1. 脚本和消息不区分大消息
2. 删除了脚本中的Group，通过增加User Group实现用例级的号码本指定
3. 删除了脚本中的Office，RF会将脚本按照Office分解好，在消息中体现
4. 消息中的From、To字段，合并改为Office字段

脚本格式：

[SendMessage]

Param: <msg>.<attr>…=<value>|<variable>; …

Message: name=<msg>|alias=<alias>

Save: <msg>.<attr> -> <variable>

Wait: <value>

执行方法：构造消息，编码，发送

[RecvMessage]

Param: <msg>.<attr>…=<value>|<variable>; …

Message: name=<msg>|alias=<alias>

Save: <msg>.<attr> -> <variable>

Index: <value> #消息的序号，0表示所有

执行方法：检查收到的消息是否与此匹配，能匹配就执行下面的脚本，直到下一个RecvMessage

修改：删除了Expect，工具中所有Recv是可选的，在RF测试用例中的必选消息，会自动增加Check

[SendEvent]

Event: <office-name>: <string>

执行方法：向RF发送事件

修改：合并了局向与事件的字段

[RecvEvent]

Event: <office-name>: <string>

Time: <value> #超时时间，0表示无限等待

执行方法：阻塞等待接收事件

修改：合并了局向与事件的字段；增加了等待超时

[Wait]

Time: <value>

执行方法：阻塞等待指定的时间

[Stop]

执行方法：向RF发送Stop消息，向RF发送测试报告。

[Force Stop]

Time: <value>

执行方法：设置定时器，在定时器超时时，测试用例还没有结束，则通过Exception结束测试执行

[SetUserGroup]

Group-Name: <user group name>

执行方法：选用指定的号码本，RF保证这一操作在所有消息前执行

[SetStratergy]

Config: <attr>=<value>;…

执行方法：设置测试策略，RF保证这一操作在所有消息前执行

[RetrieveData]

Operation: <operation name>

Param: <para>|<para>=<value>;…

Return: <variable>

StartMessage: name=<msg>|alias=<alias>

StartParameter: <msg>.<attr>…=<value>|<variable>; …

StartIndex: <index>

EndMessage: name=<msg>|alias=<alias>

EndParameter: <msg>.<attr>…=<value>|<variable>; …

EndIndex: <index>

执行方法：执行Operation指定的计算，将Return variable与计算结果记入测试报告

RF—Tools消息接口：（是否可以换一种协议？xmlrpc？）

SendScript <tool type>/<tool version>

Office: <office-name>

Length: <value>

Content: <script>

方向：RF --> Tools

内容格式：参见“脚本格式”一节

使用时机：RF在开始测试前，通过这一消息将脚本分发给各个模拟网元

ScriptRecieved <tool type>/<tool version>

Office: <office-name>

方向：RF <-- Tools

使用时机：工具收到脚本后向RF发送该消息

修改：该消息不需要参数，因此省略了Length和Content

Start <tool type>/<tool version>

Office: <office-name>

方向：RF --> Tools

使用时机：RF分发完所有脚本后，向各个工具逐一发送该消息，顺序由工具决定

修改：该消息不需要参数，因此省略了Length和Content

Started <tool type>/<tool version>

Office: <office-name>

方向：RF <-- Tools

使用时机：工具收到Start，并已经开始运行后，向RF发送

修改：该消息不需要参数，因此省略了Length和Content

Event <tool type>/<tool version>

Office: <office-name>

Length: <value>

Content: <event>

方向：RF <--> Tools

内容格式：<office-name>: <event>

使用时机：工具执行到脚本中的SendEvent后，向RF发送；RF收到工具该消息后，通过组播向所有工具发送

Variable <tool type>/<tool version>

Office: <office-name>

Length: <value>

Content: <variable>

方向：RF <--> Tools

内容格式：<imsi> <name> <value> <type> #imsi似乎不应该属于这里，关于变量还要考虑一下。

使用时机：工具执行到脚本中的变量赋值后，向RF发送；RF收到工具该消息后，通过组播向所有工具发送

Report <tool type>/<tool version>

Office: <office-name>

Length: <value>

Content: <reprot>

方向：RF <-- Tools

内容格式：<token>: <value>

使用时机：工具发送End后发送，包含记录到Report中的所有内容

End <tool type>/<tool version>

Office: <office-name>

方向：RF <-- Tools

使用时机：工具执行到Stop脚本后，向RF发送

修改：该消息不需要参数，因此省略了Length和Content

Stop <tool type>/<tool version>

Office: <office-name>

方向：RF --> Tools

使用时机：RF收到工具的stop或Exception后，向所有其他工具组播

修改：该消息不需要参数，因此省略了Length和Content

Stopped <tool type>/<tool version>

Office: <office-name>

方向：RF <-- Tools

使用时机：工具收到Stop后，向RF发送

修改：该消息不需要参数，因此省略了Length和Content

Exception <tool type>/<tool version>

Office: <office-name>

Length: <value>

Content: <exception>

方向：RF <-- Tools

内容格式：<exception description string>

使用时机：工具中，脚本无法执行，向RF发送

变量

1. 变量系统指的是在测试用例执行过程中，将某些（收发消息中的）数据保存下来，供后续使用的机制；
2. 在测试用例中需要使用变量，在知识库中也需要使用变量；
3. 测试用例中的变量是用户测试时感兴趣的内容，而知识库中的变量存储的是维护各种接口会话的信息，二者目的不同；
4. 应该将测试用例中的变量与知识库中的变量区分开来，二者互相不可见；
5. 进一步的，知识库中的变量是维护各种接口的连接与会话用的，其本质就是接口上下文，因此将知识库中的变量称为上下文，更准确，也避免混淆；
6. 测试用例中的变量的生存期为测试用例的执行过程，提供（名称<-->值）的映射即可，与工具现在做的通用存储相当；
7. 知识库中的上下文系统需要重新整理。不同接口，不同协议，不同过程，上下文的生存期是不一样的，因此不能笼统的设置一个用户上下文，否则就会有现在工具“无法支持多承载”这样的问题。

测试用例中的变量系统

1. 在Send和Recv中，增加Save描述，用于将用户感兴趣的内容保存到变量中；
2. 在Send和Recv的Param中，参数的取值可以来自已经保存的变量；
3. 当收到消息时，可能只想保存相关信息，不做其他操作，因此应该允许Recv后没有动作
4. 保存的变量在整个测试用例内有效，测试用例执行结束后删除；
5. 一个测试用例，可能涉及到多个工具，所有这些工具应能看到同一份变量表，因此有变量的广播机制，保证各个工具看到的变量列表同步；

知识库中的协议上下文

1. 同一个协议可能应用在多个接口，每个接口又涉及到两个网元，有些接口又区分控制面/媒体面，因此，上下文的组织需要细化，大致可以按照“接口/网元/面”来区分。
2. 每一个上下文，应包含这个上下文的创建/删除时机，接口消息中哪些信息需要保存，接口消息中哪些信息需要引用，根据接口消息索引到上下文的方法；
3. 在收发消息时，首先根据收发的消息找到对应的上下文模板，再根据索引方法找到上下文实例，最后按照上下文模板中记录的方法，保存或读取参数信息；
4. 不管是测试用例中的消息，还是知识库中的消息，都需要经过这个上下文；



|  |  |  |  |
| --- | --- | --- | --- |
| Message Index | | | 备注 |
| Messsage |  |  | 消息名称 |
| Context | Context Type |  | 对应的上下文模板类型 |
| Need Create? |  | 是否需要创建上下文 |
| Need Delete? |  | 是否需要删除上下文 |
| Criteria | Parameter | 要索引的参数 |
| Variable Name | 需要与参数值相等的变量 |

|  |  |  |  |
| --- | --- | --- | --- |
| Context Template | | | 备注 |
| Context Type |  |  | 上下文模板类型 |
| Message Info Process | Message |  | 消息名称 |
| Parameter To Save | Parameter | 要保存的参数名 |
| Variable Name | 保存到的变量名 |
| Parameter To Retrieve | Parameter | 要读取的参数名 |
| Variable Name | 从哪个变量读取 |

|  |  |  |
| --- | --- | --- |
| Context Instance | | 备注 |
| Context Type |  | 上下文模板类型 |
| Saved Variable | Name | 变量名 |
| Value | 变量值 |