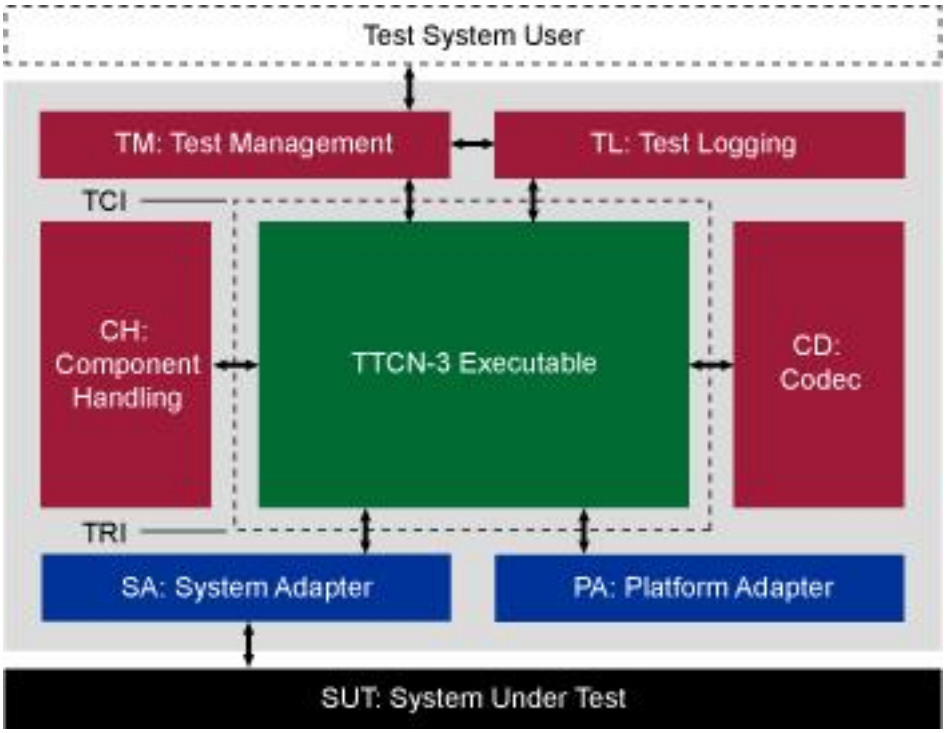
TTCN-3 测试系统外部编解码器

1. 编解码器在 TTCN-3 测试系统中实现的任务

许多协议和软件系统采用自己专有的方式编码信息，如果 TTCN-3 工具提供商没有为这些协议和系统提供编解实现，用户需要自己根据协议或者软件系统的编码规则来开发编解码实现。例如，DNS 和 SIP 协议都用专有的编码规则，因此需要用户定义编解码。只有当待测系统通信完全采用标准的编码规则如 ASN.1 的 BERs 或者 PERs,TTCN-3 工具提供商有可能提供现成可用的编解码器。一般来讲，是外部的编解码（External Codecs）负责执行 value representation used in the TE 和待测系统（SUT）可识别值间的编码，以及反过来的解码执行。



图一、TTCN-3 测试系统架构图

CD 编解码器实际执行两个操作，一个是编码操作 tciEncode，一个是解码操作 tciDecode。所以实际上编解码器的开发就是完成这两个函数的编写。

2. 访问 TTCN-3 值

值的编码和解码，外部编解码器都必须处理 TTCN-3 值。在编码过程中,一个值需要被检查和

仔细解析，依据编码规则生成二进制表示（binary representation）。在解码过程中，依据待测发过来的编码值解码生成一个新的 TTCN-3 值。这就要求外部编解码器可以处理 TE 内部的值。显然，不同 TTCN-3 工具 TE 内部值的表达会有变化。TCI 标准定义了抽象的类型和值的界面，弱化了不同 TTCN-3 供应商对编解码器值的影响。有了这些标准界面，编解码器实现（甚至是编解码生成工具）可以独立于 TTCN-3 工具实现。

3. 编码器实现

一个编码操作的实现实际是非常简单的。检查过要编码的 TTCN-3 值的类型后，一个合适的编码机制将被唤起，这个编码机制根据选定的编码规则构建和返回编码的二进制字符串。当然，取决于编码机制，需要或多或少的努力来实际生成这个编码形式。编码一个值通常需要系统的遍历值的结构（这个结构可以被认为是一个标记的树），以及恰当的装配值的子树的编码。

遍历和检查 TTCN-3 值树是通过 TCI Value 界面完成, TCI Value 界面提供了若干访问 TTCN-3 值的操作，包括访问所有基础类型和用户定义 TTCN-3 类型的值。例如，GetField 操作检索 record 或者 set 类型特定域的值,getBoolean 返回实际分配给 TTCN-3 布尔变量的值，依此类推。对于编码数据的构建，也就是 encode 操作返回值，TCI 仅指定具体值的结构，这和 TriMessageType type 用的是同样的。TRI 和 TCI 都没有为构造这样的值定义操作，因此外部编解码器必须自己实现这样的操作。

在我们的 DNS 消息的例子情况下，我们正在处理下边表中展示的类型和值。在编码时，TE 调用外部编解码器中的 encode 操作，传递模版 a_NokiaQuestion 作为要编码的 DnsMessage 值，编码这个值，encolder 首先应用 getField 操作检索 DnsMessage 值的 identifier 字段，然后用 getInt 提取这个字段的值，这个值就是 12345.DNS 标准强制编码 identifier 字段为 16 bits 以网络字节顺序。因此，encoder 将以字节 0X3039 开始消息编码。接着开始访问这个 record 类型的 messagekind 字段，提取 e_question 枚举类型，新增一个 zero bit 到编码数据,然后继续处理消息的其余部分。

```
Type integer            Identification (0..65535 );
Type enumerated         MessageKind { e_question, e_answer} ;
Type charstring         Question ;
Type charstring         Answer;

Type record DnsMessage {
   Identification       identification,
   MessageKind          messageKind,
   Question             question,
   Answer               answer    optional
}

Template DNSMessage a_NokiaQuestion := {
   Identification       := 12345
   messageKinda         := e_question,
   question             := www.nokia.com” ,
   answer               := omit
```

4. 解码器实现

依据预期的消息类型及编码的数据，Decode 操作实现试图构建一个 TTCN-3 的值。第一项任务因此通常是检测预期的消息类型，选择解码器负责这个特定类型解码的解码器实现。在详细的检查这个编码的数据时，解码器试图构建值的结构。构造值再次使用抽象的 TCI Value 界面。

在我们的 DNS 例子里，当调用解码器时假设消息是 DnsMessage，解码器在 Type 界面调用 newInstance 操作开始建造一个空的 DnsMessage 值。解码器然后开始检测接收的数据，试图以”network byte order”整数解码开始的 16 bit。解码的结果应用 setInt 操作存储在一个新建实例（instance）Identifier 里边。最终，采用 setField 操作，这个值会被设定为 DnsMessage 的 identifier 域。应用同样的方式，编码消息的其余部分将被检测，填进 DnsMessage 的其它域。一旦编码的消息完全被检测，DnsMessage 的所有值域都被设定，解码完成。

解码器在构造值时会碰到一些不合法的编码或者编码数据的信息丢失情况，这是一个解码错误，解码操作返回给 TE 一个空的解码值来报告错误。否则，成功解码的结果传递给 TE.

5. 编解码器实现中的一些高级问题

Advanced Aspects of Codec Implementation

The handling of decoding of subtyped values requires special attention because the set of allowed values is currently not discernible through the Type or Value interface .Still,The CD will need to check if the value restriction specified by a TTCN-3 type is met before the decoded value is set in TE.An example of such a problematic case would be when an interger value is decoded that lies outside the admissible valules for a field, for example decoding a negative number to be put into the identifier field of a DnsMessage value.These case need to be treated as a decoding error because any attempt to set an incorrect value via the value interface will cause a test case error and hence the termination of the test execution.

This mean that there will have to be a very close resemblance between the modelling of message types in the TTCN-3 code and actual codec implementation .In particular ,it must be ensured that a decoder will not successfully decode a message that cannot be represented inside the expected subtype constraints within the TE.In the case of the identifier field of the DnsMessage,this close resemblance is automatically present because any unsigned 16-bit interger value lies within the admissible range of the Identifier type.

Unlike the implementation of SA and PA, it is not necessary for encoders and decoders to execute in parallel to the TE.That is because when the TE invokes the encode or decode operation ,it is prepared to wait for the completion of the encoding or decoding process.Implementation will have to be re-entrant ,though ,because concurrently executing test components may lead to simultaneous invocation of encode and decode.

Similar to our previous discussion of SA and PA implementation .the TCI does not make assumptions about the concrete nature of a CD implementation .For example hard-wiring separate encoder or decoder for specific types is not required ,as we have suggested here when discussing codecs for our DNS example. It is also possible to implement more generic codecs,which perform edcoding and decoding based on a type’s structure(as it is possible for BER or PER encoders) or on user-specified meta-information like encoding attributes.We have already sketched this approach in Section 7.5.4