 和 TSI 端口 `tsiPortId` 的过程调用的回答。 对于这一操作的成功执行，`triReply` 操作将返回 ***TRI_OK***，否则返回 ***TRI_Error***。在任何 *in* 参数的值非空或者未定义的返回值与 null 不同的情况下，SA 必须指示无错误。

5.5.4.6 triReplyMC (TE → SA)

特征	`TriStatusType triReplyMC( in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriAddressListType SUTaddresses,`
	`in TriSignatureIdType signatureId,`
	`in TriParameterListType parameterList,`
	`in TriParameterType returnValue)`

In 参数	`componentId` 回答测试组件的标识符 `tsiPortId` 回答经由其发送给 SUT 适配实体的测试系统端口接口的标识符 `SUTaddresses` SUT 内的目的地地址 `signatureId` 过程调用的特征的标识符 `parameterList` 编码参数（指示的特征的一部分）的列表。`parameterList` 中的参数按照它们在 TTCN-3 特征声明中的出现排序 `returnValue` 过程调用（任选的）编码返回值
Out 参数	n.a.
返回值	`triReply` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当 TE 在一个已经映射到 TSI 端口的组件端口上执行 TTCN-3 多播回答操作时调用这一操作。如果尚未为测试案例规定系统组件，即为测试案例仅创建一个 MTC 测试组件，则对于所有 TTCN-3 回答操作，由 TE 调用这一操作。 所有的 *in* 和 *inout* 过程参数和返回值包含编码的值。 `parameterList` 包含过程调用参数。这些过程参数是在 TTCN-3 特征模板中规定的参数。它们的编码必须在这一 TRI 操作调用之前在 TE 中进行。 如果在 TTCN-3 ATS 内尚未为过程特征定义返回类型，则惟一值 `null` 必须作为返回值传送。
作用	在这一操作的调用中，SA 可多播对应于特征标识符 `signatureId` 和 TSI 端口 `tsiPortId` 的过程调用的回答。 对于这一操作的成功执行，`triReply` 操作将返回 ***TRI_OK***，否则返回 ***TRI_Error***。在任何 *in* 参数的值非空或者未定义的返回值与 null 不同的情况下，SA 必须指示无错误。

5.5.4.7 triRaise (TE → SA)

特征	`TriStatusType triRaise( in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriAddressType SUTaddress,`
	`in TriSignatureIdType signatureId,`
	`in TriExceptionType exc)`

In 参数	`componentId` 提出异常的测试组件的标识符 `tsiPortId` 异常经由其发送给 SUT 适配实体的测试系统端口接口的标识符 `SUTaddress` SUT 内的（任选的）目的地地址 `signatureId` 异常与之相关的过程调用的特征的标识符 `exc` 编码的异常
Out 参数	n.a.
返回值	`triRaise` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当 TE 在一个已经映射到 TSI 端口的组件端口上执行 TTCN-3 单播提出异常（raise）操作时调用这一操作。如果尚未为测试案例规定系统组件，即为测试案例仅创建一个 MTC 测试组件，则对于所有 TTCN-3 raise 操作，由 TE 调用这一操作。 异常的编码必须在这一 TRI 操作调用之前已经在 TE 中进行。
作用	在这一操作的调用中，SA 可提出对应于特征标识符 `signatureId` 和 TSI 端口 `tsiPortId` 的过程调用的异常。 对于这一操作的成功执行，`triRaise` 操作返回 *TRI_OK*，否则返回 *TRI_Error*。

5.5.4.8 triRaiseBC (TE → SA)

特征	`TriStatusType triRaiseBC( in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriSignatureIdType signatureId,`
	`in TriExceptionType exc)`

In 参数	`componentId` 提出异常的测试组件的标识符 `tsiPortId` 异常经由其发送给 SUT 适配实体的测试系统端口接口的标识符 `signatureId` 异常与之相关的过程调用的特征的标识符 `exc` 编码的异常
Out 参数	n.a.
返回值	`triRaise` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当 TE 在一个已经映射到 TSI 端口的组件端口上执行 TTCN-3 广播 raise 操作时调用这一操作。如果尚未为测试案例规定系统组件，即为测试案例仅创建一个 MTC 测试组件，则对于所有 TTCN-3 raise 操作，由 TE 调用这一操作。 异常的编码必须在这一 TRI 操作调用之前已经在 TE 中进行。
作用	在这一操作的调用中，SA 可提出和广播对应于特征标识符 `signatureId` 和 TSI 端口 `tsiPortId` 的过程调用的异常。 对于这一操作的成功执行，`triRaise` 操作返回 *TRI_OK*，否则返回 *TRI_Error*。

5.5.4.9 triRaiseMC (TE → SA)

特征	`TriStatusType triRaiseMC( in TriComponentIdType componentId,`
	`in TriPortIdType tsiPortId,`
	`in TriAddressListType SUTaddresses,`
	`in TriSignatureIdType signatureId,`
	`in TriExceptionType exc)`

In 参数	`componentId` 提出异常的测试组件的标识符 `tsiPortId` 异常经由其发送给 SUT 适配实体的测试系统端口接口的标识符 `SUTaddresses` SUT 内的目的地地址 `signatureId` 异常与之相关的过程调用的特征的标识符 `exc` 编码的异常
Out 参数	n.a.
返回值	`triRaise` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当 TE 在一个已经映射到 TSI 端口的组件端口上执行 TTCN-3 多播 raise 操作时调用这一操作。如果尚未为测试案例规定系统组件，即为测试案例仅创建一个 MTC 测试组件，则对于所有 TTCN-3 raise 操作，由 TE 调用这一操作。 异常的编码必须在这一 TRI 操作调用之前已经在 TE 中进行。
作用	在这一操作的调用中，SA 可提出和多播对应于特征标识符 `signatureId` 和 TSI 端口 `tsiPortId` 的过程调用的异常。 对于这一操作的成功执行，`triRaise` 操作返回 *TRI_OK*，否则返回 *TRI_Error*。

5.5.4.10 triEnqueueCall (SA → TE)

特征	`void triEnqueueCall( in TriPortIdType tsiPortId,`
	`in TriAddressType SUTaddress,`
	`in TriComponentIdType componentId,`
	`in TriSignatureIdType signatureId,`
	`in TriParameterListType parameterList)`

In 参数	`tsiPortId` 过程调用经由其由 SUT 适配实体排队的测试系统端口接口的标识符 `SUTaddress` SUT 内的（任选的）源地址 `componentId` 接收测试组件的标识符 `signatureId` 过程调用的特征的标识符 `parameterList` 编码参数（指示的特征的一部分）的列表。`parameterList` 中的参数按照它们在 TTCN-3 特征声明中的出现排序。对作为操作的参数从调用实体向被调用实体传送的数据的描述。
Out 参数	n.a.
返回值	void
约束	SA 在其已经从 SUT 接收一个过程调用后可调用这一操作。它可仅当 `tsiPortId` 之前已经映射到 `componentId` 的端口或在之前的 `triExecuteTestCase` 声明内引用时使用。 在 `triEnqueueCall` 操作的调用中，所有 *in* 和 *inout* 过程参数包含编码的值。
作用	TE 可用 TSI 端口 `tsiPortId` 映射到的组件 `componentId` 的端口的特征标识符 `signatureId` 对这一过程调用排队。过程参数的解码必须在 TE 中进行。 在任何 *out* 参数的值非空的情况下，TE 必须指示无错误。

5.5.4.11 triEnqueueReply (SA → TE)

特征	`void triEnqueueReply( in TriPortIdType tsiPortId,`
	`in TriAddressType SUTaddress,`
	`in TriComponentIdType componentId,`
	`in TriSignatureIdType signatureId,`
	`in TriParameterListType parameterList,`
	`in TriParameterType returnValue)`

In 参数	`tsiPortId` 回答经由其由 SUT 适配实体排队的测试系统端口接口的标识符 `SUTaddress` SUT 内的（任选的）源地址 `componentId` 接收测试组件的标识符 `signatureId` 过程调用的特征的标识符 `parameterList` 编码参数（指示的特征的一部分）的列表。`parameterList` 中的参数按照它们在 TTCN-3 特征声明中的出现排序。 `returnValue` 过程调用（任选的）编码返回值
Out 参数	n.a.
返回值	void
约束	SA 在其已经从 SUT 接收一个回答后可调用这一操作。它可仅当 `tsiPortId` 之前已经映射到 `componentId` 的端口或在之前的 `triExecuteTestCase` 声明内引用时使用。 在 `triEnqueueReply` 操作的调用中，所有 *out* 和 *inout* 过程参数包含编码的值。 如果尚未为 TTCN-3 ATS 内的过程特征定义返回类型，必须为返回值使用惟一值 `null`。
作用	TE 可用 TSI 端口 `tsiPortId` 映射到的组件 `componentId` 的端口的特征标识符 `signatureId` 对过程调用的这一回答排队。过程参数的解码必须在 TE 中进行。 在任何 *in* 参数的值非空的情况下，TE 必须指示无错误。

5.5.4.12 triEnqueueException (SA → TE)

特征	`void triEnqueueException( in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriComponentIdType componentId, in TriSignatureIdType signatureId, in TriExceptionType exc)`
In 参数	`tsiPortId` 异常经由其由 SUT 适配实体排队的测试系统端口接口的标识符 `SUTaddress` SUT 内的（任选的）源地址 `componentId` 接收测试组件的标识符 `signatureId` 异常与之相关的过程调用的特征的标识符 `exc` 编码的异常
Out 参数	n.a.
返回值	void
约束	SA 在其已经从 SUT 接收一个回答后可调用这一操作。它可仅当 `tsiPortId` 之前已经映射到 `componentId` 的端口或在之前的 `triExecuteTestCase` 声明内引用时使用。 在 `triEnqueueException` 操作的调用中，`exception` 必须包含一个编码的值。
作用	TE 可用 TSI 端口 `tsiPortId` 映射到的组件 `componentId` 的端口的特征标识符 `signatureId` 对这一过程调用的异常排队。 过程参数的解码必须在 TE 中进行。

5.5.5 杂项操作

5.5.5.1 triSUTactionInformal (TE → SA)

特征	`TriStatusType triSUTactionInformal(in string description)`
In 参数	`description` 将在 SUT 上进行的动作的非正式描述
Out 参数	n.a.
返回值	`triSUTactionInformal` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	TE 当其执行一个 TTCN-3 SUT 动作操作时调用这一操作，它仅包含一个串。
作用	在调用这一操作时，SA 必须初始化将在 SUT 进行的描述的动作，例如，转换、初始化或发送一个消息给 SUT。 对于这一操作的成功执行，`triSUTactionInformal` 操作返回 *TRI_OK*，否则返回 *TRI_Error*。注意这一 TRI 操作的返回值未做出任何与将在 SUT 上进行的动作的成功或失败有关的声明。

5.5.5.2 triSUTactionTemplate (TE → SA)

废弃。

5.6 平台接口操作

5.6.1 triPAReset (TE → PA)

特征	`TriStatusType triPAReset()`
In 参数	n.a.
Out 参数	n.a.
返回值	`triPAReset` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	TE 可在任何时间调用这一操作来重置 PA。
作用	PA 必须重置其当前执行的所有定时行为，例如停止所有的运行定时器，丢弃超时定时器的任何悬而未决的超时。 在操作已经成功执行的情况下，`triResetSA` 操作返回 *TRI_OK*，否则返回 *TRI_Error*。

5.6.2 定时器操作

5.6.2.1 triStartTimer (TE → PA)

特征	`TriStatusType triStartTimer( in TriTimerIdType timerId, in TriTimerDurationType timerDuration)`
In 参数	`timerId` 定时器实例的标识符 `timerDuration` 定时器时长，以秒计
Out 参数	n.a.
返回值	`triStartTimer` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当需要启动定时器时由 TE 调用这一操作。
作用	在调用这一次操作时，PA 必须启动具有指定时长的指定的定时器。定时器从值零（0.0）运行到 timerDuration 规定的最大值。如果指定的定时器已经运行，则它将重启动。当定时器期满，PA 将调用具有 `timerId` 的 `triTimeout()` 操作。 如果定时器已经成功启动，则 `triStartTimer` 操作返回 *TRI_OK*，否则返回 *TRI_Error*。

5.6.2.2 triStopTimer (TE → PA)

特征	`TriStatusType triStopTimer( in TriTimerIdType timerId)`
In 参数	`timerId` 定时器实例的标识符
Out 参数	n.a.
返回值	`triStopTimer` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当将要停止定时器时由 TE 调用这一操作。
作用	在调用这一次操作时，PA 必须使用 `timerId` 来停止指定的定时器实例。不活动定时器（即尚未启动或者已经期满的定时器）的停止应无作用。 如果操作已经成功执行，则 `triStopTimer` 操作返回 ***TRI_OK***，否则返回 ***TRI_Error***。注意，不活动定时器的停止是一个有效的操作。在这一情况下，必须返回 ***TRI_OK***。

5.6.2.3 triReadTimer (TE → PA)

特征	`TriStatusType triReadTimer( in TriTimerIdType timerId, out TriTimerDurationType elapsedTime)`
In 参数	`timerId` 定时器实例的标识符
Out 参数	`elapsedTime` 自定时器已经启动起所经历的时间的值，按秒计
返回值	`triReadTimer` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当将在指定定时器上执行 TTCN-3 读取定时器操作时，TE 可能调用这一操作（见第 5.3.1 节）。
作用	在调用这一次操作时，PA 必须使用 `timerId` 获取自这一定时器启动起所经历的时间。返回值 `elapsedTime` 必须以秒提供。不活动定时器（即尚未启动或者已经期满的定时器）的读取，必须返回为 0 的所经历的时间。 如果操作已经成功执行，则 `triReadTimer` 操作返回 ***TRI_OK***，否则返回 ***TRI_Error*** 。

5.6.2.4 triTimerRunning (TE → PA)

特征	`TriStatusType triTimerRunning( in TriTimerIdType timerId, out boolean running)`
In 参数	`timerId` 定时器实例的标识符
Out 参数	`running` 定时器的状态
返回值	`triTimerRunning` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	当将在指定定时器上执行 TTCN-3 运行定时器操作时，TE 可能调用这一操作（见第 5.3.1 节）。
作用	在调用这一次操作时，PA 必须使用 `timerId` 获取定时器的状态。如果和仅仅如果定时器当前正在运行，则操作设置运行为布尔值 `true`。 如果定时器的状态已经成功地确定，则 `triTimerRunning` 操作返回 ***TRI_OK***，否则返回 ***TRI_Error***。

5.6.2.5 triTimeout (PA → TE)

特征	`void triTimeout(in TriTimerIdType timerId)`
In 参数	`timerId` 定时器实例的标识符
Out 参数	n.a.
返回值	void
约束	在之前已经使用 `triStartTimer` 操作启动的定时器期满之后，即它已经达到其最大时长值，PA 调用这一操作。
作用	具有 `timerId` 超时可被加入到 `TE` 中的超时列表。`TE` 中这一操作的实现必须以它为在 ITU-T Z.143 建议书 [4]中定义的定时器说明不同的 TTCN-3 语义的方式进行（也见第 `5.3.1` 节）。

5.6.3 杂项操作

5.6.3.1 triExternalFunction (TE → PA)

特征	`TriStatusType triExternalFunction( in TriFunctionIdType functionId, inout TriParameterListType parameterList, out TriParameterType returnValue)`
In 参数	`functionId` 外部函数的标识符
Out 参数	`returnValue` （任选的）编码返回值
InOutParameters	`parameterList` 指定函数的编码参数列表。`parameterList` 中的参数按照它们在 TTCN-3 函数声明中的出现排序。
返回值	`triExternalFunction` 操作的返回状态。返回状态指示操作局部成功（***TRI_OK***）或失败（***TRI_Error***）。
约束	TE 当执行被定义为 TTCN-3 外部（即所有非外部函数都在 TE 内实现）的函数时调用这一操作。 在由 TE 对 `triExternalFunction` 操作的调用中，所有 *in* 和 *inout* 函数参数包含编码的值。在任何 *out* 参数的值非空的情况下，TE 必须指示无错误。
作用	对于每个在 TTCN-3 ATS 中规定的外部函数，PA 必须实现行为。在这一操作的调用中，PA 必须调用由标识符 `functionId` 指示的函数。它必须在 `parameterList` 中获取指定的 *in* 和 *inout* 函数参数，使用这些参数的值评估外部函数，并为 `parameterList` 中的 *inout* 和 *out* 参数计算值。然后操作必须为 *inout* 和 *out* 参数返回编码的值，为外部函数返回编码的值。 如果在 TTCN-3 ATS 中尚未为这一外部函数定义返回类型，则必须为后者使用惟一值 `null`。 如果 PA 成功完成了外部函数的评估，则 `triExternalFunction` 操作返回 ***TRI_OK***，否则返回 ***TRI_Error***。 但是要注意，所有其他的 TRI 操作被认为是非阻塞的，`triExternalFunction` 操作被认为是阻塞的。那意味着操作不得在指定的外部函数已经完全评估之前返回。外部函数必须仔细执行，因为它们会造成测试组件执行或甚至整个测试系统实现的死锁。

6 Java语言映射

6.1 引言

本节介绍了 TRITRI Java 语言映射。出于效率的原因，介绍一种专用的语言映射，取代使用 OMG IDL 到 Java 语言的映射。

TTCN-3 运行时间接口的 Java 语言定义了在第 5 节中描述的 IDL 语言是如何映射到 Java 语言的。语言映射与使用的 Java 版本无关，因为它们都仅使用基本的 Java 语言结构。

6.2 名称和范围

6.2.1 名称

尽管在 IDL 定义和 Java 语言中使用的标识符之间没有冲突，但是某些命名解析规则适用于 IDL 标识符。

- Java 参数标识符必须以小写字母开头，构建参数的随后部分将以大写字母开头。例如，IDL 参数标识符 **`SUTaddress`** 映射为 Java 中的 `sutAddress`。
- Java 接口或分类标识符省略了在 IDL 定义中使用的拖尾 `Type`。例如，IDL 类型 **`TriPortIdType`** 映射为 Java 中的 `TriPortId`。

得出的映射遵循 Java 编码管理的标准。

6.2.2 范围

IDL 模块 **`triInterface`** 被映射到 Java 包 `org.etsi.ttcn3.tri`。在这一模块内的所有 IDL 类型声明被映射到这一包内的 Java 分类或接口声明。

6.3 类型映射

6.3.1 基本类型映射

表 3 给出了有关如何将所使用的基本 IDL 类型映射到 Java 类型的概述。

表 3/Z.144－基本类型映射

IDL类型	Java 类型
`boolean`	`org.etsi.ttcn.tri.TriBoolean`
`string`	`java.lang.String`

其他 IDL 基本类型不在 IDL 定义内使用。

6.3.1.1 Boolean类型

IDL **`boolean`** 类型被映射到接口 `org.etsi.ttcn.tri.TriBoolean`，所以实现这一接口的对象可担当持有者对象。

下列接口定义用于 `org.etsi.ttcn.tri.TriBoolean`:

```
// TriBoolean
package org.etsi.ttcn.tri;
public interface TriBoolean {
    public void setBooleanValue(boolean value);
    public boolean getBooleanValue();
}
```

6.3.1.1.1 方法

- `setBooleanValue(boolean value)`

  设置这一 `TriBoolean` 为布尔值 `value`。

- `getBooleanValue()`

  返回由这一 `TriBoolean` 表示的布尔值。

6.3.1.2 String类型

IDL **string** 类型被映射到接口 java.lang.String 分类，而不检验串中字符的范围或界限。在TTCN-3 中定义的所有可能的串可被转换为 java.lang.String。

6.3.2 结构类型映射

TRI IDL 描述将用户定义类型定义为本机类型。在 Java 语言映射中，这些类型被映射到 Java 接口。接口定义了实现这一接口的对象可获得的方法和属性。

6.3.2.1 TriPortIdType

TriPortIdType 被映射到下列接口：

```
// TRI IDL TriPortIdType
package org.etsi.ttcn.tri;
public interface TriPortId {
    public String getPortName();
    public TriComponentId getComponent();
    public boolean isArray();
    public int getPortIndex();
}
```

6.3.2.1.1 方法

- getPortName()

返回端口名称，如 TTCN-3 规范所定义的。

- getComponent()

返回这一 TriPortId 所属的组件标识符，如 TTCN-3 规范所定义的。

- isArray()

如果这一端口是端口数组的一部分，返回 true，否则返回 false。

- getPortIndex()

如果这一端口是以零开始的端口数组的一部分，返回端口索引。如果端口不是端口数组的一部分，返回–1。

6.3.2.2 TriPortIdListType

TriPortIdListType 被映射到下列接口：

```
// TRI IDL TriPortIdListType
package org.etsi.ttcn.tri;
public interface TriPortIdList {
    public int size();
    public boolean isEmpty();
    public java.util.Enumeration getPortIds();
    public TriPortId get(int index);
}
```

6.3.2.2.1 方法

size()

返回这一列表中端口的数量。

isEmpty()

如果这一列表不包含端口，则返回 true。

getPortIds()

在列表的端口上返回 Enumeration。枚举按照它们在列表中出现的相同顺序提供端口。

get(int index)

在指定的位置返回 TriPortId。

6.3.2.3 TriComponentIdType

TriComponentIdType 被映射到下列接口：

```
// TRI IDL TriComponentIdType
package org.etsi.ttcn.tri;
public interface TriComponentId {
    public String getComponentId();
    public String getComponentName();
    public String getComponentTypeName();
    public TriPortIdList getPortList();
    public boolean equals(TriComponentId port);
}
```

6.3.2.3.1 方法

- getComponentId()

返回这一惟一组件标识符的表述。

- getComponentName()

返回组件名称，如 TTCN-3 规范所定义的。如果未提供名称，则返回空串。

- getComponentTypeName()

返回组件类型名称，如 TTCN-3 规范所定义的。

- getPortList()

返回组件的端口列表，如 TTCN-3 规范所定义的。

- equals(TriComponentId component)

比较 component 与这一 TriComponentId 是否相等。如果且仅如果两个组件具有这一惟一组件标识符的相同表述，返回 true，否则返回 false。

6.3.2.4 TriComponentIdListType

TriComponentIdListType 被映射到下列接口：

```
// TRI IDL TriComponentIdListType
package org.etsi.ttcn.tri;
public interface TriComponentIdListType {
    public int size();
    public boolean isEmpty();
    public java.util.Enumeration getComponents();
    public TriComponentId get(int index);
    public void clear();
    public void add(TriComponentId comp);
}
```

6.3.2.4.1 方法

size()

返回这一列表中组件的数量。

isEmpty()

如果这一列表不包含组件，则返回 true。

getComponents()

在列表的组件上返回 Enumeration。枚举按照它们在列表中出现的相同顺序提供组件。

get(int index)

在指定的位置返回 TriComponentId。

clear()

从这一 TriComponentIdList 中移除所有组件。

add(TriComponentId comp)

将 comp 加入到这一 TriComponentIdList 的末尾。

6.3.2.5 TriMessageType

TriMessageType 被映射到下列接口：

```
// TRI IDL TriMessageType
package org.etsi.ttcn.tri;
public interface TriMessage {
    public byte[] getEncodedMessage();
    public void setEncodedMessage(byte[] message);
    public boolean equals(TriMessage message);
}
```

6.3.2.5.1 方法

- getEncodedMessage()

    按照在 TTCN-3 规范中定义的编码规则返回编码的消息。

- setEncodedMessage(byte[] message)

    设置这一 TriMessage 的编码消息表述为 message。

- equals(TriMessage message)

    比较 message 与这一 TriMessage 是否相等。如果且仅如果两个消息具有相同的编码表述，返回 true，否则返回 false。

6.3.2.6 TriAddressType

TriAddressType 被映射到下列接口：

```
// TRI IDL TriAddressType
package org.etsi.ttcn.tri;
public interface TriAddress {
    public byte[] getEncodedAddress();
    public void setEncodedAddress(byte[] address);
    public boolean equals(TriAddress address);
}
```

6.3.2.6.1 方法

- getEncodedAddress()

    返回编码的地址。

- setEncodedAddress(byte[] address)

    设置这一 TriAddress 的编码地址为 address。

- equals(TriAddress address)

    比较 address 与这一 TriAddress 是否相等。如果且仅如果两个地址具有相同的编码表述，返回 true，否则返回 false。

6.3.2.7 TriAddressListType

TriAddressListType 被映射到下列接口：

```
// TRI IDL TriAddressListType
package org.etsi.ttcn.tri;
public interface TriAddressListType {
    public int size();
    public boolean isEmpty();
    public java.util.Enumeration getAddresses();
    public TriAddress get(int index);
    public void clear();
    public void add(TriAddress addr);
}
```

6.3.2.7.1 方法

size()

返回这一列表中组件的数量。

isEmpty()

如果这一列表不包含组件，则返回 true。

getAddresses()

在列表的组件上返回 Enumeration。枚举按照它们在列表中出现的相同顺序提供地址。

get(int index)

在指定的位置返回 TriAddress。

clear()

从这一 TriAddressList 中移除所有地址。

add(TriAddress comp)

将 comp 加入到这一 TriAddressList 的末尾。

6.3.2.8 TriSignatureIdType

TriSignatureIdType 被映射到下列接口：

```
// TRI IDL TriSignatureIdType
package org.etsi.ttcn.tri;
public interface TriSignatureId {
    public String getSignatureName();
    public void setSignatureName(String sigName);
    public boolean equals(TriSignatureId sig);
}
```

6.3.2.8.1 方法

- getSignatureName()

返回特征标识符，如 TTCN-3 规范所定义的。

- setSignatureName(String sigName)

设置这一 TriSignatureId 的特征标识符为 sigName。

- equals(TriSignatureId sig)

比较 sig 与这一 TriSignatureId 是否相等。如果且仅如果两个特征具有相同的特征标识符，返回 true，否则返回 false。

6.3.2.9 TriParameterType

TriParameterType 被映射到下列接口：

```
// TRI IDL TriParameterType
package org.etsi.ttcn.tri;
public interface TriParameter {
    public String getParameterName();
    public void setParameterName(String name);
    public int getParameterPassingMode();
    public void setParameterPassingMode(in mode);
    public byte[] getEncodedParameter();
    public void setEncodedParameter(byte[] parameter);
}
```

6.3.2.9.1 方法

- getParameterName()

返回参数名称，如 TTCN-3 规范所定义的。

- setParameterName(String name)

设置这一 TriParameter 参数的名称为 name。

- getParameterPassingMode()

返回这一参数的参数标示模式。

- setParameterPassingMode(in mode)

设置这一 TriParameter 参数的参数模式为 mode。

- getEncodedParameter()

返回这一 TriParameter 的编码参数表述，如果参数包含惟一值 null，返回 null 对象（也见第 5.5.4.1 节）。

- setEncodedParameter(byte[] parameter)

设置这一 TriParameter 的编码参数表述为 parameter。如果必须设置惟一值 null 来指示这一参数未持有值，则 Java null 必须作为 parameter 传送（也见第 5.5.4.1 节）。

6.3.2.10 TriParameterPassingModeType

TriParameterPassingModeType 被映射到下列接口：

```
// TRI IDL TriParameterPassingModeType
package org.etsi.ttcn.tri;
public interface TriParameterPassingMode {
    public final static int TRI_IN = 0;
    public final static int TRI_INOUT = 1;
    public final static int TRI_OUT = 2;
}
```

6.3.2.10.1 常量

- TRI_IN

将被用于指示 TriParameter 是一个 in 参数。

- TRI_INOUT

将被用于指示 TriParameter 是一个 inout 参数。

- TRI_OUT

将被用于指示 TriParameter 是一个 out 参数。

6.3.2.11 TriParameterListType

TriParameterListType 被映射到下列接口：

```
// TRI IDL TriParameterListType
package org.etsi.ttcn.tri;
public interface TriParameterList {
    public int size();
    public boolean isEmpty();
    public java.util.Enumeration getParameters();
    public TriParameter get(int index);
    public void clear();
    public void add(TriParameter parameter);
}
```

6.3.2.11.1 方法

size()

返回这一列表中参数的数量。

isEmpty()

如果这一列表不包含参数，则返回 true 。

getParameters()

在列表的参数上返回 Enumeration。枚举按照参数在列表中出现的相同顺序提供参数。

get(int index)

在指定的位置返回 TriParameter 。

clear()

从这一 TriParameterList 中移除所有参数。

add(TriParameter parameter)

将 parameter 加入到这一 TriParameterList 的末尾。

6.3.2.12 TriExceptionType

TriExceptionType 被映射到下列接口：

```
// TRI IDL TriExceptionType
package org.etsi.ttcn.tri;
public interface TriException {
    public byte[] getEncodedException();
    public void setEncodedException(byte[] message);
    public boolean equals(TriException exc);
}
```

6.3.2.12.1 方法

- getEncodedException()

按照在 TTCN-3 规范中定义的编码规则返回编码的异常。

- setEncodedMessage(byte[] exc)

设置这一 TriException 的编码的异常表述为 exc。

- equals(TriException exc)

比较 exc 与这一 TriException 是否相等。如果且仅如果两个异常具有相同的编码表述，返回 true，否则返回 false。

6.3.2.13 TriTimerIdType

TriTimerIdType 被映射到下列接口：

```
// TRI IDL TriTimerIdType
package org.etsi.ttcn.tri;
public interface TriTimerId {
    public String getTimerName();
    public boolean equals(TriTimerId timer);
}
```

6.3.2.13.1 方法

- getTimerName()

返回这一定时器标识符的名称，如 TTCN-3 规范所定义的。在隐式定时器的情况下，结果是与实现相关的（见第 4.1.2 节）。

- equals(TriTimerId timer)

比较 timer 与这一 TriTimerId 是否相等。如果且仅如果两个定时器标识符表示同一个定时器，返回 true，否则返回 false。

6.3.2.14 TriTimerDurationType

TriTimerDurationType 被映射到下列接口：

```
// TRI IDL TriTimerDurationType
package org.etsi.ttcn.tri;
public interface TriTimerDuration {
    public double getDuration();
    public void setDuration(double duration);
    public boolean equals(TriTimerDuration duration);
}
```

6.3.2.14.1 方法

- getDuration()

返回定时器的时长为 double。

- setDuration(double duration)

设置这一 TriTimerDuration 的时长为 duration。

- equals(TriTimerDuration duration)

比较 duration 与这一 TriTimerDuration 是否相等。如果且仅如果二者具有相同的时长，返回 true，否则返回 false。

6.3.2.15 TriFunctionIdType

TriFunctionIdType 被映射到下列接口：

```
// TRI IDL TriFunctionIdType
package org.etsi.ttcn.tri;
public interface TriFunctionId {
    public String toString();
    public String getFunctionName();
    public boolean equals(TriFunctionId fun);
}
```

6.3.2.15.1 方法

- toString()

返回函数的串表述，如 TTCN-3 规范所定义的。

- getFunctionName()

返回函数标识符，如 TTCN-3 规范所定义的。

- equals(TriFunctionId fun)

比较 fun 与这一 TriFunctionId 是否相等。如果且仅如果两个函数具有相同的函数标识符，返回 true，否则返回 false。

6.3.2.16 TriTestCaseIdType

TriTestCaseIdType 被映射到下列接口：

```
// TRI IDL TriTestCaseIdType
package org.etsi.ttcn.tri;
public interface TriTestCaseId {
    public String toString();
    public String getTestCaseName();
    public boolean equals(TriTestCaseId tc);
}
```

6.3.2.16.1 方法

- toString()

返回测试案例的串表述，如 TTCN-3 规范所定义的。

- getTestCaseName()

返回测试案例标识符，如 TTCN-3 规范所定义的。

- equals(TriTestCaseId tc)

比较 tc 与这一 TriTestCaseId 是否相等。如果且仅如果两个测试案例具有相同的测试案例标识符，返回 true，否则返回 false。

6.3.2.17 TriActionTemplateType

废弃。

6.3.2.18 TriStatusType

TriStatusType 被映射到下列接口：

```
// TriStatusType
package org.etsi.ttcn.tri;
public interface TriStatus {
    public final static int TRI_OK = 0;
    public final static int TRI_ERROR = -1;
    public String toString();
    public int getStatus();
    public void setStatus(int status);
    public boolean equals(TriStatus status);
}
```

6.3.2.18.1 方法

- toString()

返回状态的串表述。

- getStatus()

返回这一 TriStatus 的状态。

- setStatus(int status)

设置这一 TriStatus 的状态。

- equals(TriStatus status)

比较 status 与这一 TriStatus 是否相等。如果且仅如果它们具有相同的状态，返回 true，否则返回 false。

6.4 常量

在这一 Java 语言内，已经规定了映射常量。所有常量都被定义为 public final static，并可从每个包的每个对象获得。在这一节中定义的常量未在 IDL 章节中定义。替代地，它们由 TRI IDL 类型的规范标记为本机类型。

下列常量可被用于确定 TTCN-3 参数的参数标示模式（也见第 6.3.2.10 节）。

- org.etsi.ttcn.tri.TriParameterPassingMode.TRI_IN；
- org.etsi.ttcn.tri.TriParameterPassingMode.TRI_INOUT；
- org.etsi.ttcn.tri.TriParameterPassingMode.TRI_OUT。

这些常量的实例的值必须反映 TTCN-3 过程特征中定义的参数标示模式。

对于惟一参数值 `null`，编码的参数值必须被设置为 Java `null`。

下列常量必须被用于指示方法局部成功（也见第 6.3.2.18 节）：

- org.etsi.ttcn.tri.TriStatus.TRI_OK；
- org.etsi.ttcn.tri.TriStatus.TRI_ERROR。

6.5 接口的映射

TRI IDL 定义定义了两个接口，即 **`triCommunication`** 和 **`triPlatform`** 接口。由于操作定义用于这一接口内的不同方向，即某些操作仅可由系统适配实体上的 TTCN-3 执行实体（TE）调用，而其他仅可由 TE 上的 SA 调用。这可通过将 TRI IDL 接口分成两个子接口来反映，每个由被调用实体下标。

表 4/Z.144－子接口

调用/被调用	TE	SA	PA
TE	-	`TriCommunicationSA`	`triPlatformPA`
SA	`TriCommunicationTE`	-	-
PA	`TriPlatformTE`	-	-

所有在这些接口中定义的方法应按第 5 节中规定的运作。

6.5.1 Out和InOut参数标示模式

下列 IDL 类型在 `out` 或 `inout` 参数标示模式中使用：

- TriParameter。
- TriParameterList。
- TriBoolean。
- TriTimerDuration。

在它们在 `out` 或 `inout` 参数标示模式中使用的情况下，各个分类的对象将用方法调用标示。然后被调用实体可获取设置返回值的方法。

6.5.2 triCommunication — 接口

`triCommunication` 接口被分为两个接口，即定义从 TE 到 SA 的调用的 `triCommunicationSA` 接口和定义从 SA 到 TE 的调用的 `triCommunicationTE` 接口。

6.5.2.1 triCommunicationSA

`triCommunicationSA` 接口被映射到下列接口：

```
// TriCommunication
// TE -> SA
package org.etsi.ttcn.tri;
public interface TriCommunicationSA {
    // 重置操作
    // 参考：TRI 定义 5.5.1
    TriStatus triSAReset();

    // 连接处理操作
    // 参考：TRI 定义 5.5.2.1
    public TriStatus triExecuteTestCase(TriTestCaseId
        testCaseId,TriPortIdList tsiPorts);
    // 参考：TRI 定义 5.5.2.2
```

```
    public TriStatus triMap(TriPortId compPortId, TriPortId tsiPortId);
    // 参考: TRI 定义 5.5.2.3
    public TriStatus triUnmap(TriPortId compPortId, TriPortId tsiPortId);

    // 基于消息的通信操作
    // 参考: TRI 定义 5.5.3.1
    public TriStatus triSend(TriComponentId componentId, TriPortId tsiPortId,
            TriAddress sutAddress, TriMessage sendMessage);
    // 参考: TRI 定义 5.5.3.2
    public TriStatus triSendBC(TriComponentId componentId, TriPortId tsiPortId,
            TriMessage sendMessage);
    // 参考: TRI 定义 5.5.3.3
    public TriStatus triSendMC(TriComponentId componentId, TriPortId tsiPortId,
            TriAddressList addresses, TriMessage sendMessage);

    // 基于过程的通信操作
    // 参考: TRI 定义 5.5.4.1
    public TriStatus triCall(TriComponentId componentId,
            TriPortId tsiPortId, TriAddress sutAddress,
            TriSignatureId signatureId, TriParameterList parameterList);
    // 参考: TRI 定义 5.5.4.2
    public TriStatus triCallBC(TriComponentId componentId,
            TriPortId tsiPortId,
            TriSignatureId signatureId, TriParameterList parameterList);
    // 参考: TRI 定义 5.5.4.3
    public TriStatus triCallMC(TriComponentId componentId,
            TriPortId tsiPortId, TriAddressList sutAddresses,
            TriSignatureId signatureId, TriParameterList parameterList);

    // 参考: TRI 定义 5.5.4.4
    public TriStatus triReply(TriComponentId componentId,
            TriPortId tsiPortId, TriAddress sutAddress,
            TriSignatureId signatureId, TriParameterList parameterList,
            TriParameter returnValue);
    // 参考: TRI 定义 5.5.4.5
    public TriStatus triReplyBC(TriComponentId componentId,
            TriPortId tsiPortId,
            TriSignatureId signatureId, TriParameterList parameterList,
            TriParameter returnValue);
    // 参考: TRI 定义 5.5.4.6
    public TriStatus triReplyMC(TriComponentId componentId,
            TriPortId tsiPortId, TriAddressList sutAddresses,
            TriSignatureId signatureId, TriParameterList parameterList,
            TriParameter returnValue);

    // 参考: TRI 定义 5.5.4.7
    public TriStatus triRaise(TriComponentId componentId, TriPortId tsiPortId,
            TriAddress sutAddress,
            TriSignatureId signatureId,
            TriException exc);
    // 参考: TRI 定义 5.5.4.8
    public TriStatus triRaiseBC(TriComponentId componentId, TriPortId tsiPortId,
            TriSignatureId signatureId,
            TriException exc);
    // 参考: TRI 定义 5.5.4.9
    public TriStatus triRaiseMC(TriComponentId componentId,TriPortId tsiPortId,
            TriAddresses sutAddresses,
            TriSignatureId signatureId,
            TriException exc);

    // 杂项操作
    // 参考: TRI 定义 5.5.5.1
    public TriStatus triSutActionInformal(String description);

}
```

6.5.2.2 triCommunicationTE

triCommunicationTE 接口被映射到下列接口：

```
// TriCommunication
// SA -> TE
package org.etsi.ttcn.tri;
public interface TriCommunicationTE {
    // 基于消息的通信操作
    // 参考: TRI 定义 5.5.3.4
    public void triEnqueueMsg(TriPortId tsiPortId,
            TriAddress sutAddress, TriComponentId componentId,
            TriMessage receivedMessage);
```

```
    // 基于过程的通信操作
    // 参考：TRI 定义 5.5.4.10
    public void triEnqueueCall(TriPortId tsiPortId,
            TriAddress SUTaddress, TriComponentId componentId,
            TriSignatureId signatureId, TriParameterList parameterList );

    // 参考：TRI 定义 5.5.4.11
    public void triEnqueueReply(TriPortId tsiPortId, TriAddress sutAddress,
            TriComponentId componentId, TriSignatureId signatureId,
            TriParameterList parameterList, TriParameter returnValue);

    // 参考：TRI 定义 5.5.4.12
    public void triEnqueueException(TriPortId tsiPortId,
            TriAddress sutAddress, TriComponentId componentId,
        TriSignatureId signatureId, TriException exc);
}
```

6.5.3 triPlatform — 接口

triPlatform 接口被分为两个接口，即定义从 TE 到 PA 的调用的 triPlatformPA 接口和定义从 PA 到 TE 的调用的 triPlatformTE 接口。

6.5.3.1 TriPlatformPA

triPlatformPA 接口被映射到下列接口：

```
// TriPlatform
// TE -> PA
package org.etsi.ttcn.tri;
public interface TriPlatformPA {
    // 参考：TRI 定义 5.6.1
    public TriStatus triPAReset();

    // 定时器处理操作
    // 参考：TRI 定义 5.6.2.1
    public TriStatus triStartTimer(TriTimerId timerId,
        TriTimerDuration timerDuration);

    // 参考：TRI 定义 5.6.2.2
    public TriStatus triStopTimer(TriTimerId timerId);

    // 参考：TRI 定义 5.6.2.3
    public TriStatus triReadTimer(TriTimerId timerId,
        TriTimerDuration elapsedTime);

    // 参考：TRI 定义 5.6.2.4
    public TriStatus triTimerRunning(TriTimerId timerId,
        TriBoolean running);

    // 杂项操作

    // 参考：TRI 定义 5.6.3.1
    public TriStatus triExternalFunction(TriFunctionId functionId,
     TriParameterList parameterList, TriParameter returnValue);
}
```

6.5.3.2 TriPlatformTE

triPlatformTE 接口被映射到下列 Java 接口：

```
// TriPlatform
// PA -> TE
package org.etsi.ttcn.tri;
public interface TriPlatformTE {
    // 参考：TRI 定义 5.6.2.5
    public void triTimeout(TriTimerId timerId);
}
```

6.6 可选参数

第 5.4 节定义了保留的值必须被用于指示没有可选参数。对于 Java 语言映射，Java null 值必须被用于指示没有可选值。例如，如果在 triSend 操作中，地址参数必须被省略，则操作调用必须是 triSend(componentId, tsiPortId, null, sendMessage)。

6.7 TRI初始化

所有方法都是非静态的，即操作仅可在对象上调用。由于这一建议书不定义 TE、SA 和 PA 的具体实现策略，所以 TE、SA 和 PA 借以开始知道有关各个对象的处理的机制超出了本建议书的范围。

工具供货商必须为 SA 和 PA 的开发商提供方法来将 TE、SA 和 PA 注册到它们各自的通信伙伴。

6.8 错误处理

与第 5.2 节所定义的错误处理相比，在这一 Java 语言映射内未定义另外的错误处理。特别的，未定义异常处理机制。

7 ANSI-C语言映射

7.1 引言

本节为在第 5.3 节中规定的抽象数据类型定义了 TRI ANSI-C 语言映射。对于基本 IDL 类型，映射遵循 OMG 建议书。

7.2 名称和范围

C 参数标识符必须以小写字母开头，构建参数标识符的随后部分将以大写字母开头。例如 IDL 参数 `SUTaddress` 映射为 C 语言中的 `sutAddress` 。

C 中的抽象数据类型省略了 IDL 定义中使用的拖尾 `Type`。例如 IDL 类型 `TriPortIdType` 映射为 C 语言中的 `TriPortId`。

较早的 C 规范已经限制标识符的惟一性为最重要的 8 个字符。然而，新近的 ANSI-C 规范已经将此限制改为 31 个最重要字符。除了这一发布，在这一映射中尚未标识命名或范围冲突。

7.2.1 抽象类型映射

TRI ADT	ANSI-C 表述	注释和注解
`TriAddress`	`BinaryString`	

`TriAddressList`	`typedef struct TriAddressList`
	`{`
	`TriAddress** addrList;`
	`long int length;`
	`} TriAddressList;`
		注 — 没有特殊的值标记 `addrList[]` 的末尾。`Length` 字段必须被用于正确地遍历这一数组。

`TriComponentId`	`typedef struct TriComponentId`
	`{`
	`BinaryString compInst;`
	`String compName;`
	`QualifiedName compType;`
	`} TriComponentId;`
		注 — `compInst` 用于组件实例。

`TriComponentIdList`	`typedef struct TriComponentIdList`
	`{`
	`TriComponentId** compIdList;`
	`long int length;`
	`} TriComponentIdList;`
		注 — 没有特殊的值标记 `compIdList[]` 的末尾。`Length` 字段必须被用于正确地遍历这一数组。

`TriException`	`BinaryString`	
`TriFunctionId`	`QualifiedName`	
`TriMessage`	`BinaryString`	

TRI ADT	ANSI-C 表述	注释和注解
TriParameterList	typedef struct TriParameterList { TriParameter** parList; long int length; } TriParameterList;	注 — 没有特殊的值标记 parList 的末尾。Length 字段必须被用于正确地遍历这一数组。
TriParameter	typedef struct TriParameter { BinaryString par; TriParameterPassingMode mode; } TriParameter;	
TriParameterPassingMode	typedef enum { TRI_IN = 0, TRI_INOUT = 1, TRI_OUT = 2 } TriParameterPassingMode;	注 — 这一类型的实例的值必须反映在相应 TTCN-3 过程特征中定义的参数标示模式。
TriPortIdList	typedef struct TriPortIdList { TriPortId** portIdList; long int length; } TriPortIdList;	注 — 没有特殊的值标记 portIdList[] 的末尾。Length 字段必须被用于正确地遍历这一数组。
TriPortId	typedef struct TriPortId { TriComponentId compInst; char* portName; long int portIndex; QualifiedName portType; void* aux; } TriPortId;	注 1 — compInst 用于组件实例。 注 2 — 对于单数（非数组）声明。portIndex 值应为 –1。 注 3 — aux 字段用于 TRI 功能性的未来可扩展性。
TriSignatureId	QualifiedName	
TriStatus	long int #define TRI_ERROR -1 #define TRI_OK 0	注 — 所有的赋值保留用于 TRI 功能性的未来可扩展性。
TriTestCaseId	QualifiedName	
TriTimerDuration	Double	
TriTimerId	BinaryString	注 — 关于定时器的悬而未决的声明和快照语义可能影响将来的表述。

7.2.2 ANSI-C类型定义

C ADT	类 型 定 义	注释和注解
BinaryString	typedef struct BinaryString { unsigned char* data; long int bits; void* aux; } BinaryString;	注 1 — data 是非空终止串。 注 2 — bits 是在数据中使用的比特数。比特值–1 用于指示省略的值。 注 3 — aux 字段用于 TRI 功能性的未来可扩展性。
QualifiedName	typedef struct QualifiedName { char* moduleName; char* objectName; void* aux; } QualifiedName;	注 1 — moduleName 和 objectName 字段是字面意义上的 TTCN-3 标识符。 注 2 — aux 字段用于 TRI 功能性的未来可扩展性。

7.2.3 IDL类型映射

IDL类型	ANSI-C表述	注释和注解
Boolean	unsigned char	从 OMG IDL 到 C++ 映射
String	char*	从 OMG IDL 到 C++ 映射

7.2.4 TRI 操作映射

IDL表述	ANSI-C表述
TriStatusType triSAReset()	TriStatus triSAReset()
TriStatusType triExecuteTestCase (in TriTestCaseIdType testCaseId, in TriPortIdListType tsiPortList)	TriStatus triExecuteTestCase (const TriTestCaseId* testCaseId, const TriPortIdList* tsiPortList)
TriStatusType triMap (in TriPortIdType compPortId, in TriPortIdType tsiPortId)	TriStatus triMap (const TriPortId* compPortId, const TriPortId* tsiPortId)
TriStatusType triUnmap (in TriPortIdType compPortId, in TriPortIdType tsiPortId)	TriStatus triUnmap (const TriPortId* compPortId, const TriPortId* tsiPortId)
TriStatusType triSend (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriMessageType sendMessage)	TriStatus triSend (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriMessage* sendMessage)
TriStatusType triSendBC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriMessageType sendMessage)	TriStatus triSendBC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriMessage* sendMessage)
TriStatusType triSendMC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressListType SUTaddresses, in TriMessageType sendMessage)	TriStatus triSendMC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddressList* sutAddresses, const TriMessage* sendMessage)
void triEnqueueMsg (in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriComponentIdType componentId, in TriMessageType receivedMessage)	void triEnqueueMsg (const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriComponentId* componentId, const TriMessage* receivedMessage)
TriStatusType triCall (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriSignatureIdType signatureId, in TriParameterListType parameterList)	TriStatus triCall (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriSignatureId* signatureId, const TriParameterList* parameterList)
TriStatusType triCallBC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriSignatureIdType signatureId, in TriParameterListType parameterList)	TriStatus triCallBC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriSignatureId* signatureId, const TriParameterList* parameterList)
TriStatusType triCallMC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressListType SUTaddresses, in TriSignatureIdType signatureId, in TriParameterListType parameterList)	TriStatus triCallMC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddressList* sutAddresses, const TriSignatureId* signatureId, const TriParameterList* parameterList)

IDL表述	ANSI-C表述
TriStatusType triReply (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)	TriStatus triReply (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriSignatureId* signatureId, const TriParameterList* parameterList, const TriParameter* returnValue)
TriStatusType triReplyBC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)	TriStatus triReplyBC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriSignatureId* signatureId, const TriParameterList* parameterList, const TriParameter* returnValue)
TriStatusType triReplyMC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressListType SUTaddresses, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)	TriStatus triReplyMC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddressList* sutAddresses, const TriSignatureId* signatureId, const TriParameterList* parameterList, const TriParameter* returnValue)
TriStatusType triRaise (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriSignatureIdType signatureId, in TriExceptionType exc)	TriStatus triRaise (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriSignatureId* signatureId, const TriException* exception)
TriStatusType triRaiseBC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriSignatureIdType signatureId, in TriExceptionType exc)	TriStatus triRaiseBC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriSignatureId* signatureId, const TriException* exception)
TriStatusType triRaiseMC (in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressListType SUTaddresses, in TriSignatureIdType signatureId, in TriExceptionType exc)	TriStatus triRaiseMC (const TriComponentId* componentId, const TriPortId* tsiPortId, const TriAddressList* sutAddresses, const TriSignatureId* signatureId, const TriException* exception)
void triEnqueueCall (in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriComponentId componentId, in TriSignatureIdType signatureId, in TriParameterListType parameterList)	void triEnqueueCall (const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriComponentId* componentId, const TriSignatureId* signatureId, const TriParameterList* parameterList)
void triEnqueueReply (in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriComponentIdType componentId, in TriSignatureIdType signatureId, in TriParameterListType parameterList, in TriParameterType returnValue)	void triEnqueueReply (const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriComponentId* componentId, const TriSignatureId* signatureId, const TriParameterList* parameterList, const TriParameter* returnValue)

IDL表述	ANSI-C表述
void triEnqueueException (in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriComponentIdType componentId, in TriSignatureIdType signatureId, in TriExceptionType exc)	void triEnqueueException (const TriPortId* tsiPortId, const TriAddress* sutAddress, const TriComponentId* componentId, const TriSignatureId* signatureId, const TriException* exception)
TriStatusType triSUTActionInformal (in string description)	TriStatus triSUTActionInformal (const char* description)
TriStatusType triPAReset()	TriStatus triPAReset()
TriStatusType triStartTimer (in TriTimerIdType timerId, in TriTimerDurationType timerDuration)	TriStatus triStartTimer (const TriTimerId* timerId, TriTimerDuration timerDuration)
TriStatusType triStopTimer (in TriTimerIdType timerId)	TriStatus triStopTimer (const TriTimerId* timerId)
TriStatusType triReadTimer (in TriTimerIdType timerId, out TriTimerDurationType elapsedTime)	TriStatus triReadTimer (const TriTimerId* timerId, TriTimerDuration* elapsedTime)
TriStatusType triTimerRunning (in TriTimerIdType timerId, out boolean running)	TriStatus triTimerRunning (const TriTimerId* timerId, unsigned char* running)
void triTimeout (in TriTimerIdType timerId)	void triTimeout (const TriTimerId* timerId)
TriStatusType triExternalFunction (in TriFunctionIdType functionId, inout TriParameterListType parameterList, out TriParameterType returnValue)	TriStatus triExternalFunction (const TriFunctionId* functionId, TriParameterList* parameterList, TriParameter* returnValue)

7.3 内存管理

废弃。

7.4 错误处理

对这一映射尚未定义错误处理。

8 使用方案

本节包含可以帮助 TRI 用户和工具供货商提供对本建议书内定义的操作语义的 TRI 理解的方案。

按照消息序列图(MSC)定义了 3 种方案。方案由 TTCN-3 编码片段构成，该片段使用到 SUT 的 TTCN-3 函数以及定时处理函数。MSC 示出 TE、SA 和 PA 实体与 SUT 之间的交互。

请注意，TTCN-3 片段不完整，因为片段的主要指标是动态行为的使用。所有现有的方案使用图 2 中示出的 TRI 操作的公共前导序列。

注意在本节中出现的 MSC 使用消息对对每个 TRI 操作建模。MSC 消息 triMap 后随 triMapOK 表示，例如，TRI 操作 triMap 已经被 TE 调用，它成功地从 SA 返回。TRI 操作调用使用抽象类型和值示出，旨在仅出于阐述目的服务。在特殊目标语言中的这些参数的具体表述在各语言映射中定义。

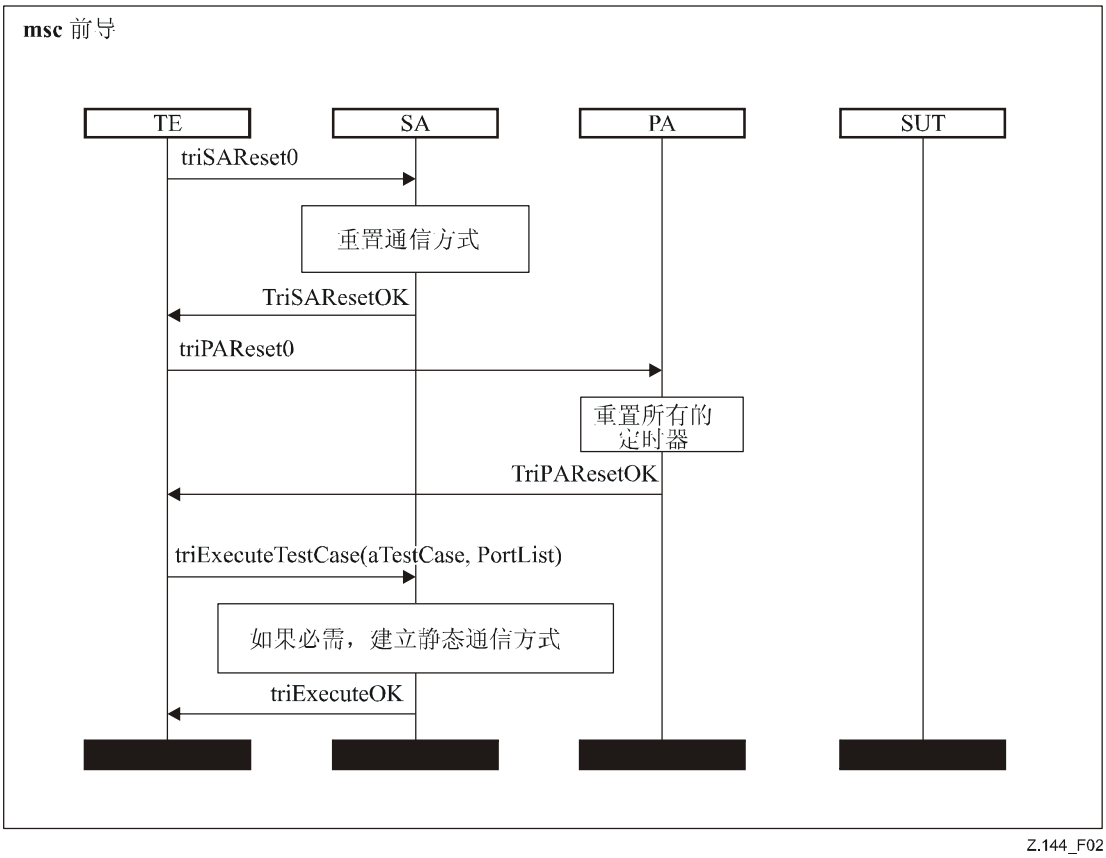

图 2/Z.144－公共MSC前导

8.1 第一种方案

第一种方案示出 TTCN-3 定时器操作（即启动和定时器运行）、基于消息的通信操作（即发送和接收）以及连接处理操作（即映射和去映射）。

8.1.1 TTCN-3片段

```
module triScenario1
{
 external function MyFunction();

 type port PortTypeMsg message { inout integer }

 type component MyComponent {
    port PortTypeMsg MyPort;
    timer MyTimer
 }

 type component MyTSI {
    port PortTypeMsg PCO1;
 }

 testcase scenario1() runs on MyComponent system MyTSI
 {
    MyPort.clear;
    MyPort.start;
    MyTimer.start(2);

    map(MyComponent: MyPort, system: PCO1);
    MyPort.send (integer : 5);
    if (MyTimer.running)
    {
     MyPort.receive(integer:7);
    }
    else
    {
     MyFunction();
    }
    unmap(MyComponent: MyPort, system:PCO1);
```

```
    MyPort.stop;
 }


 control {
    execute( scenario1() );
 }

}
```

8.1.2 消息序列图

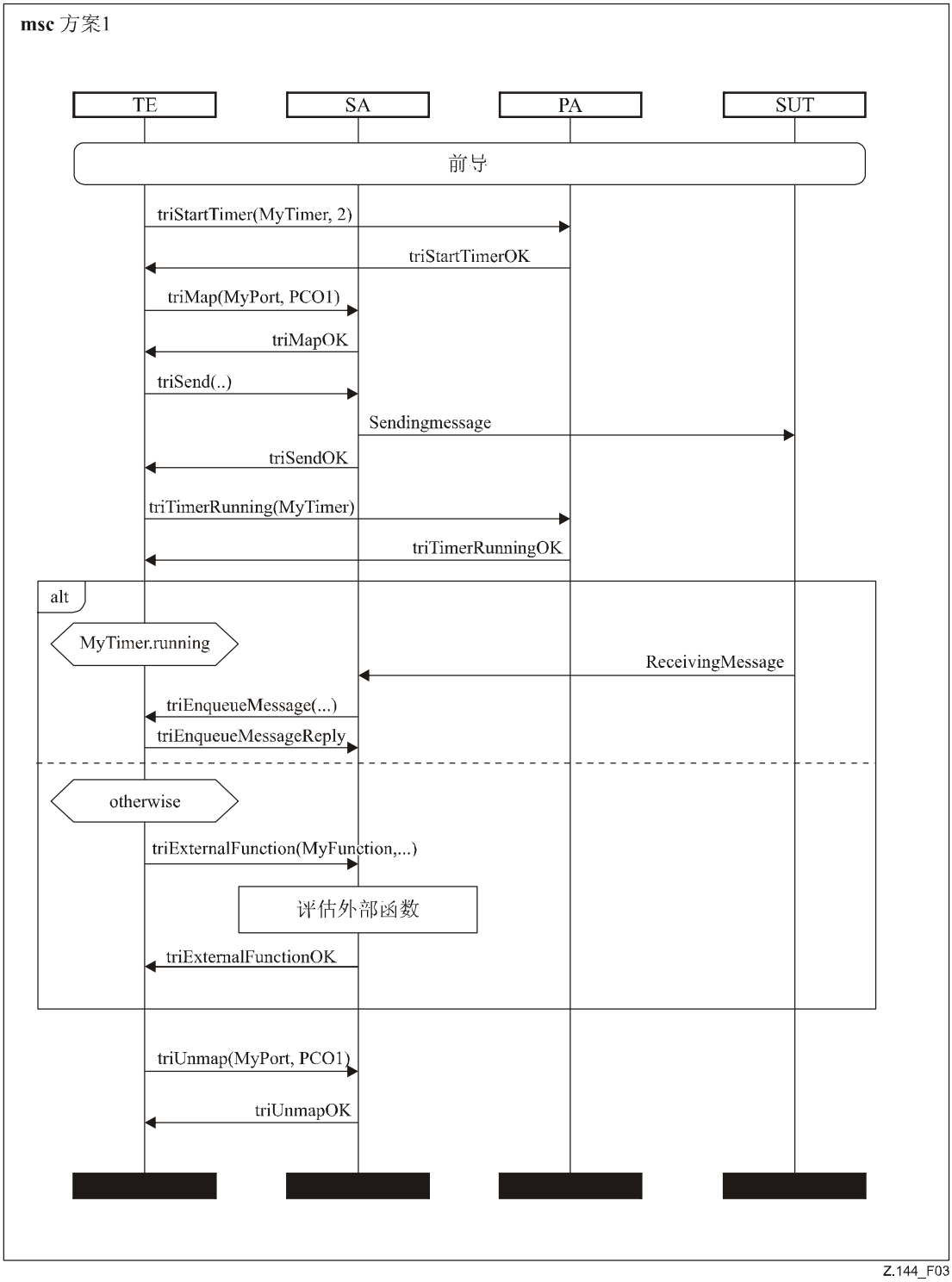

图 3/Z.144－使用方案1

8.2 第二种方案

第二个例子示出类似的方案，它也使用由测试组件 MyComponent 初始化的定时的基于过程的通信操作。在这一例子中，假定 MyComponent 作为 MTC 运行。

8.2.1 TTCN-3片段

```
module triScenario2
{

 signature MyProc ( in float par1, inout float par2)
    exception(MyExceptionType);

 type record MyExceptionType { FieldType1 par1, FieldType2 par2 }

 type port PortTypeProc procedure { out MyProc }

 type component MyComponent {
    port PortTypeProc MyPort;
    timer MyTimer = 7
 }

 testcase scenario2() runs on MyComponent
 {
    var float MyVar;

    MyPort.clear;
    MyPort.start;
    MyTimer.start;

 MyVar := MyTimer.read;

 if (MyVar>5.0) {
     MyPort.call (MyProc:{MyVar, 5.7}, 5);
     alt {
     [] MyPort.getreply(MyProc:{-,MyVar*5}) {}
     [] MyPort.catch (MyProc, MyExceptionType:* ) {}
     [] MyPort.catch (timeout) {}
     }
    }
MyTimer.stop;
    MyPort.stop;
 }

 control {
    execute( scenario2() );
 }

}
```

8.2.2 消息序列图

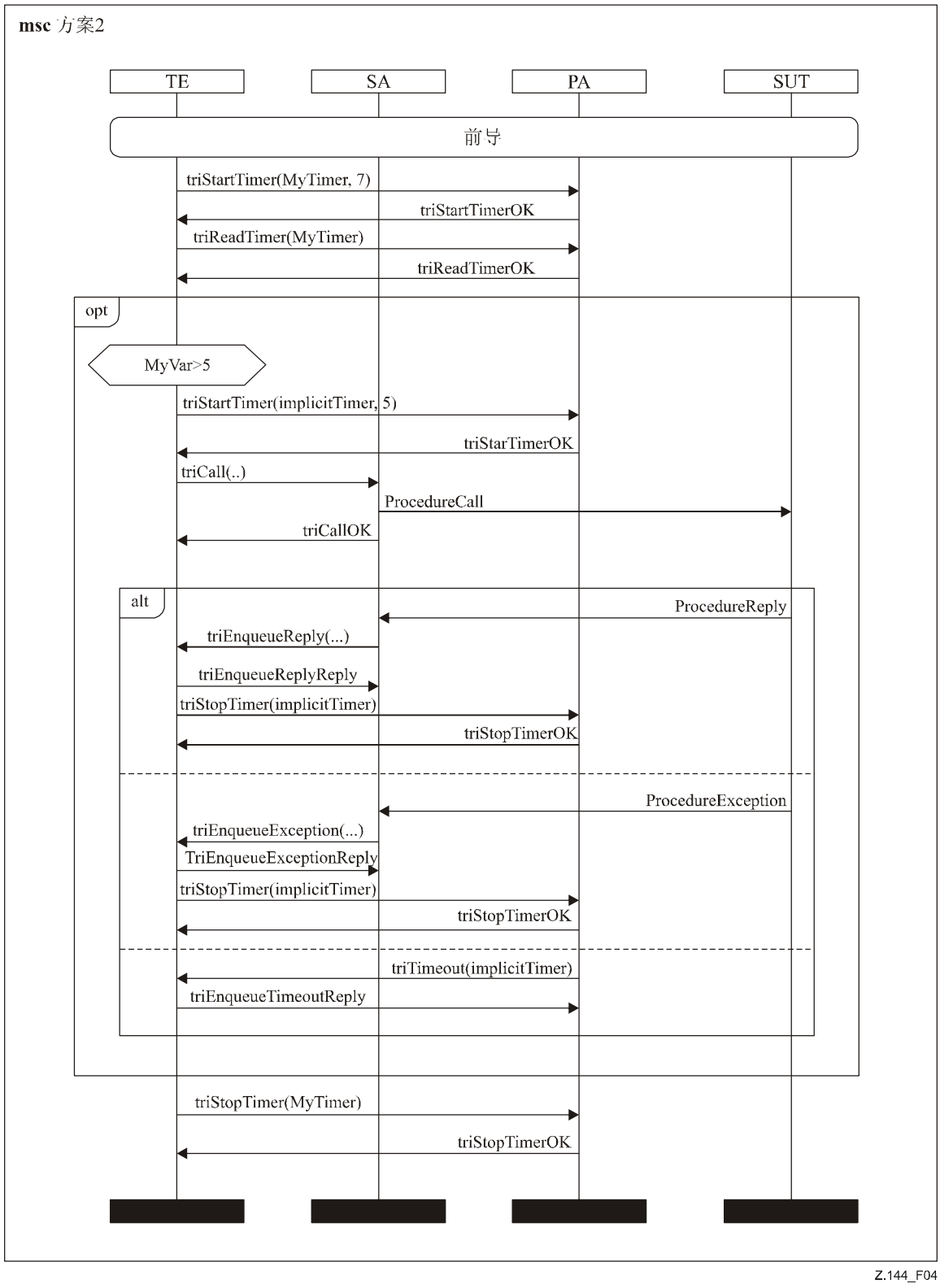

图 4/Z.144－使用方案2

8.3 第三种方案

使用第三种方案示出过程的异常以及基于这一接收的调用的异常的回答和提出。再次假定 MyComponent 作为 MTC 运行。FieldType1, FieldType2、p1 和 p2 假定在别处定义。

8.3.1 TTCN-3片段

```
module triScenario3
{
 signature MyProc ( in float par1, inout float par2)
    exception(MyExceptionType);

 type record MyExceptionType { FieldType1 par1, FieldType2 par2 }

 type port PortTypeProc procedure { in MyProc }

 type component MyComponent {
    port PortTypeProc MyPort;
    timer MyTimer = 3
 }

 testcase scenario3(integer x) runs on MyComponent
 {
    MyPort.start;
    MyTimer.start;
    alt
    {
     [] MyPort.getcall(MyProc:{5.0, 6.0})
     {
     MyTimer.stop;
     }
     [x>5] MyTimer.timeout
     {
     MyPort.reply(MyProc:{-, 30.0});
     }
     [x<=5] MyTimer.timeout
     {
     MyPort.raise(MyProc, MyExceptionType:{p1, p2} );
     }
    }
    MyPort.stop;
 }

 control {
    execute( scenario3(4) );
 }
}
```

8.3.2 消息序列图

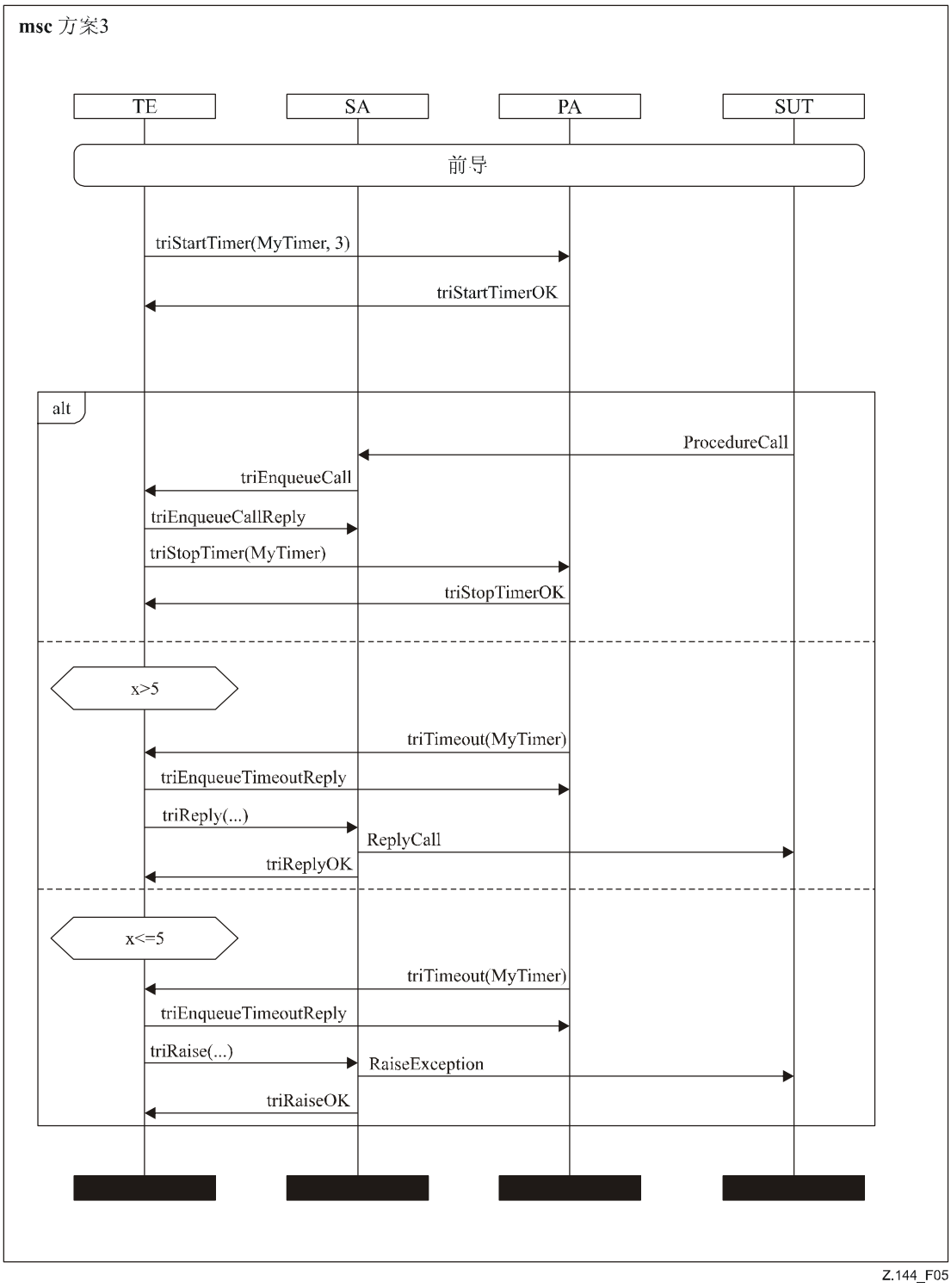

图 5/Z.144－使用方案3

附件A（规范性）

IDL摘要

这一附件概述了 TRI 操作的 IDL 定义，如第 5 节中定义的。

```
// ********************************************************************************
// TTCN-3 运行时间接口的接口定义

// ********************************************************************************

module triInterface
{

 //
 // ********************************************************************************
 // 类型
 // ********************************************************************************
 //

 // 连接
 native TriPortIdType;
 typedef sequence<TriPortIdType> TriPortIdListType;
 native TriComponentIdType;
 typedef sequence<TriComponentIdType> TriComponentIdListType;

 // 通信
 native TriMessageType;
 native TriAddressType;
 typedef sequence<TriAddressType> TriAddressListType;
 native TriSignatureIdType;
 native TriParameterType;
 typedef sequence<TriParameterType> TriParameterListType;
 native TriExceptionType;

 // 定时器
 native TriTimerIdType;
 native TriTimerDurationType;

 // 杂项
 native TriFunctionIdType;
 native TriTestCaseIdType;
 native TriStatusType;

 //
 // ********************************************************************************
 // 接口
 // ********************************************************************************
 //

 //
 // ********************************************************************************
 // 通信接口(参考：TRI 定义：5.5)
 // ********************************************************************************
 //
 interface triCommunication
 {

    // 重置操作

    // 参考：TRI 定义 5.5.1
    TriStatusType triSAReset();


    // 连接处理操作

    // 参考：TRI 定义 5.5.2.1
    TriStatusType triExecuteTestCase(in TriTestCaseIdType testCaseId,
    in TriPortIdListType tsiPortList);

    // 参考：TRI 定义 5.5.2.2
    TriStatusType triMap(in TriPortIdType compPortId, in TriPortIdType tsiPortId);

    // 参考：TRI 定义 5.5.2.3
    TriStatusType triUnmap(in TriPortIdType compPortId, in TriPortIdType tsiPortId);
```

```
    // 基于消息的通信操作

    // 参考：TRI 定义 5.5.3.1
    TriStatusType triSend(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
    in TriAddressType SUTaddress, in TriMessageType sendMessage);
    // 参考：TRI 定义 5.5.3.2
    TriStatusType triSendBC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
    in TriMessageType sendMessage);
    // 参考：TRI 定义 5.5.3.3
    TriStatusType triSendMC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
    in TriAddressListType SUTaddresses, in TriMessageType sendMessage);

    // 参考：TRI 定义 5.5.3.4
    void triEnqueueMsg(in TriPortIdType tsiPortId , in TriAddressType SUTaddress,
     in TriComponentIdType componentId, in TriMessageType receivedMessage);


    // 基于过程的通信操作

    // 参考：TRI 定义 5.5.4.1
    TriStatusType triCall(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
    in TriAddressType SUTaddress, in TriSignatureIdType signatureId,
    in TriParameterListType parameterList);
    // 参考：TRI 定义 5.5.4.2
    TriStatusType triCallBC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
    in TriSignatureIdType signatureId,
    in TriParameterListType parameterList);
    // 参考：TRI 定义 5.5.4.3
    TriStatusType triCallMC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
    in TriAddressListType SUTaddresses, in TriSignatureIdType signatureId,
    in TriParameterListType parameterList);

    // 参考：TRI 定义 5.5.4.4
    TriStatusType triReply(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
    in TriAddressType SUTaddress, in TriSignatureIdType signatureId,
    in TriParameterListType parameterList, in TriParameterType returnValue );
    // 参考：TRI 定义 5.5.4.5
    TriStatusType triReplyBC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
    in TriSignatureIdType signatureId,
    in TriParameterListType parameterList, in TriParameterType returnValue );
    // 参考：TRI 定义 5.5.4.6
    TriStatusType triReplyMC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
    in TriAddressListType SUTaddresses, in TriSignatureIdType signatureId,
    in TriParameterListType parameterList, in TriParameterType returnValue );

    // 参考：TRI 定义 5.5.4.7
    TriStatusType triRaise(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
    in TriAddressType SUTaddress, in TriSignatureIdType signatureId,
    in TriExceptionType exc);
    // 参考：TRI 定义 5.5.4.8
    TriStatusType triRaiseBC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
    in TriSignatureIdType signatureId,
    in TriExceptionType exc);
    // 参考：TRI 定义 5.5.4.9
    TriStatusType triRaiseMC(in TriComponentIdType componentId, in TriPortIdType tsiPortId,
    in TriAddressListType SUTaddresses, in TriSignatureIdType signatureId,
    in TriExceptionType exc);

    // 参考：TRI 定义 5.5.4.10
    void triEnqueueCall(in TriPortIdType tsiPortId, in TriAddressType SUTaddress,
    in TriComponentIdType componentId, in TriSignatureIdType signatureId,
    in TriParameterListType parameterList );

    // 参考：TRI 定义 5.5.4.11
    void triEnqueueReply(in TriPortIdType tsiPortId, in TriAddressType SUTaddress,
    in TriComponentIdType componentId, in TriSignatureIdType signatureId,
    in TriParameterListType parameterList, in TriParameterType returnValue );

    // 参考：TRI 定义 5.5.4.12
    void triEnqueueException(in TriPortIdType tsiPortId, in TriAddressType SUTaddress,
    in TriComponentIdType componentId, in TriSignatureIdType signatureId,
    in TriExceptionType exc);


    // 杂项操作

    // 参考：TRI 定义 5.5.5.1
    TriStatusType triSUTactionInformal(in string description);

};
```

```
//
// ***************************************************************************
// 平台接口(参考：TRI 定义：5.6)
// ***************************************************************************
//
interface triPlatform
{

   // 重置操作

   // 参考：TRI 定义 5.6.1
   TriStatusType triPAReset();


   // 定时器处理操作

   // 参考：TRI 定义 5.6.2.1
   TriStatusType triStartTimer(in TriTimerIdType timerId,
   in TriTimerDurationType timerDuration);

   // 参考：TRI 定义 5.6.2.2
   TriStatusType triStopTimer(in TriTimerIdType timerId);

   // 参考：TRI 定义 5.6.2.3
   TriStatusType triReadTimer(in TriTimerIdType timerId,
   out TriTimerDurationType elapsedTime);

   // 参考：TRI 定义 5.6.2.4
   TriStatusType triTimerRunning(in TriTimerIdType timerId, out boolean running);

   // 参考：TRI 定义 5.6.2.5
   void triTimeout(in TriTimerIdType timerId);


   // 杂项操作

   // 参考：TRI 定义 5.6.3.1
   TriStatusType triExternalFunction(in TriFunctionIdType functionId,
    inout TriParameterListType parameterList,
    out TriParameterType returnValue);

};
};
```

参考资料

– OMG CORBA (V2.2): *The Common Object Request Broker: Architecture and Specification, Section 3, February 1998.*

– INTOOL CGI/NPL038 (V2.2): *Generic Compiler/Interpreter interface*; GCI Interface Specification, Infrastructural Tools, December 1996.

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