[1. 如何描述测试用例？ 3](#_Toc391884293)

[2. 测试用例的语言 4](#_Toc391884294)

[2.1 测试准备、执行与检查 4](#_Toc391884295)

[2.2 测试准备 4](#_Toc391884296)

[2.2.1 拓扑与网元映射 4](#_Toc391884297)

[2.2.2 预定义过程 5](#_Toc391884298)

[2.2.3 工具协作 6](#_Toc391884299)

[2.3 测试执行 7](#_Toc391884300)

[2.3.1 开始执行测试 7](#_Toc391884301)

[2.3.2 执行完成的判定 7](#_Toc391884302)

[2.4 测试检查 8](#_Toc391884303)

[2.4.1 延迟检查 8](#_Toc391884304)

[2.4.2 测试报告 8](#_Toc391884305)

[2.4.3 测试检查项目的描述 8](#_Toc391884306)

[3. 测试工具的语言 8](#_Toc391884307)

[3.1 与被测系统交互 9](#_Toc391884308)

[3.2 过程控制 9](#_Toc391884309)

[3.3 工具协作 9](#_Toc391884310)

[4. 自动化测试的部署 9](#_Toc391884311)

[4.1 分布式部署 9](#_Toc391884312)

[4.1.1 分布式部署的优点 9](#_Toc391884313)

[4.1.2 本系统的部署 10](#_Toc391884314)

[4.2 Robot Framework 10](#_Toc391884315)

[4.3 关键字库 10](#_Toc391884316)

[4.4 测试工具 11](#_Toc391884317)

[5. 演示系统简介 11](#_Toc391884318)

[5.1 概要 11](#_Toc391884319)

[5.2 模拟测试工具 12](#_Toc391884320)

[5.2.1 特性 12](#_Toc391884321)

[5.2.2 框架 12](#_Toc391884322)

[5.3 模拟被测网元 12](#_Toc391884323)

[5.3.1 特性 12](#_Toc391884324)

[5.3.2 框架 13](#_Toc391884325)

[6. 几个实例 13](#_Toc391884326)

[6.1 Attach、CreateSession——顺序过程、分布式部署 13](#_Toc391884327)

[6.2 媒体面触发的配额下发——循环过程 13](#_Toc391884328)

[6.3 多次Attach——使用事件通讯、高级关键字 13](#_Toc391884329)

[6.4 OMM过程——另一种测试工具 13](#_Toc391884330)

[6.5 5个用户的Attach——数据驱动测试 13](#_Toc391884331)

[7. 待完善 13](#_Toc391884332)

[8. 后续扩展 13](#_Toc391884333)

[8.1 自动回复的模拟网元 13](#_Toc391884334)

[8.2 Monkey Test 14](#_Toc391884335)

# 如何描述测试用例？

测试用例描述的内容，当然是测试的操作步骤和目的。那么进一步的，如何描述测试步骤与目的？

对于我们的通信系统而言，最准确无误的，是各个接口的消息序列，能够描述清楚这些消息序列，我们就能没有歧义的描述一个测试的**过程**。因此我们已有的测试工具中使用消息来描述测试过程。

可是直接描述消息序列，就缺少了对测试的**目的**的直接说明，这时，要想了解测试目的，就必须通过分析消息序列与参数来判断。另外，测试过程中使用的任何过程，都需要从一条条消息、一个个参数的定义开始做起，测试用例的轮廓将淹没在琐碎的细节中。从这个角的考虑，为了易于理解测试，测试的描述应该更概括、更易懂。显然，使用自然语言书写的测试用例，能够满足这个要求，但是不可避免的，自然语言中会有大量的歧义、信息缺失等，无法直接被工具执行。

事实上，上述两种描述方式的优点，对于测试用例都是不可或缺的，正因为这样，我们往往需要同时输出文字版的测试用例和执行时的测试脚本，维护和同步工作量都比较大。本方案中，需要考虑的是将二者结合，使用一种类似自然语言的方式，简洁而精确的描述出测试用例的实现步骤，同时这些步骤都是可执行的。在测试技术中，这种方式称为**关键字驱动的测试方法**，也就是本文中描述的系统需要使用的测试方法。



一个简洁而精确的例子：

“**云长**……往豫山埋伏，等彼军至，放过休敌；……但看南面火起，可纵兵出击，就焚其粮草。

**翼德**……山谷中埋伏，只看南面火起，……向博望城旧屯粮草处纵火烧之。

**关平、刘封**……至初更兵到，便可放火矣。

……**赵云**，令为前部，不要赢，只要输。

**我**只坐守县城。（-\_-|||）

**主公**……便弃营而走；但见火起，即回军掩杀。

命**孙乾、简雍**……安排功劳簿伺候。”

这是一个描述分布式系统协同工作的例子，其中包括了：

* 各个独立的功能实体——关张赵等；
* 中心调度——诸葛；
* 通讯方式——放火；
* 步骤描述——各种行动、策略、时间；
* 报告统计——功劳簿；

这个例子中，不需要诸葛详细描述如何行军布阵等细节，只需要安排步骤与信号，自己坐守县城即可。其中关键在于有关张赵这些能够自主决策的将军，他们能够独立完成安排给自己的任务。

关键字驱动的测试方法中，也需要能够自理的工具来支撑。在测试用例中，可以使用工具能够理解的语言来描述操作步骤，并提供必要的关键数据，具体的执行过程及细节数据，交给专门的工具来负责即可。通过这样的方法，就能够使得测试用例概要而精确的描述测试的过程与目的。

# 测试用例的语言

## 测试准备、执行与检查

一个测试用例中，需要在测试开始前准备好各个测试步骤，安排好测试从开始到结束的各种操作，然后将安排好的操作分发给各个工具，由工具独立执行，最后上报测试报告给测试执行引擎，做测试结果检查。

从这个过程中可以看出，一个测试用例中有三个部分：准备、执行、检查。

**准备** 在测试用例中描述被测系统，设计整个测试过程的操作，安排各个测试工具中如何协调，定义测试过程中需要提取加工哪些数据等。

**执行** 测试执行引擎将准备好的测试脚本分解到各个工具中，由各个工具各自独立执行，并按照约定的信号互相协作。

**检查** 各个测试工具分别上报测试报告给执行引擎，测试引擎按照测试用例的描述检查测试报告中的数据是否通过测试。

在测试用例中，需要安排上述各个阶段的操作，这就需要编写测试用例使用的语言能够实现上面几个阶段中描述的各项内容。



## 测试准备

### 拓扑与网元映射

测试中设计到的各类网元，包括被测网元、模拟网元以及辅助测试网元，可以从两个不同的角度来描述：测试环境中存在的、测试预定义过程中需要的。

测试环境中存在的网元，就是……测试环境中存在的网元。如：2个MME、1个HSS、3个eNB等等。测试预定义过程中需要的网元，指的是在执行特定过程时，需要的网元。如Attach过程中，涉及到一个eNB、新旧两个MME、一个HSS、两个GW等。

可以看出，“存在的网元”与“需要的网元”的映射关系是要根据实际测试需求来定义的，因此测试用例中要提供语言来描述这样的映射关系。

另外，这两个角度的描述中，都需要知道哪些网元是真实的，哪些网元是模拟的，哪些又是被测对象，这些也是根据测试环境不同而变化的。所以在测试用例中要能够描述网元的属性，是真实还是模拟。



### 预定义过程

测试过程中，使用了很多消息，每个消息又有大量参数，这些消息与参数的排列组合将会是一个天文数字，不可穷举。但是实际测试时，往往只用到了一些固定的消息序列，通常也只需要设置其中少量参数，其他参数取默认值即可。

根据这一特征，我们可以将相对固定的流程在工具中固化下来，通过流程的参数来调整流程的走向，这样，在描述测试用例时，只需要指定流程名称和几个参数，工具就能够明确的得知需要使用哪些消息来构建测试。一个完整的测试用例，将通过顺序执行一系列预定义过程来实现。

但是，总有一些特殊场景下，需要设置不常使用的消息参数和消息序列，因此，在测试用例层面，也需要提供能够直接构建消息和参数的能力，以应对这些特殊的情况。

预定义过程应该在工具中完成定义，对外表现为一个名称和一组参数，通过名称和参数的取值，就能准确的构建出需要的参数序列。由于预定义过程实现的功能千差万别，不同的过程需要的参数也各不相同，因此每个过程都可以有自己个性化的控制参数。

举例如下：

PSTT提供过程：Authenticate UE，参数result、reason、source

在测试用例中：

|  |  |
| --- | --- |
| **测试用例描述** | **消息** |
| Authenticate UE | -> AuthenticationRequest <- AuthenticationResponse(success) |
| Authenticate UE result=fail source=ue reason=mac error | -> AuthenticationRequest <- AuthenticationResponse(mac error) |
| Authenticate UE result=fail source=cn reason=illegal ue | -> AuthenticationRequest <- AuthenticationResponse -> AuthenticationReject(illegal ue) |

### 工具协作

对于一些较复杂的测试场景，可能需要工具间进行协作，比如，某一过程需要等待另一个过程执行完成后才能开始，这就需要工具间具有通信的能力。

工具并没有关于过程间关系的知识，这些知识往往与具体的测试要求相关，因此工具间的通信，需要在准备测试用例阶段显式说明。在本方案中，某一个工具可能需要等待某一特定时间发生后，才能执行特定操作，当这个工具执行到等待事件时，将阻塞自己，等待事件发生；在另一工具中，执行了需要通知其他工具的步骤后，会主动发出一个事件，通过集中调度将这个事件广播到所有工具中；阻塞等待事件收到这一事件，判断事件的来源与内容，解除阻塞状态，继续执行。这样，就实现了工具间的协调功能。

这一过程如下图描述，其中各个工具，就是PSTT以及网管操作等工具，调度者由Robot Framework来担任。在测试用例中，这一过程通过Send Event关键字来完成，这一关键字中，同时指出事件的源和等待者，并指明事件内容。



## 测试执行

### 开始执行测试

在测试用例中编写的测试过程、工具协作等，都不是立即执行的，需要在测试过程安排完成后，显式的通知所有工具开始测试。因为只有这样各个工具得到的才是完整的测试脚本，所有的工具协作才能够顺畅的执行下去。

测试调度负责根据已经准备的测试步骤，将各个工具需要关心的内容分发给各个不同的工具。在测试中，必然有一个工具是测试的启动点，测试调度需要先将启动点之外的其他工具逐一启动，最后通知启动点开始执行，从而触发整个测试过程的执行。

测试执行一旦开始，测试调度就不需要关注测试过程了，所有的测试执行操作将由各个工具自动完成，测试调度只需要分发事件和收集报告即可。



### 执行完成的判定

本方案中，测试的完成以收到所有测试工具的测试报告为准。测试调度不参与工具的停止，每个工具都需要能够自动停止，这就要求工具在可能无限运行的地方，引入停止机制，保证能够自行停止。

测试调度不参与工具停止的控制，主要是为了避免调度过程与具体测试逻辑的耦合，如果测试调度主动控制工具停止，就要面对不同测试逻辑的要求，而这些与具体测试过程相关的要求，理应放到具体的测试过程中去，工具中的过程库就是一个比较合适的地方。因此本方案中，测试调度不主动停止测试工具。

另一方面，测试调度与测试工具间的传输在本方案中为可靠的，如果由于传输原因，引起报告丢失，造成测试过程无限运行，那么其他调度与工具间的消息也就存在丢失的可能，测试将无法顺畅执行，这些异常情况的处理在这里暂不考虑。为使测试树立执行，我们在部署网络时，应尽可能选择可靠的传输，以便简化工具设计，当确实出现传输异常时，也应优先从传输层面考虑如何修正，尽量不要在工具的实现中预防这些异常情况。

## 测试检查

### 延迟检查

测试执行过程中，测试工具应该尽最大可能将分派给自己的测试流程执行完成，工具按照测试用例要求记录、加工测试数据，但是不需要对数据合理性进行评判。测试工具将所有需要的数据记录在测试报告中，供测试引擎（Robot Framework）检查。

这样做的原因是，任何一个工具，都只能看到测试的一部分，因此工具没有足够的知识来得出测试结论，必须将所有工具的测试报告综合分析，才能够判断一个测试的成功与失败。另一方面，假如测试工具来判断测试结论，当发生预期以外的情况时，如果工具停止执行，将导致整个流程执行异常，测试后的清理与回收工作将非常困难。所以在本方案中，要求工具尽可能完整的执行业务流程，将测试结论的判定交给测试引擎来做。

测试工具记录的数据，不仅仅限于原始数据，也可以是加工过的数据。所有采集与分析的需求，都在测试用例中指定，工具负责执行。

### 测试报告

测试完成后，测试工具上报的测试报告应该有统一的格式。这一部分目前的方案中还没有深入分析，可能会使用xml这样通用的结构化描述，能够适用于不同的测试场景。

### 测试检查项目的描述

在目前的方案中，测试数据采集与分析项目还没有明确如何传递给测试工具，初步的设想是，将待采集的数据分为两类，一是必须采集的，如消息码流，另一种是可选的，如特定过程关注的信息，可以通过在预定义过程中增加控制参数来指定可选的数据分析要求。

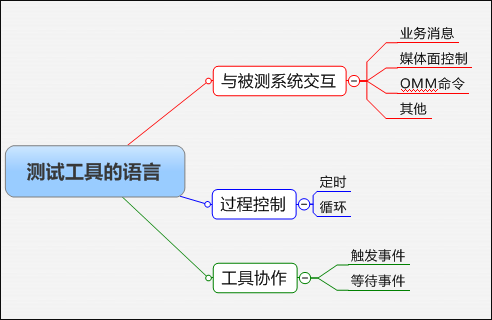
对于数据的检查，测试用例可以基于测试报告的结构框架，提供几个抽象的检查方法，通过携带不同的参数，来实现不同的检查目的。

在测试数据检查中，还需要考虑测试报告间的比对与印证。

# 测试工具的语言

在测试工具中，接收到来自测试调度的脚本与用户直接编写的测试用例并不相同，测试工具中的脚本是供工具使用的，已经细化到可执行层次的描述。

测试工具使用的脚本中，分为如下几个大类，包括：与被测系统交互、测试过程控制、工具协作。



## 与被测系统交互

业务消息：各种信令接口消息

媒体面控制：媒体面Profile参数控制，以及媒体面启动、停止的过程控制

OMM命令：OMM命令执行、OMM命令结果检查等

其他工具需求：对于特定的工具，可能需要的各种其他命令，如性能统计、计费等

## 过程控制

定时：测试过程中，需要测试脚本暂停指定的时间后继续执行

循环：测试过程中，需要测试脚本中某个段落循环执行

在本方案的过程控制中，没有引入条件判断。这样做的原因是，引入条件判断将会在一条测试用例中，增加多种可能的执行路径，而多种可能意味着测试结果不确定或者测试覆盖不全面，这与测试的目的不符，因此在本设计中，没有使用条件判断的控制逻辑。

## 工具协作

触发事件：测试工具主动上报某一事件

等待事件：测试工具需要等待某一事件发生后才能够继续执行

# 自动化测试的部署

## 分布式部署

### 分布式部署的优点

在本方案中，各种测试工具使用分布式的方式来部署，通过Robot Framework来集中调度。

使用分布式部署有如下优点：

* 各个工具间相互独立，可以使用各自最合适的方式，来实现各种需要的功能；
* 保持工具与测试调度间的接口不变的情况下，工具可以自由修改，没有兼容性问题；
* 测试调度的变更与修改，与测试工具无关；
* 可以根据实际的硬件资源情况，不同的工具可以复用硬件，减少成本；

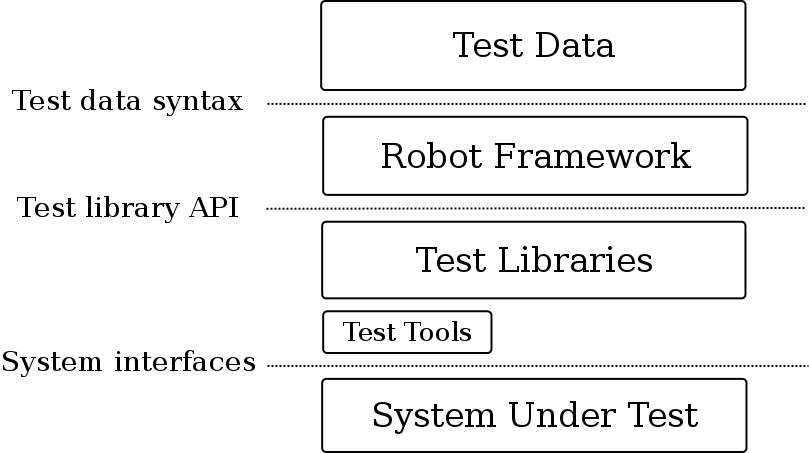
### 本系统的部署



## Robot Framework

Robot Framework是一款python编写的功能自动化测试框架，具备良好的可扩展性，支持关键字驱动。Robot Framework可以同时测试多种类型的客户端或者接口，可以进行分布式测试执行。Robot Framework是开源软件，由Nokia Siemens Networks开发并提供支持，该工具与2005年诞生，2008年发布2.0版本，并开放源代码。

Robot Framework的架构如下图所示，该框架读取测试数据（测试用例），通过框架调用关键字库，驱动各种工具完成被测系统的测试。



## 关键字库

关键字库的内部结构如下图所示：

其中：

* EPC Test Keywords为关键字库对RF的接口，所有关键字都从这里提供给RF使用；
* Test Manager用于加载拓扑等全局操作，也用于管理关键字库中的关键对象，目前包括分发器和消息序列，后续将加入测试报告检索模块的管理；
* Composer是各种业务过程的实际生成模块，从这个模块派生各种不同的业务过程，如emm、媒体面等；
* Message Sequence是测试脚本的消息序列，由Composer将Message的对象添加到消息序列中，供分发器读取；
* Message是抽象消息，从抽象消息继承派生出各种具体的消息，按照协议组织在一起，如l3、gtpv2等，后续可以改为按照接口来组织；
* Dispatcher是消息序列分发模块，用于将消息序列分解给到工具实体，并分发给各个工具执行，后续增加测试结果检查模块，分发器也需要与测试结果检查模块接口。

## 测试工具

测试工具需要实现的功能包括：

* 解析收到的测试脚本并执行各种业务消息；
* 信令接口的消息编解码能力；
* 按照脚本的控制功能，执行等待与循环等控制功能；
* 按照测试脚本上报和等待工具事件；
* 记录测试数据，上报测试报告。

# 演示系统简介

## 概要

由于本方案涉及到的系统众多，难以使用真实的DUT和测试工具进行演示，因此在本方案中，对测试工具和DUT都使用模拟程序来替代。Robot Framework和模拟测试工具、模拟DUT对接，模拟真实的测试系统的行为，来演示测试的执行过程。

本演示中，所有工具与DUT都为独立程序，通过UDP Socket相互通信。

## 模拟测试工具

### 特性

模拟测试工具主要包括工具的如下特征：

* 按照测试脚本执行测试过程；
* 执行各种过程控制，包括等待和循环；
* 事件上报与检测；
* 未实现接口消息的编解码。

### 框架



## 模拟被测网元

### 特性

模拟被测网元主要包括如下特征：

* 完成各种业务逻辑（由控制台指定业务）；
* 与模拟工具进行消息序列互动；
* 未实现接口消息的编解码。

### 框架



# 几个实例

## Attach、CreateSession——顺序过程、分布式部署

这两个例子中，将演示顺序过程的执行，以及分布式部署。

## 媒体面触发的配额下发——循环过程

这个例子演示媒体面触发配额下发这个循环过程如何执行。

## 多次Attach——使用事件通讯、高级关键字

这个例子演示多个工具间的协作以及高级关键字的使用。

## OMM过程——另一种测试工具

这个例子说明了OMM只不过是另一种测试工具，与其他测试工具没有明显的区别，类似的，后续扩展性能统计、计费等，也是一样的原理。

## 5个用户的Attach——数据驱动测试

这个例子演示了数据驱动测试的编辑和使用。

# 待完善

1. 预定义过程展开为消息序列的过程移到工具中；
2. 测试报告的格式与内容；
3. 测试结果检查的实现框架；
4. 其他工具纳入到该测试框架下的实现细化；

# 后续扩展

## 自动回复的模拟网元

测试过程中，有些网元在测试中不关心，又没有真实网元来执行其角色，可以考虑使用一种自动回复的模拟网元。

在Robot Framework中，自动回复的模拟网元与真实网元等同，也就意味着在测试用例中它是不可见的，这样就只能使用自动回复的模拟网元进行辅助测试，但是难以对其详细过程进行检查。自动回复的模拟网元不能代替模拟工具来执行测试。

自动回复的模拟网元应具备如下功能：

* 自动实现各种收到消息的自动应答；
* 这个工具中，看到的是收到的一个个消息，并逐一响应，因此工具不区分一个完整的业务流程，因此也无法区分一次测试的开始与结束；
* 由于这个工具无法区分业务的开始与结束，因此也就不需要向Robot Framework上报测试报告；
* 工具与RF间仍需要接口，用于设置消息应答内容，消息应答内容可以通过Robot Framework设置；
* 工具应提供控制台，用于设置各种消息应答内容，以及启动、停止等操作；

## Monkey Test

在本方案的框架中，工具提供了各种预定义过程，那么一种可能的测试方式就是随机组合各种测试过程来执行，即Monkey Test。

要执行这样的测试，首先要设置测试的目标。基于本方案的框架，可能有如下几种目标，这些目标的测试方法是不一样的。

1. 测试系统处理异常的健壮性。这种情况下可以不受限制的随机组合各种过程进行测试，不考虑资源回收，也不做每个测试用例的结果检查，只做用例无关的检查，如系统是否重启，是否有资源吊死长时间不释放等；
2. 测试某一个特定方向的健壮性。为实现这一测试目标，可以设置一些约束条件，在不违背约束的情况下，随机组合各种过程进行测试。这些约束条件，需要体现测试目标的倾向。是否在测试用例中执行检查，依赖于约束条件的强度。
3. 业务正确性遍历。可以设置较强的约束规则，按照规则随机组合的测试用例，基本都是可回收、可检查的，这种情况下可以设置用例级的通用检查项目，如业务是否能够发起，用户是否在线等。

上述三个测试目标，其基础都是随机组合测试过程，但是其约束条件逐渐增强。如何设置约束条件，将是实现这种测试方式的关键。另外，这种测试的有效性有待考察。