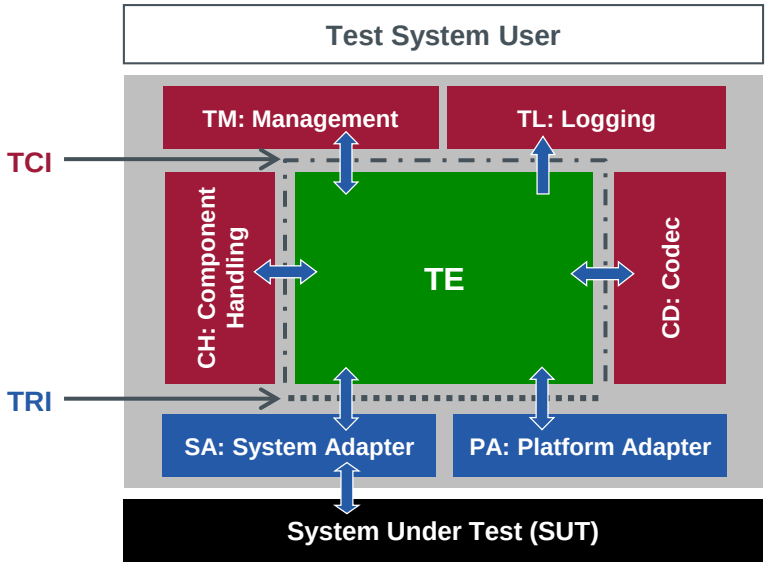
Executing TTCN-3 Tests

The Runtime/Control Interface (TRI/TCI)

Agenda

- Introduction
- Integrating TTCN-3 into test environment
  - TCI: Test Management (TM) and Component Handling (CH)
- Integrating TTCN-3 into test devices
  - TRI: System/Platform Adapter (SA/PA)
  - Coding Examples – TA
  - TCI: Codec Interface
  - Coding Examples – CD
- Summary

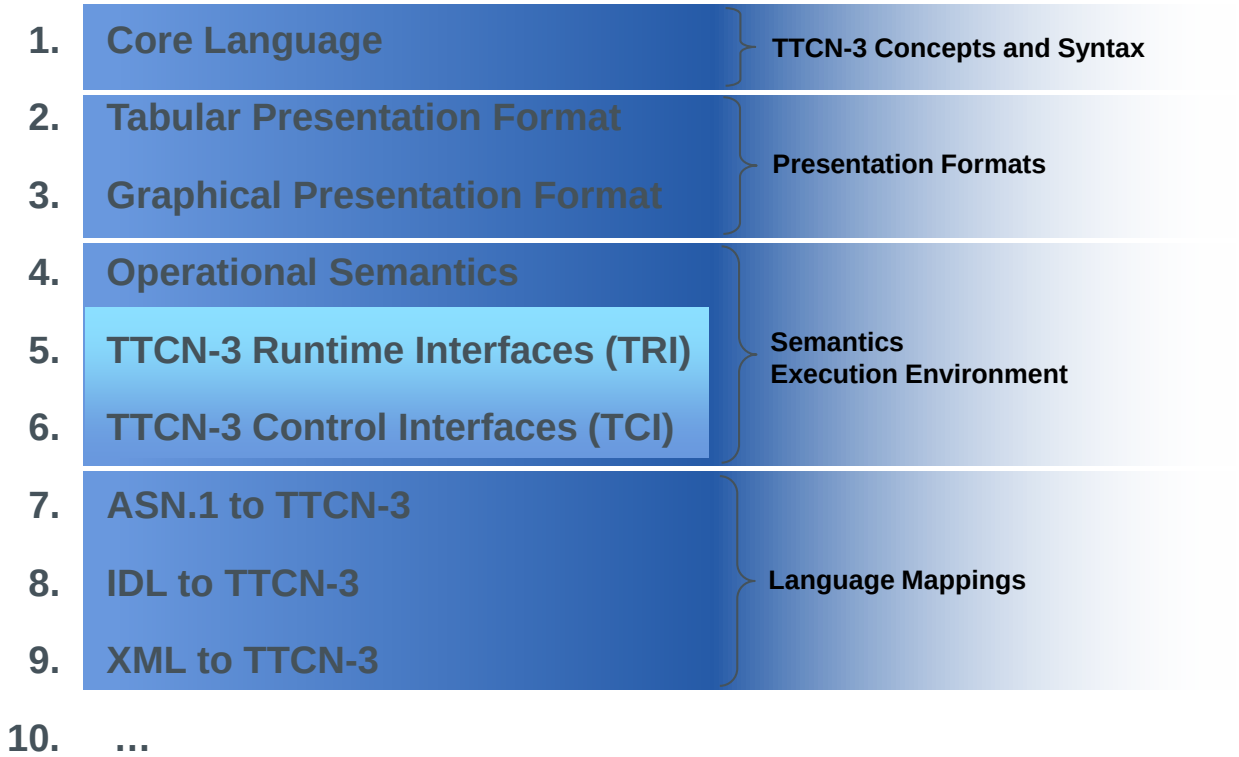
A TTCN-3 test system



SUT – System Under Test
TE – TTCN-3 Executable
TM – Test Management
TL – Test Logging
CD – Codec
CH – Component Handling
SA – System Adapter
PA – Platform Adapter

ETSI ES 201 873-1 TTCN-3 Core Language (CL)
ETSI ES 201 873-6 TTCN-3 Control Interfaces (TCI)
ETSI ES 201 873-5 TTCN-3 Runtime Interface (TRI)

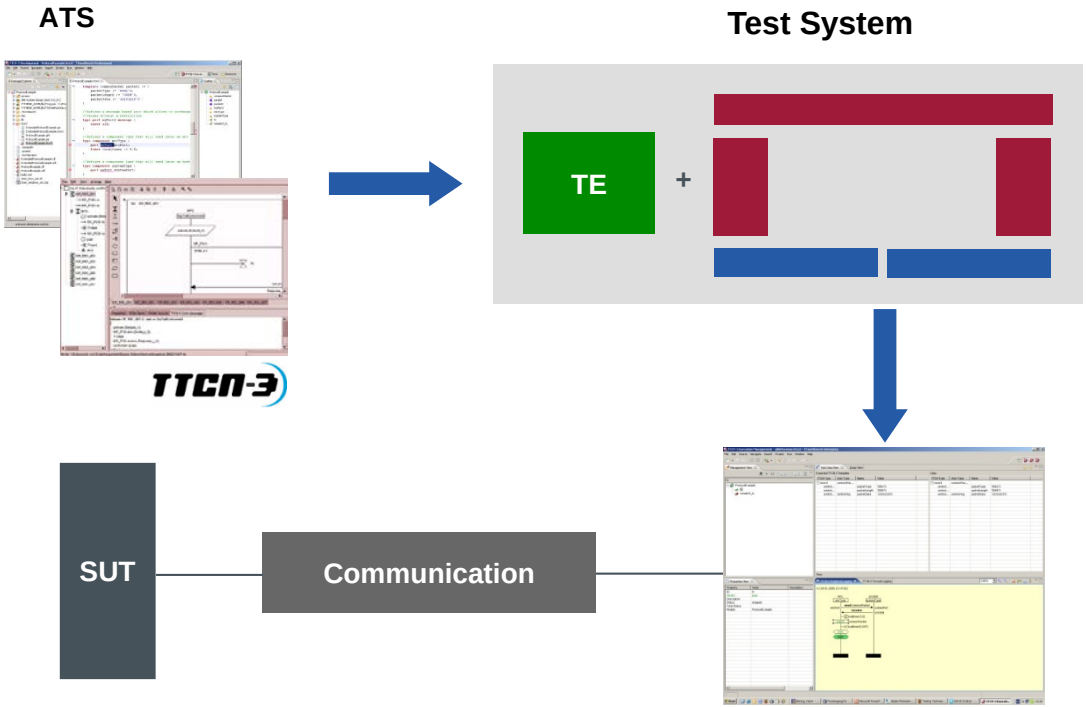
Standard overview



Steps to implement TTCN-3

- Translate TTCN-3 into executable code
- Adapt runtime environment to test management
- Implement communication and test platform aspects

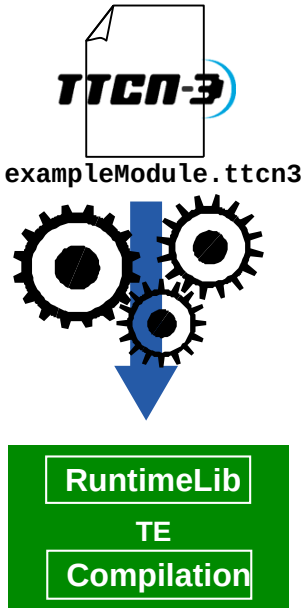
Implementation

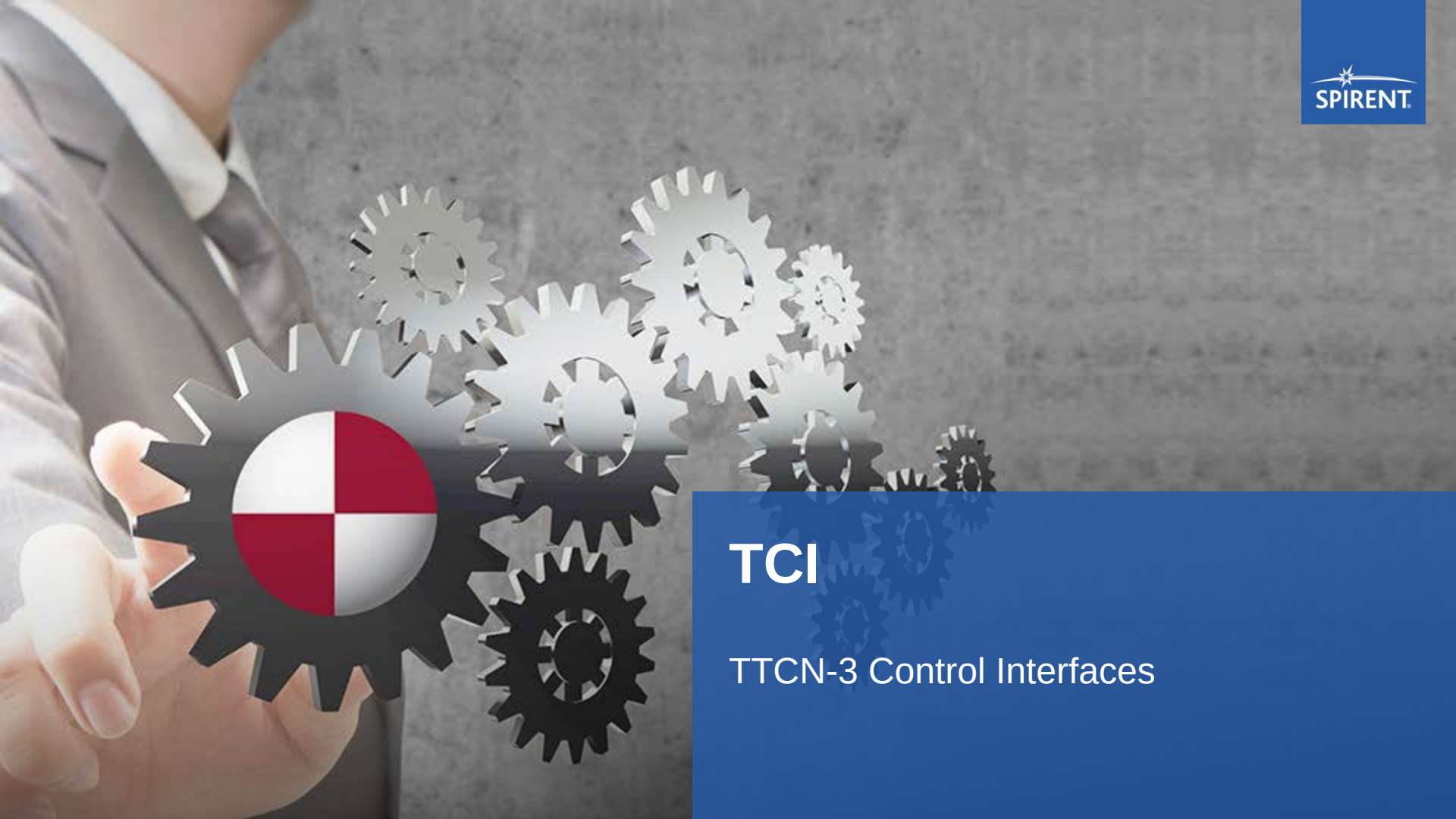


Translate TTCN-3 into executable code

```
F:\AB>TTthree DNSTest
```

- Reads module definitions written in the TTCN-3 core notation
- Generates code and compiles it into executable code
- Runtime support through runtime libraries

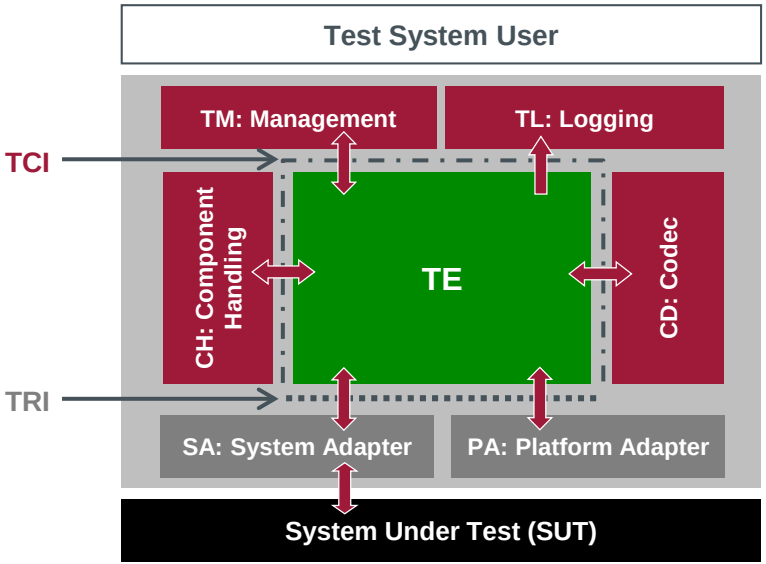




TCI

TTCN-3 Control Interfaces

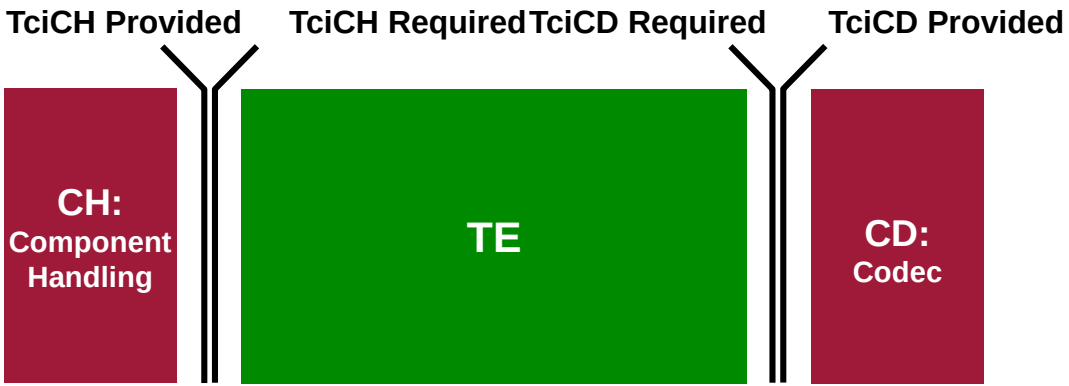
TCI – Distribution, management and codec adaptation



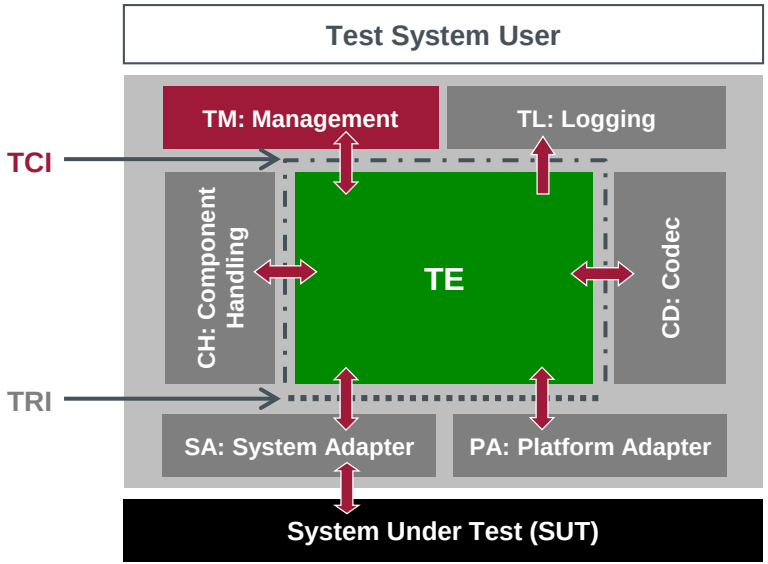
- Facts on the TTCN-3 Control Interfaces (TCI)
  - Standardized (part 6)
  - Language independent specification using IDL
  - Multi-vendor support

Common structure of sub-interfaces

- Have to be provided by the user (i.e. called by TE)
- Required functionality of the TE (i.e. called by the user)
- Applies to all TCI interfaces



A TTCN-3 test system



SUT – System Under Test
TE – TTCN-3 Executable
TM – Test Management
TL – Test Logging
CD – Codec
CH – Component Handling
SA – System Adapter
PA – Platform Adapter

ETSI ES 201 873-1 TTCN-3 Core Language (CL)
ETSI ES 201 873-6 TTCN-3 Control Interfaces (TCI)
ETSI ES 201 873-5 TTCN-3 Runtime Interface (TRI)

Why test management interface?

- Different applications need different test management functionality
  - Command line test management
  - Graphical test management
  - Web-based test management
  - Integration into existing platforms
- One TTCN-3 oriented interface needed!

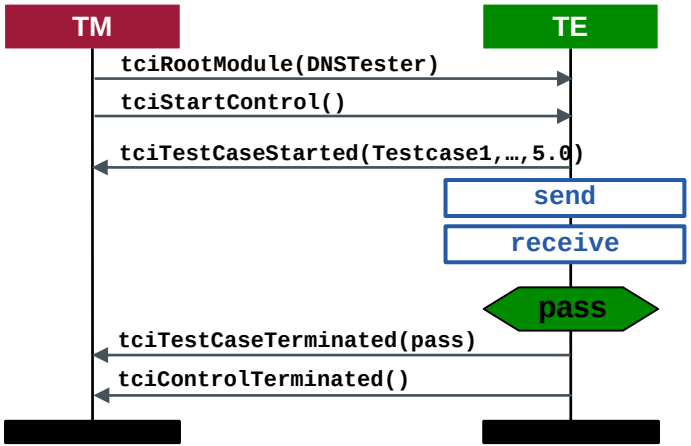
The test management interface

- TM provides
  - User interface
    - incl. error reporting
  - Keeps track of test case execution
  - Module parameter resolving
  - Logging

- TE provides
  - Entry points to the TE
  - Start/stop test case
  - Start/stop control part
- 6 operations provided
- 9 operations required

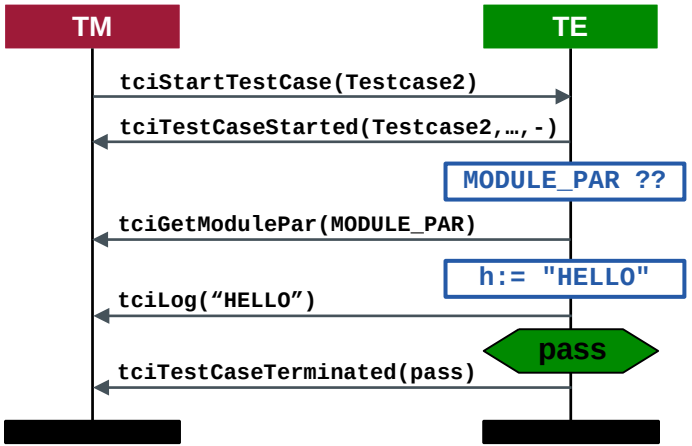
Dynamics of test management

```
testcase Testcase1() runs on DNSTester{
   P.send(query);
   P.receive(answer);
   setverdict(pass);
}
```

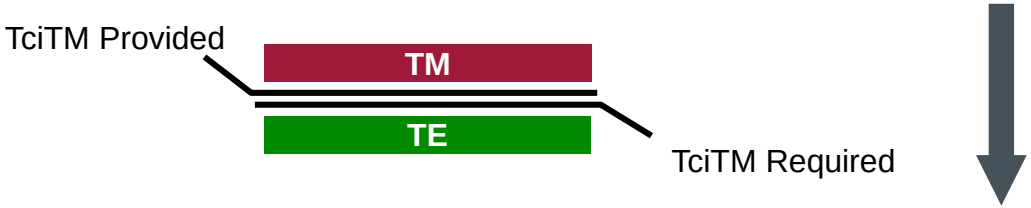


Dynamics of test management

```
testcase Testcase2() runs on DNSTester {
    var charstring h:= MODULE_PAR ;
    log(h);
    setverdict(pass);
}
```

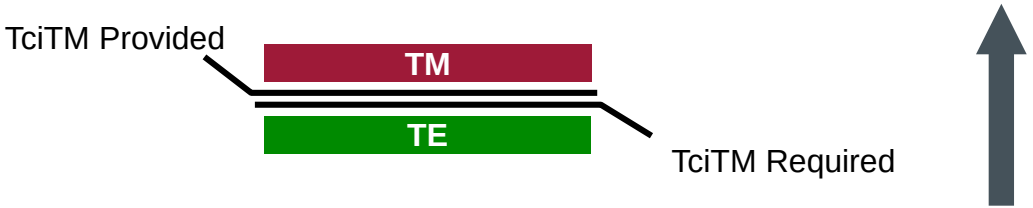


The TciTMRequired interface



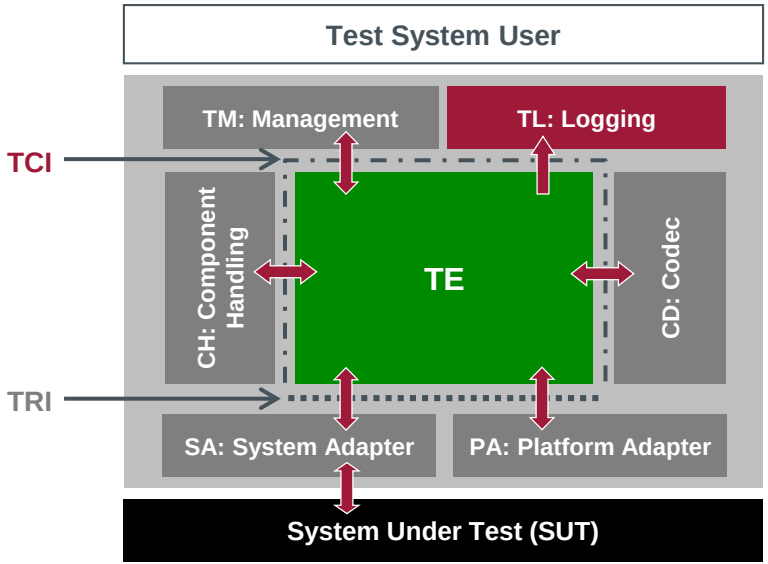
- TE offers the entry point for test case execution and some rudimentary database functionality
- Complete set of operations
  - `void tciRootModule(moduleId);`
  - `TciModuleParameterList tciGetModuleParameters(moduleId);`
  - `TciTestCaseIdList tciGetTestCases();`
  - `TciParameterTypeList tciGetTestCaseParameters(testCaseId);`
  - `TriPortIdList tciGetTestCaseTSI(testCaseId);`
  - `void tciStartTestCase(testCaseId, parameterList );`
  - `void tciStopTestCase();`
  - `TriComponentId tciStartControl();`
  - `void tciStopControl();`

The TciTMProvided interface



- Feedback of the TE to the status of execution or request for module parameters
- Complete set of operations
  - `void tciTestCaseStarted (testCaseId, parameterList, timerValue);`
  - `void tciTestCaseTerminated ( verdict, parameterList);`
  - `void tciControlTerminated ();`
  - `Value tciGetModulePar (parameterId);`
  - `void tciLog (testComponentId, message);`
  - `void tciError (String message);`

A TTCN-3 test system



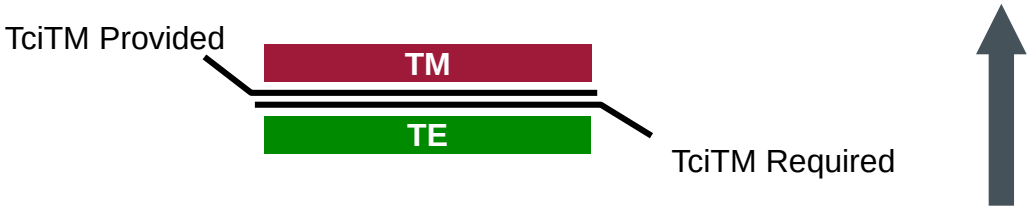
SUT – System Under Test
TE – TTCN-3 Executable
TM – Test Management
TL – Test Logging
CD – Codec
CH – Component Handling
SA – System Adapter
PA – Platform Adapter

ETSI ES 201 873-1 TTCN-3 Core Language (CL)
ETSI ES 201 873-6 TTCN-3 Control Interfaces (TCI)
ETSI ES 201 873-5 TTCN-3 Runtime Interface (TRI)

Why logging interface?

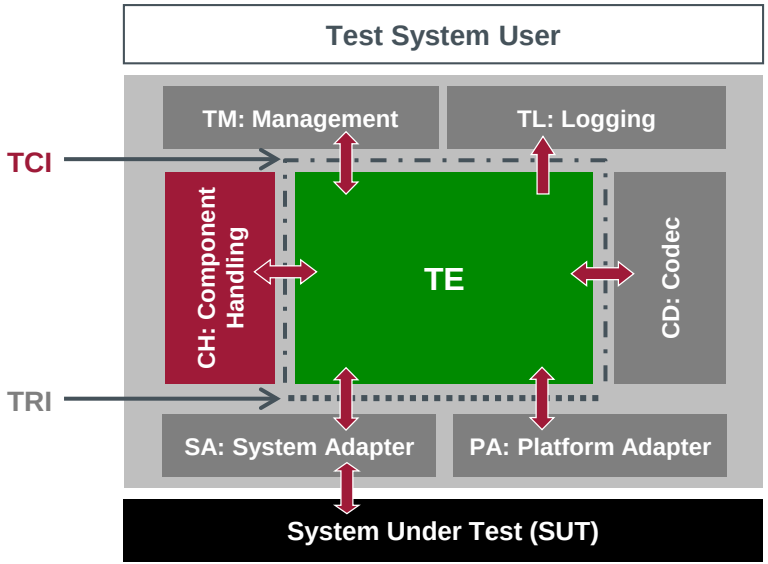
- Includes all operations needed to retrieve information about test execution
- Controls the detail level of log information performed by
  - TE
  - SA
  - PA
  - CH
  - CD

The TciTMProvided interface



- TciTL contains only **`Provided`** sub interface
- 105 operations
- Extract of operations
  - `void tliTcExecute(TString, TInteger, TString, TInteger, TriComponentIdType, TciTestCaseIdType, TriParameterListType, TriTimerDurationType)`
  - `void tliTcStart(TString, TInteger, TString, TInteger, TriComponentIdType, TciTestCaseIdType, TriParameterListType, TriTimerDurationType)`
  - `void tliTcStop(TString, TInteger, TString, TInteger,TriComponentIdType)`
  - `void tliTcStarted(TString, TInteger, TString, TInteger, TriComponentIdType, TciTestCaseIdType, TciParameterListType, TriTimerDurationType)`

A TTCN-3 test system



SUT	–	System Under Test
TE	–	TTCN-3 Executable
TM	–	Test Management
TL	–	Test Logging
CD	–	Codec
CH	–	Component Handling
SA	–	System Adapter
PA	–	Platform Adapter

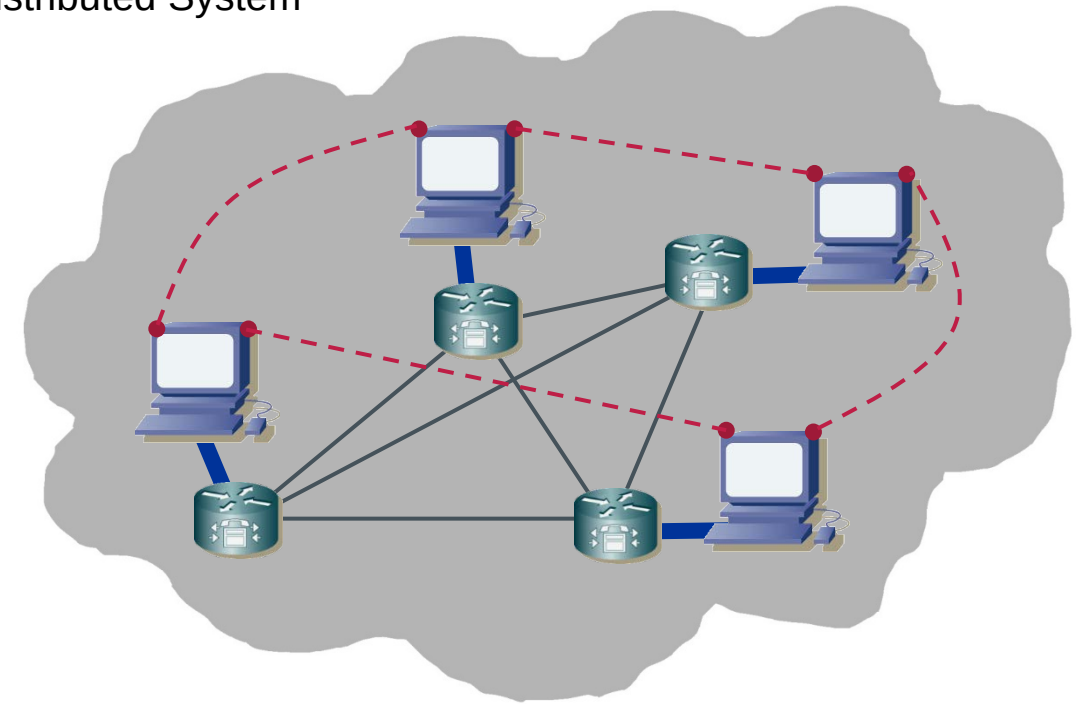
ETSI ES 201 873-1	TTCN-3 Core Language (CL)
ETSI ES 201 873-6	TTCN-3 Control Interfaces (TCI)
ETSI ES 201 873-5	TTCN-3 Runtime Interface (TRI)

Why Component Handling interface?

- TTCN-3 suitable for different test applications
  - Functional testing
  - Load testing
  - Interoperability testing
- Need to distribute test components if running short of resources!

Distributed testing

Network / Distributed System

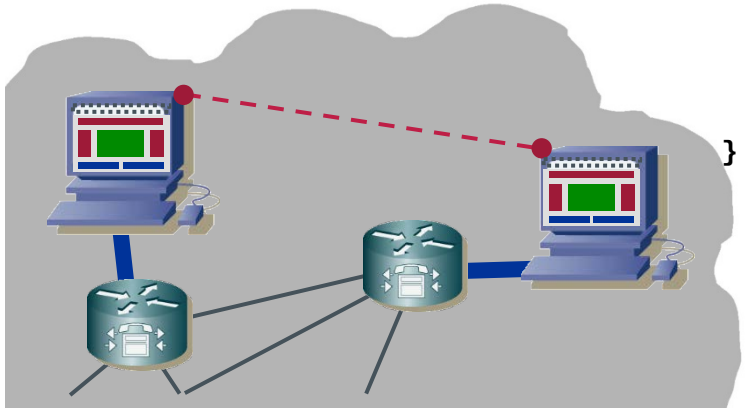


The Component Handling interface

- Management of TTCN-3 components
  - No implementation of TTCN-3 functionality
  - Distribution of TTCN-3 configuration operations
  - Distribution of TTCN-3 inter-component communication
- Concept of distributed TE, i.e. multiple TEs
  - A single component handling entity
  - Presence of a distinct TE*, i.e. the TE where a test case or the control part has been started
  - Distinct TE* responsible for final verdict calculation
- The most complex interface
  - 17 required operations
  - 17 provided operations

Example: DNS tester scalability testing

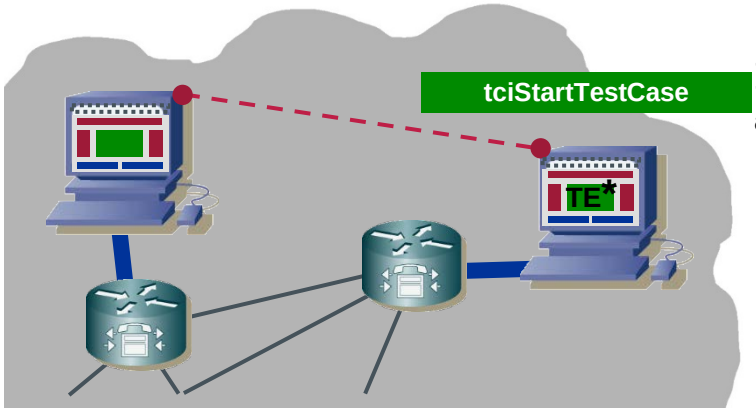
```
function clntBehavior(...) {
      runs on MyPtcType
  {
   // Do what you have to do !
    setverdict(pass) ;
    stop ;
  }
}
```



```
 testcase scalabilityTest()
     runs on MTC system TestSystemInt
{
  var integer i;

  for(i:=0;i<MAXNUMBER;i:=i+1) {
    abC[i] := MyPtcType.create;
    map(abC[i]:S, system:R);
    connect(mtc:C, abC[i]:C);
    abC[i].start(clntBehavior(USER[i]));
    C.send(....) to abC[i] ;
  }

  all component.done ;
}
```

Example: DNS tester scalability testing

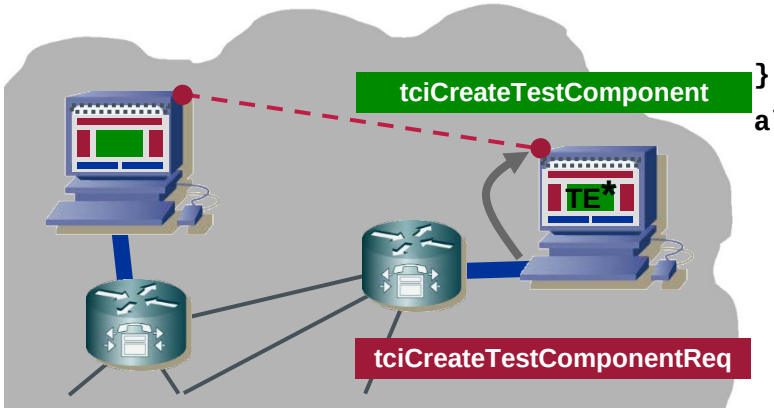
```
function clntBehavior(...) {
      runs on MyPtcType
  {
   // Do what you have to do !
    setverdict(pass) ;
    stop ;
  }
}
```




```
testcase scalabilityTest()
    runs on MTC system TestSystemInt
{
  var integer i;

  for(i:=0;i<MAXNUMBER;i:=i+1) {
    abC[i] := MyPtcType.create;
    map(abC[i]:S, system:R);
    connect(mtc:C, abC[i]:C);
    abC[i].start(clntBehavior(USER[i]));
    C.send(....) to abC[i] ;
  }
  all component.done ;
```

Example: DNS tester scalability testing

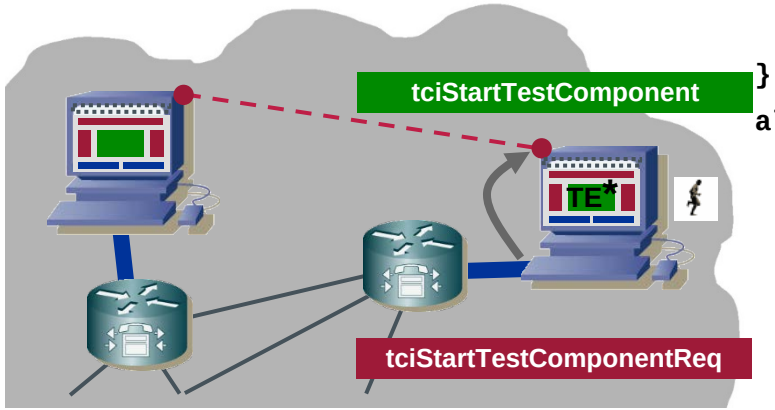
```
function clntBehavior(...) {
      runs on MyPtcType
  {
   // Do what you have to do !
    setverdict(pass) ;
    stop ;
  }
}
```




```
 testcase scalabilityTest()
     runs on MTC system TestSystemInt
{
  var integer i;

  for(i:=0;i<MAXNUMBER;i:=i+1) {
    abC[i] := MyPtcType.create;
    map(abC[i]:S, system:R);
    connect(mtc:C, abC[i]:C);
    abC[i].start(clntBehavior(USER[i]));
    C.send(....) to abC[i] ;
  }
  all component.done ;
```

Example: DNS tester scalability testing

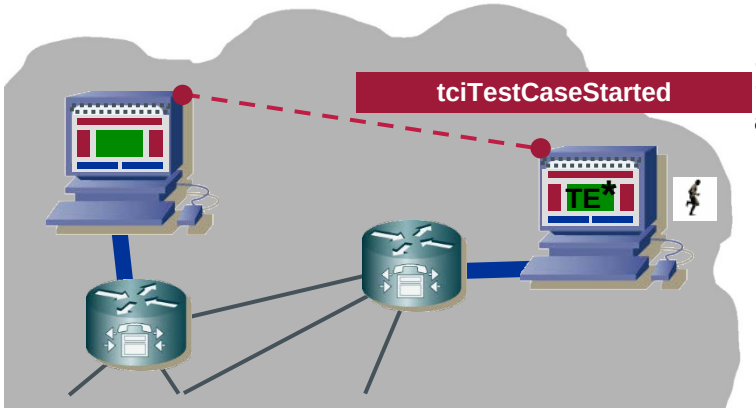
```
function clntBehavior(...) {
      runs on MyPtcType
  {
   // Do what you have to do !
    setverdict(pass) ;
    stop ;
  }
}
```




```
testcase scalabilityTest()
    runs on MTC system TestSystemInt
{
  var integer i;

  for(i:=0;i<MAXNUMBER;i:=i+1) {
    abC[i] := MyPtcType.create;
    map(abC[i]:S, system:R);
    connect(mtc:C, abC[i]:C);
    abC[i].start(clntBehavior(USER[i]));
    C.send(....) to abC[i] ;
  }
  all component.done ;
```

Example: DNS tester scalability testing

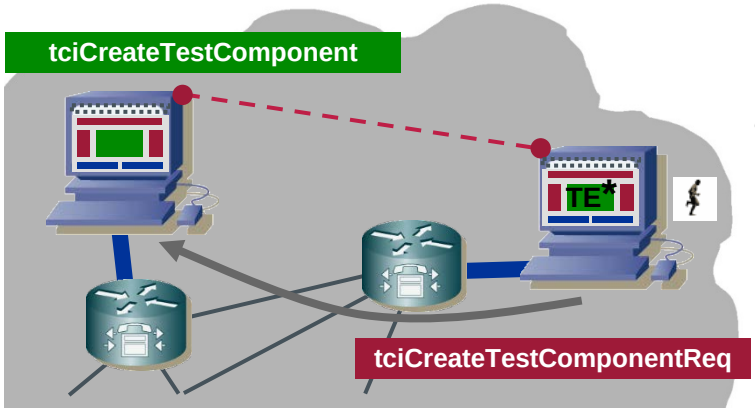
```
function clntBehavior(...) {
      runs on MyPtcType
  {
   // Do what you have to do !
    setverdict(pass) ;
    stop ;
  }
}
```




```
testcase scalabilityTest()
    runs on MTC system TestSystemInt
{
  var integer i;

  for(i:=0;i<MAXNUMBER;i:=i+1) {
    abC[i] := MyPtcType.create;
    map(abC[i]:S, system:R);
    connect(mtc:C, abC[i]:C);
    abC[i].start(clntBehavior(USER[i]));
    C.send(....) to abC[i] ;
  }
  all component.done ;
```

Example: DNS tester scalability testing

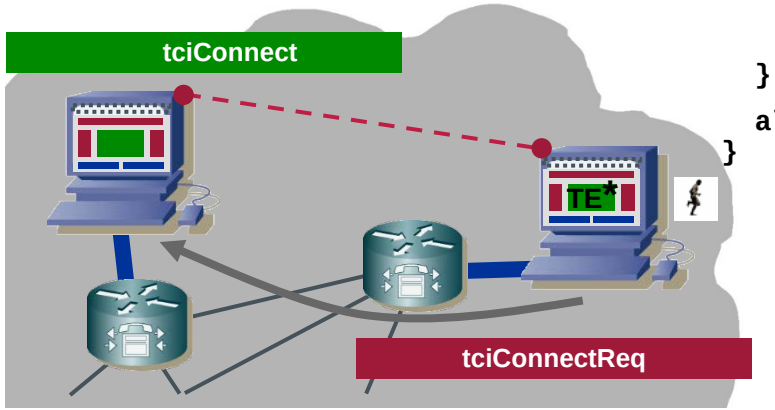
```
function clntBehavior(...) {
      runs on MyPtcType
  {
   // Do what you have to do !
    setverdict(pass) ;
    stop ;
  }
}
```




```
testcase scalabilityTest()
    runs on ABClient system ABTester
{
  var integer i;

  for(i:=0;i<MAXNUMBER;i:=i+1) {
    abC[i] := MyPtcType.create;
    map(abC[i]:S, system:R);
    connect(mtc:C, abC[i]:C);
    abC[i].start(clntBehavior(USER[i]));
    C.send(....) to abC[i] ;
  }

  all component.done ;
```

Example: DNS tester scalability testing

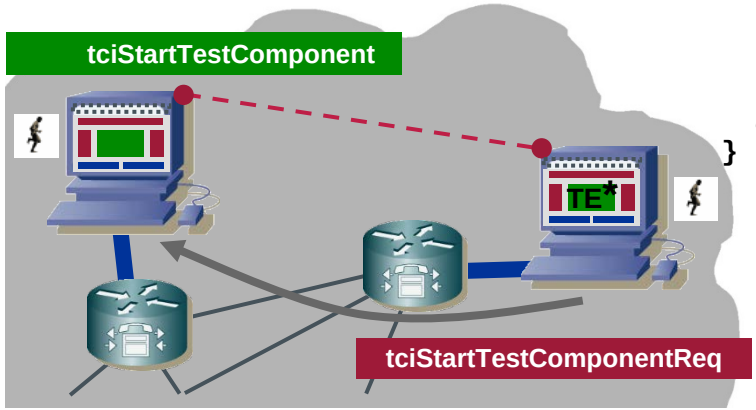
```
function clntBehavior(...) {
      runs on MyPtcType
  {
   // Do what you have to do !
    setverdict(pass) ;
    stop ;
  }
}
```




```
testcase scalabilityTest()
    runs on ABClient system ABTester
{
  var integer i;

  for(i:=0;i<MAXNUMBER;i:=i+1) {
    abC[i] := MyPtcType.create;
    map(abC[i]:S, system:R);
    connect(mtc:C, abC[i]:C);
    abC[i].start(clntBehavior(USER[i]));
    C.send(....) to abC[i] ;
  }

  all component.done ;
}
```

Example: DNS tester scalability testing

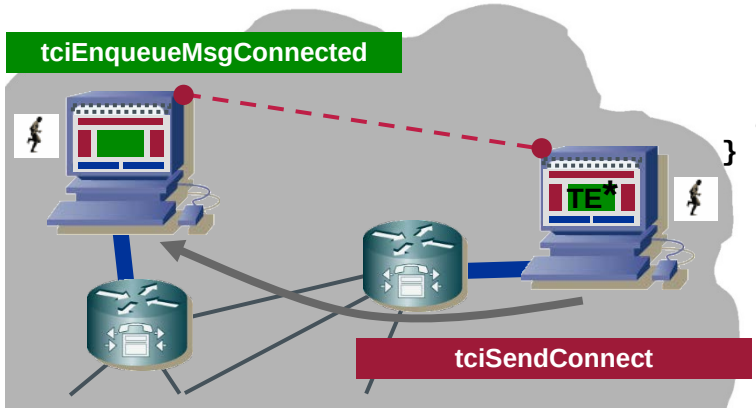
```
function clntBehavior(...) {
      runs on MyPtcType
  {
   // Do what you have to do !
    setverdict(pass) ;
    stop ;
  }
}
```




```
testcase scalabilityTest()
    runs on ABClient system ABTester
{
  var integer i;

  for(i:=0;i<MAXNUMBER;i:=i+1) {
    abC[i] := MyPtcType.create;
    map(abC[i]:S, system:R);
    connect(mtc:C, abC[i]:C);
    abC[i].start(clntBehavior(USER[i]));
    C.send(....) to abC[i] ;
  }

  all component.done ;
}
```

Example: DNS tester scalability testing

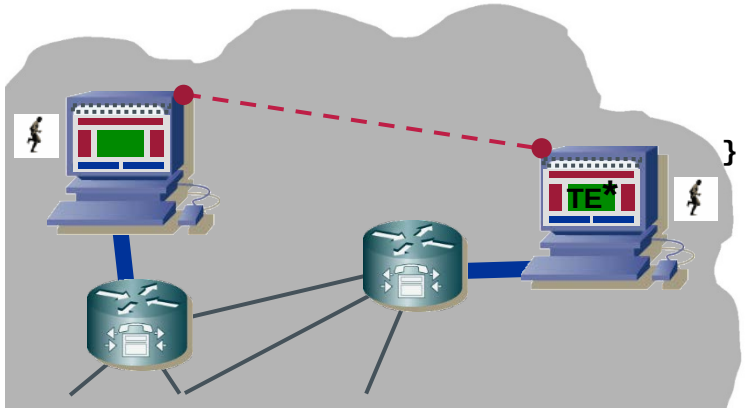
```
function clntBehavior(...) {
      runs on MyPtcType
  {
   // Do what you have to do !
    setverdict(pass) ;
    stop ;
  }
}
```




```
testcase scalabilityTest()
    runs on ABClient system ABTester
{
  var integer i;

  for(i:=0;i<MAXNUMBER;i:=i+1) {
    abC[i] := MyPtcType.create;
    map(abC[i]:S, system:R);
    connect(mtc:C, abC[i]:C);
    abC[i].start(clntBehavior(USER[i]));
    C.send(....) to abC[i] ;
  }

  all component.done ;
}
```

Example: DNS tester scalability testing

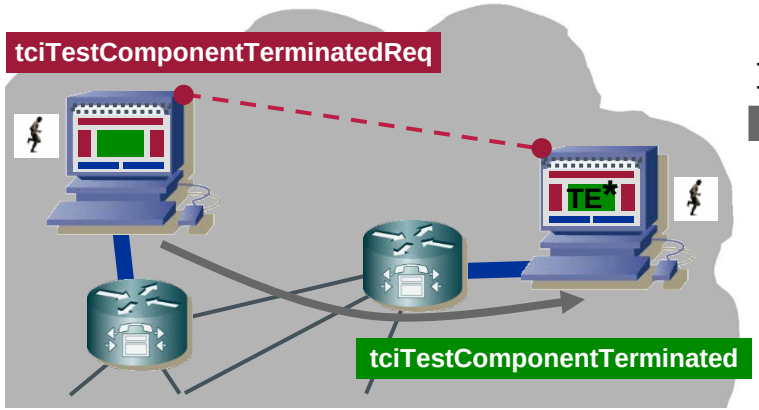
```
function clntBehavior(...) {
      runs on MyPtcType
  {
   // Do what you have to do !
    setverdict(pass) ;
    stop ;
  }
}
```




```
testcase scalabilityTest()
    runs on ABClient system ABTester
{
  var integer i;

  for(i:=0;i<MAXNUMBER;i:=i+1) {
    abC[i] := MyPtcType.create;
    map(abC[i]:S, system:R);
    connect(mtc:C, abC[i]:C);
    abC[i].start(clntBehavior(USER[i]));
    C.send(....) to abC[i] ;
  }

  all component.done ;
}
```

Example: DNS tester scalability testing

```
function clntBehavior(...) {
      runs on MyPtcType
  {
   // Do what you have to do !
    setverdict(pass) ;
    stop ;
  }
}
```

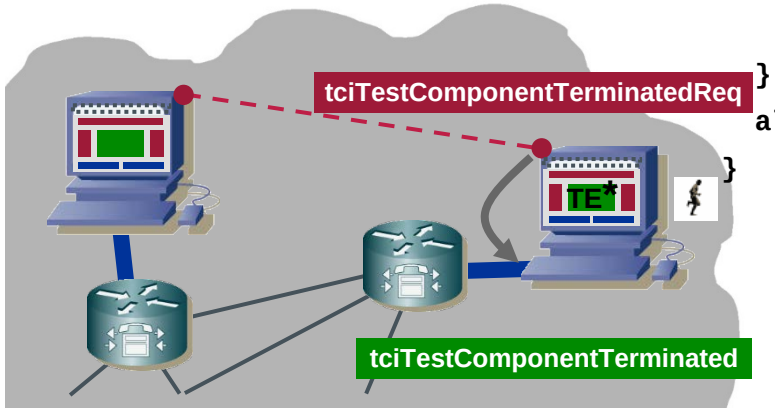



```
testcase scalabilityTest()
    runs on ABClient system ABTester
{
  var integer i;

  for(i:=0;i<MAXNUMBER;i:=i+1) {
    abC[i] := MyPtcType.create;
    map(abC[i]:S, system:R);
    connect(mtc:C, abC[i]:C);
    abC[i].start(clntBehavior(USER[i]));
    C.send(....) to abC[i] ;
  }
  all component.done ;
```

Example: DNS tester scalability testing

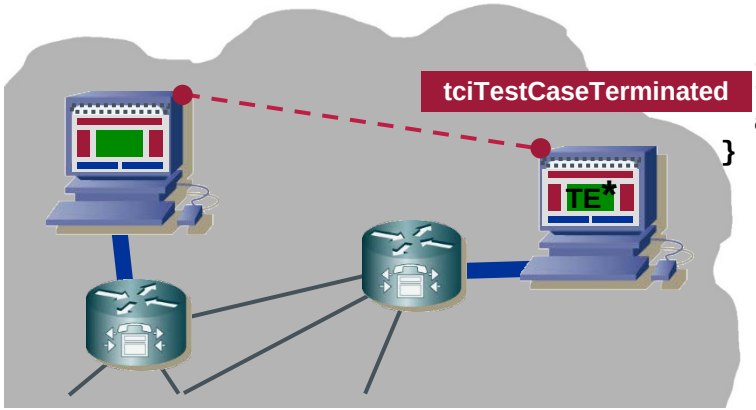
```
function clntBehavior(...) {
      runs on MyPtcType
  {
   // Do what you have to do !
    setverdict(pass) ;
    stop ;
  }
}
```

```
testcase scalabilityTest()
    runs on ABClient system ABTester
{
  var integer i;

  for(i:=0;i<MAXNUMBER;i:=i+1) {
    abC[i] := MyPtcType.create;
    map(abC[i]:S, system:R);
    connect(mtc:C, abC[i]:C);
    abC[i].start(clntBehavior(USER[i]));
    C.send(....) to abC[i] ;
  }
  all component.done ;
}
```

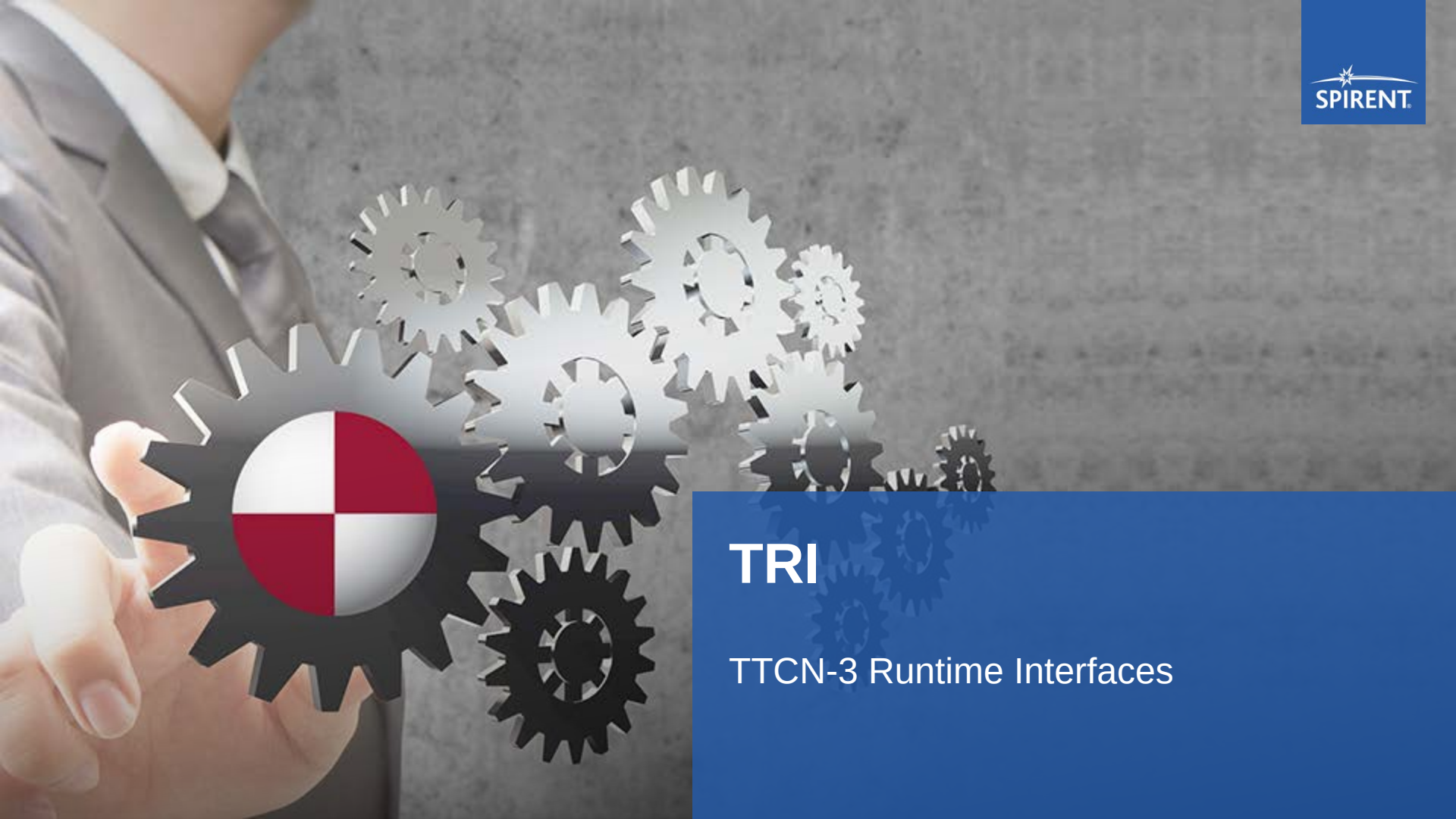


Example: DNS tester scalability testing

```
function clntBehavior(...) {
      runs on MyPtcType
  {
   // Do what you have to do !
    setverdict(pass) ;
    stop ;
  }
}
```



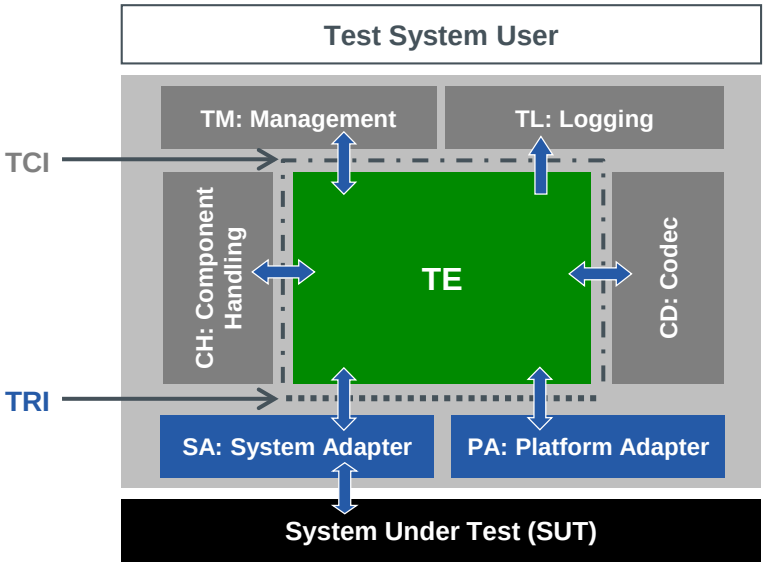

```
testcase scalabilityTest()
    runs on ABClient system ABTester
{
  var integer i;

  for(i:=0;i<MAXNUMBER;i:=i+1) {
    abC[i] := MyPtcType.create;
    map(abC[i]:S, system:R);
    connect(mtc:C, abC[i]:C);
    abC[i].start(clntBehavior(USER[i]));
    C.send(....) to abC[i] ;
  }
  all component.done ;
}
```



TRI

TTCN-3 Runtime Interfaces

TRI – Communication adaptation



- Facts on the TTCN-3 Runtime Interfaces (TRI)
  - Standardized (part 5)
  - Language independent specification
  - Multi-vendor support

Why TRI ?

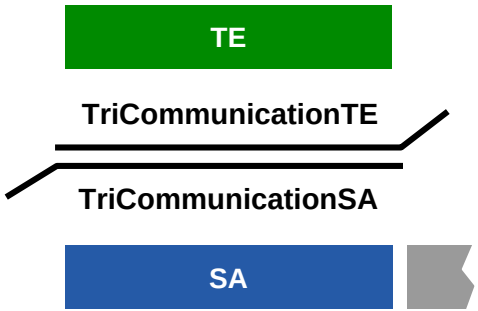
- Abstract Test Specifications (ATS) have to run on different test devices of different vendors
  - Different access to underlying protocol stacks
- ATS shall runs against systems in different development stages
  - Simulation
  - Software only
  - Embedded in hardware
- ATS can use different communications mechanisms and dynamic test configurations

Goals of TRI

- Specify a small and well-defined runtime interface for all future TTCN-3 test system implementations
- Free TRI definition from any unnecessary restrictions
  - Exclude test management and data access
  - Exclude communication between test components and their execution model
  - Avoid bias towards any particular programming language
- Historical older interface
  - Reference Implementation
  - Slightly different interface naming

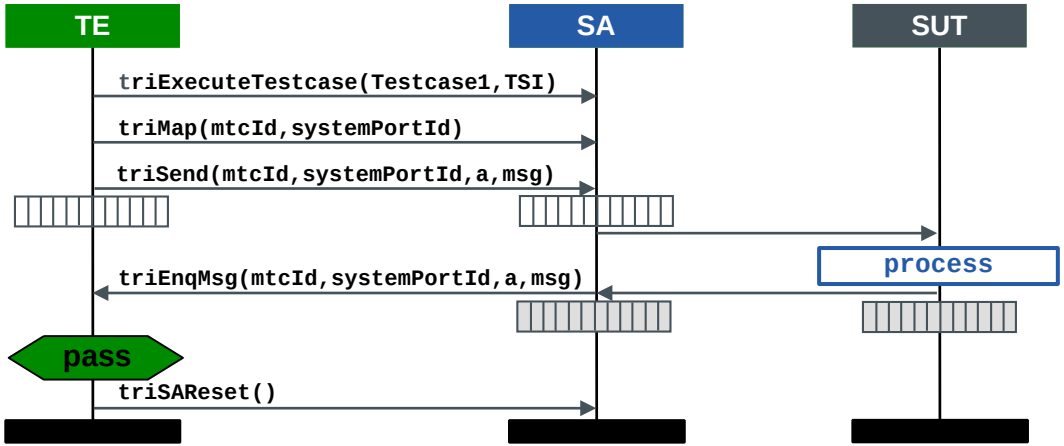
The TRI communication interface

- Interface structure
  - Due to historical reasons different naming
  - Applies to all TRI interfaces
  - SA reports status back
  - TE indicates error



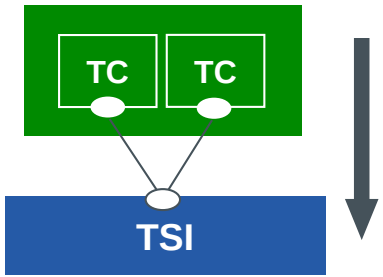
Dynamics of TRI SA

```
testcase Testcase1() runs on DNSTester system TSI {
        map(mtc:P, system:P);
        P.send(query);

        P.receive(answer);

        setverdict(pass);
}
```



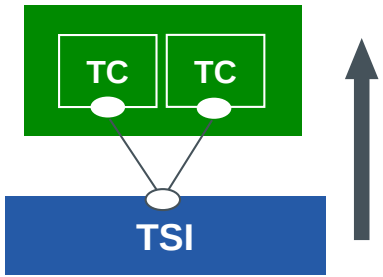
TriCommunicationSA interface

- Defines setting up configuration and sending of message to and/or calling of operations in the SUT
- Complete set of operations
  - `TriStatusType triSUTactionInformal (…);`
  - `TriStatusType triExecuteTestCase(…);`
  - `TriStatusType triMap(…);`
  - `TriStatusType triUnmap(…);`
  - `TriStatusType triSend(…);`
  - `TriStatusType triCall(…);`
  - `TriStatusType triReply(…);`
  - `TriStatusType triRaise(…);`
  - `TriStatusType triSAReset();`



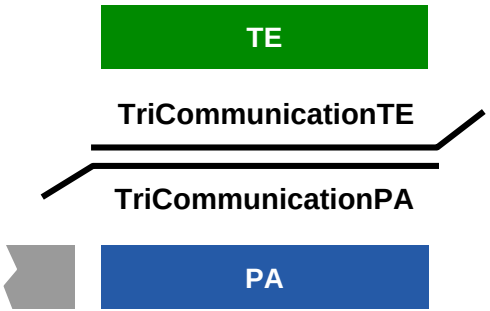
TriCommunicationTE interface

- Defines receiving of messages and/or calling of operations in the TE
- Complete set of operations
  - `void triEnqueueMsg(…);`
  - `void triEnqueueCall(…);`
  - `void triEnqueueReply(…);`
  - `void triEnqueueException(…);`



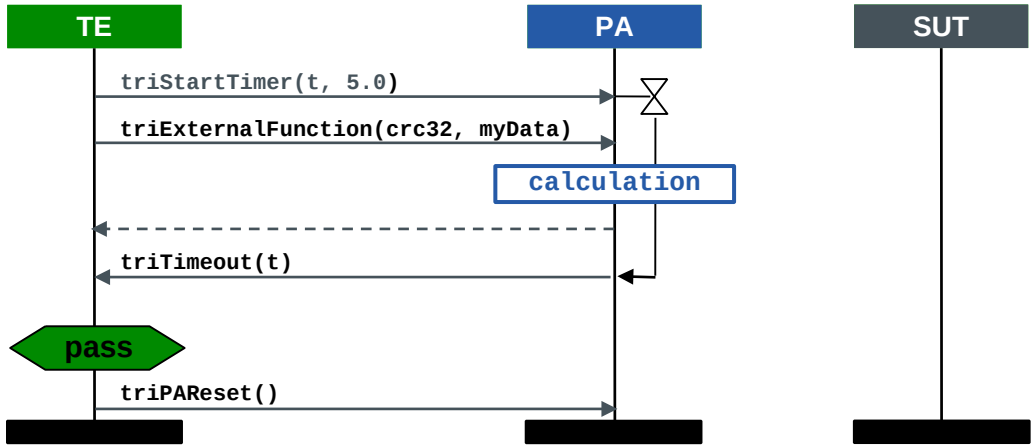
The TRI platform interface

- Interface structure
  - Implementation of time and external functions
  - PA reports status back
  - TE indicates error



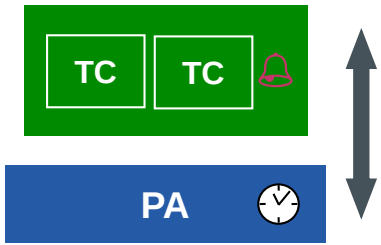
Dynamics of TRI PA

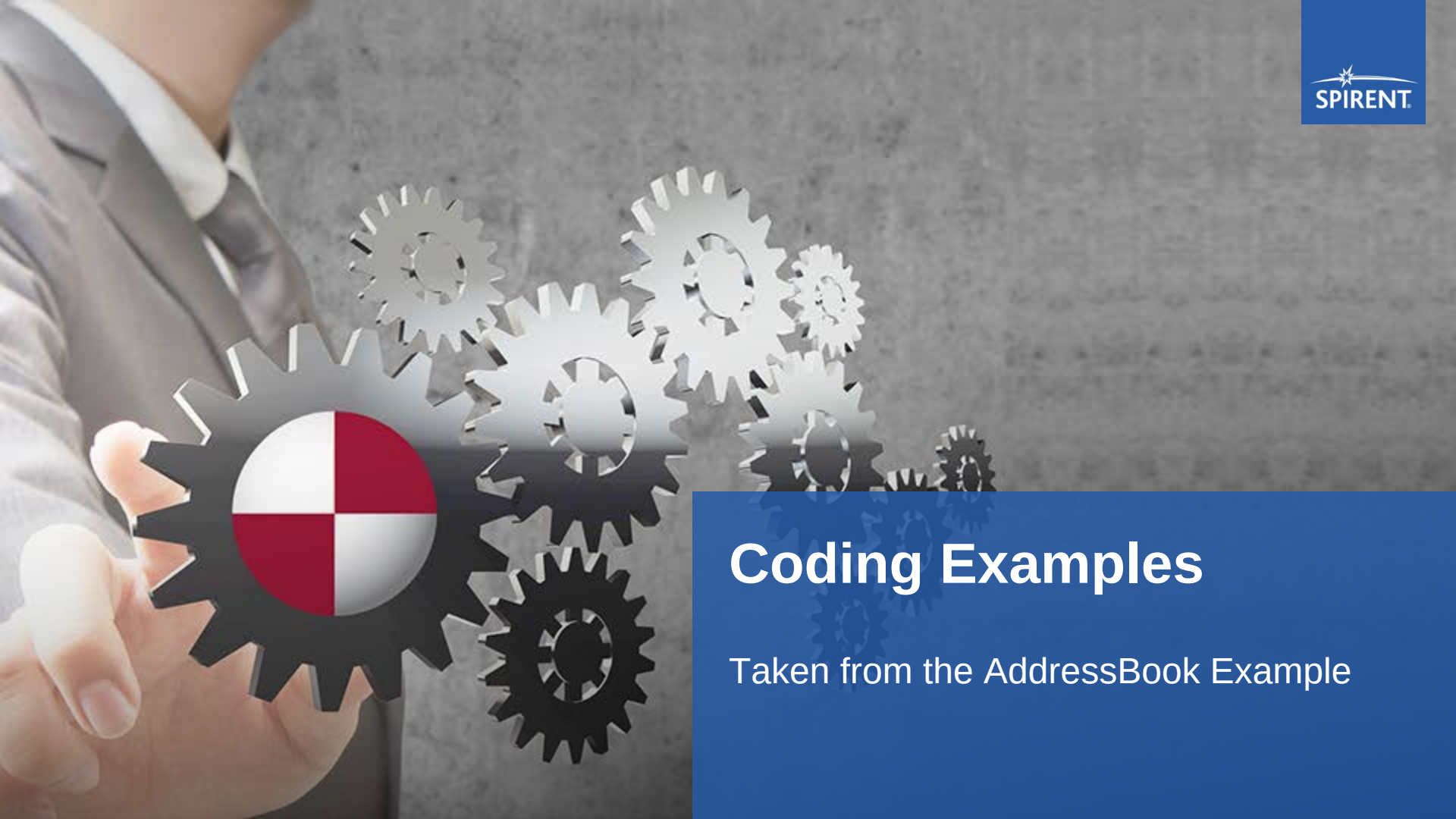
```
testcase Testcase2() runs on DNSTester system TSI{
        var timer t := 5.0 ;
        t.start ;
        var octetstring crc := crc32(myData);
        t.timeout;
        setverdict(pass);
}
```



TriPlatform interface

- Defines control of time and calling of external functions
- Complete set of operations (PA)
  - `TriStatusType triPAReset();`
  - `TriStatusType triStartTimer(…);`
  - `TriStatusType triStopTimer(…);`
  - `TriStatusType triReadTimer(…);`
  - `TriStatusType triTimerRunning(…);`
  - `TriStatusType triExternalFunction(…);`
- Complete set of operations (PA)
  - `void triTimeout(…);`





Coding Examples

Taken from the AddressBook Example

Import example projects

- TTworkbench contains several example projects
- Each TTplugin contains at least one example project too
- Import over
  File → Import → TTCN-3 → TTCN-3 Examples → Address Book Example - message based
- In the folder AddressBookMsgRuntime there are codec (…\codec) and port plugin(…\tri) implementation examples
- In port plugin there are several functions

Test Adapter practically

AddressBookTestAdapter

- Extends basic test adapter class **`com.testingtech.ttcn.tri.TestAdapter`**
- Overrides the following methods
  - **`public TriStatus triExecuteTestcase (TriTestCaseId tc, TriPortIdList tsiPortIdList)`**
  - **`public TriStatus triMap (TriPortId compPortId, TriPortId tsiPortId)`**
  - **`public TriStatus triUnmap (TriPortId compPortId, TriPortId tsiPortId)`**
  - **`public TriStatus triSend (TriComponentId componentId, TriPortId tsiPortId, TriAddress address, TriMessage sendMessage)`**
  - **`public TciCDProvided getCodec(String encodingName)`**
  - **`public void triSAReset()`**

public TriStatus triExecuteTestcase (TriTestCaseId tc, TriPortIdList tsiList)

```
public TriStatus triExecuteTestcase(final TriTestCaseId testcase, final TriPortIdList tsiList) {
    // get the parameter values from the management (TA)
    PluginIdentifier pluginIdentifier = new
    PluginIdentifier("com.testingtech.ttcn.example.AddressBookMsg Runtime");
    // read the remote IP address
    remoteIPAddress = getTAParameter(pluginIdentifier, "myPort","REMOTE_IP_ADDRESS","");
    if (remoteIPAddress.equals("")) {
         return new TriStatusImpl("could not resolve remote IP address"); }
    // read the remotePortNumber and localPortNumber in the same way
    …
    rxSocket = null;
    txSocket = null;
    return new TriStatusImpl();
}
```

- Called just before a test cases starts execution
- **`triExecuteTestcase`** gets TA parameter values
- If succeeds returns **`TRI_OK`**, usage of predefined class TriStatusImpl()

`public TriStatus triMap (TriPortId compPortId, TriPortId tsiPortId)`

- Maps a test component port to a test system interface port
- In dynamic configurations typically receiver loops are started
- Parameter **`compPortId`** is a port reference to the test component port
- Parameter **`tsiPortId`** is a port reference to the test system interface port

public TriStatus triMap (TriPortId compPortId, TriPortId tsiPortId)

```
public TriStatus triMap (final TriPortId compPortId, final TriPortId tsiPortId) {
    // prepare to be ready to communicate, define the sockets for sending and receiving
    rxSocket = new DatagramSocket(localPortNumber);
    txSocket = new DatagramSocket();
    …
    // Define and start a thread for listening on the receiver socket
    …
    while (mylock) { …
        rxSocket.receive(packet);
    if (runThread) {
        triEnqueueMsg(tsiPortId, new TriAddressImpl(new byte[] {}),
            compPortId.getComponent(), rcvMessage);
    }
}
```

- Call **`triMap`** in the default SUT adapter
- **`triMap`** receives portIds of ports being mapped
- If succeeds, returns **`TRI_OK`**

`public TriStatus triUnmap (TriPortId compPortId, TriPortId tsiPortId)`

- Unmaps a test component port from a test system interface port
- Stops the receiver loop
- Parameter **`compPortId`** is a port reference to the test component port
- Parameter **`tsiPortId`** is a port reference to the test system interface port

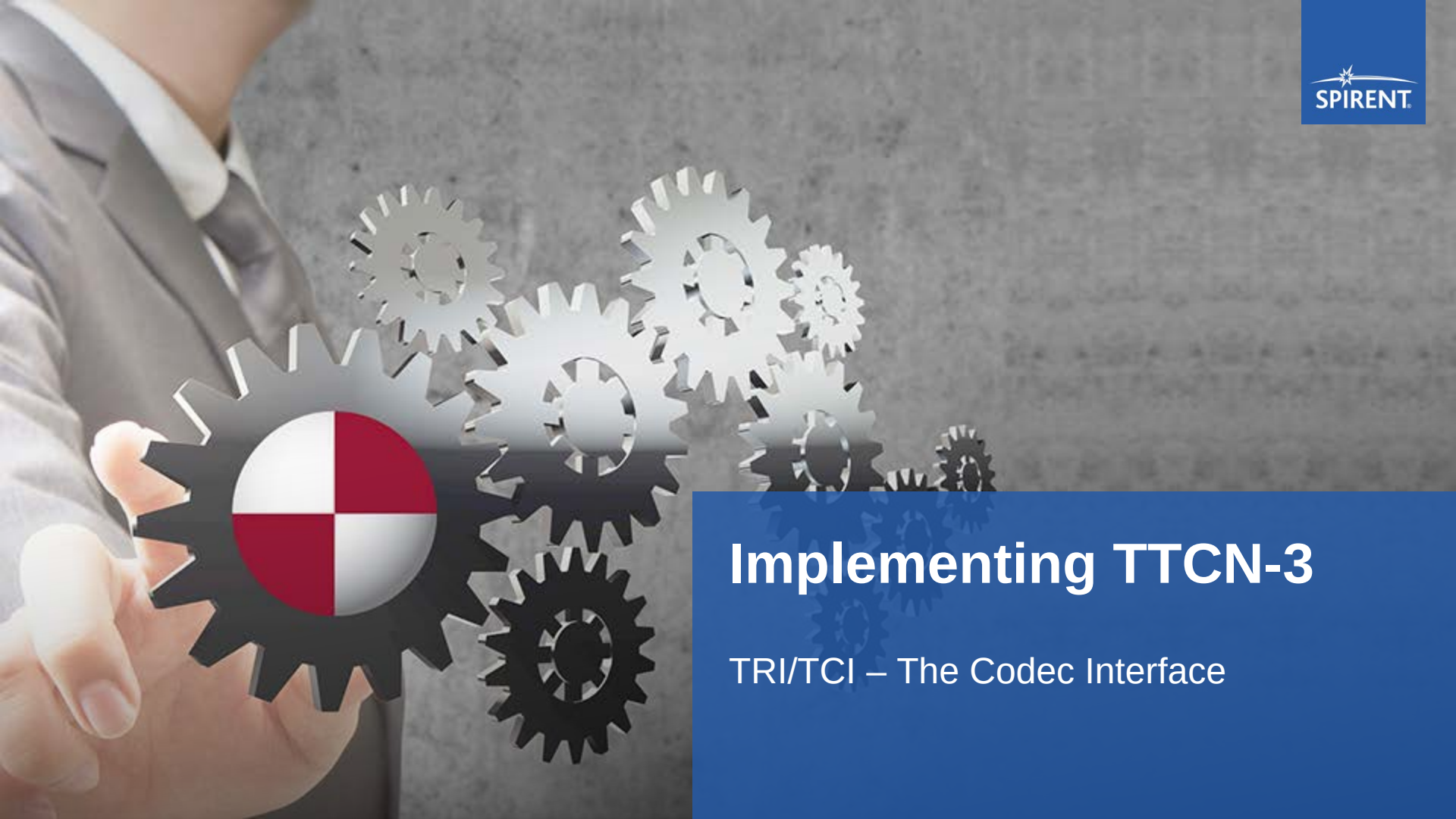
`public TriStatus triSend(TriComponentId componentId, TriPortId tsiPortId, TriAddress address, TriMessage sendMessage)`

- Is called when it executes a TTCN-3 send operation on a test component port
- Performs the real sending on the respective test system interface port
- Encoding of **`sendMessage`** has to be done prior to this operation call
- Parameter **`componentId`** is the sending test component
- Parameter **`tsiPortId`** is the test system interface port via the message is sent
- Parameter **`SUTaddress`** is the optional destination address within the SUT
- Parameter **`sendMessage`** is the encoded message to be sent

public TriStatus triSend(TriComponentId componentId, TriPortId tsiPortId, TriAddress address, TriMessage sendMessage)

```
public TriStatus triSend(final TriComponentId componentId,
        final TriPortId tsiPortId,
        final TriAddress address, final TriMessage sendMessage) {
     try {
        final byte[] mesg = sendMessage.getEncodedMessage();
        final InetAddress addr = InetAddress.getByName
                                 (remoteIPAddress);
        final DatagramPacket packet = new DatagramPacket
            (mesg, mesg.length, addr, remotePortNumber);
        // Sending message
        txSocket.send(packet);
        return new TriStatusImpl();
     }  catch (final IOException ioex) {
        return new TriStatusImpl(ioex.getMessage());
     }
}
```

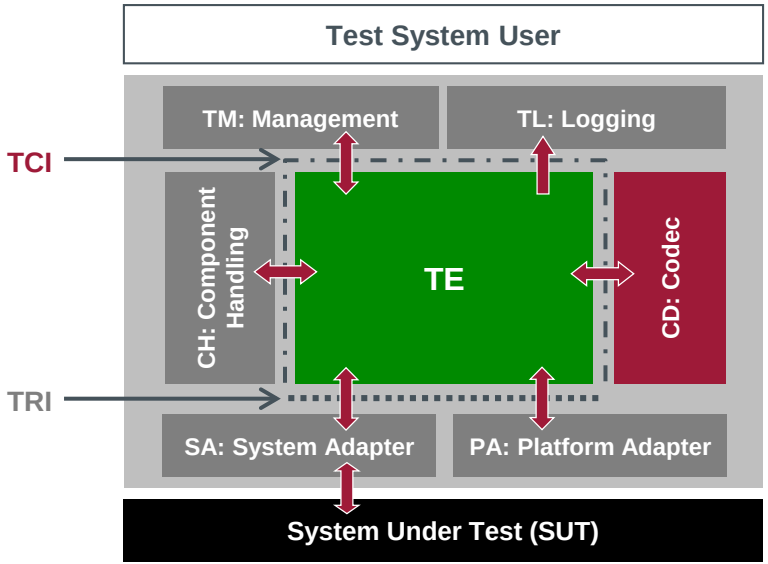
- Send the already encoded data using UDP



Implementing TTCN-3

TRI/TCI – The Codec Interface

A TTCN-3 test system



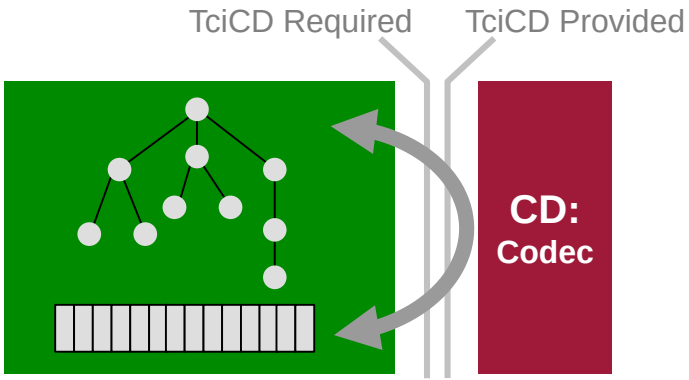
SUT – System Under Test
TE – TTCN-3 Executable
TM – Test Management
TL – Test Logging
CD – Codec
CH – Component Handling
SA – System Adapter
PA – Platform Adapter

ETSI ES 201 873-1 TTCN-3 Core Language (CL)
ETSI ES 201 873-6 TTCN-3 Control Interfaces (TCI)
ETSI ES 201 873-5 TTCN-3 Runtime Interface (TRI)

Why codec interface?

- TTCN-3 data has to be translated into a representation the SUT understands
- Two different tasks
  - Encoding
    - Internal TTCN-3 data representation to bitstring
    - Needs access to the TTCN-3 type and value system
  - Decoding
    - Bitstring to TTCN-3 data representation
    - Based upon a decoding hypothesis
    - TE may query multiple times for the decoding of the same bitstring

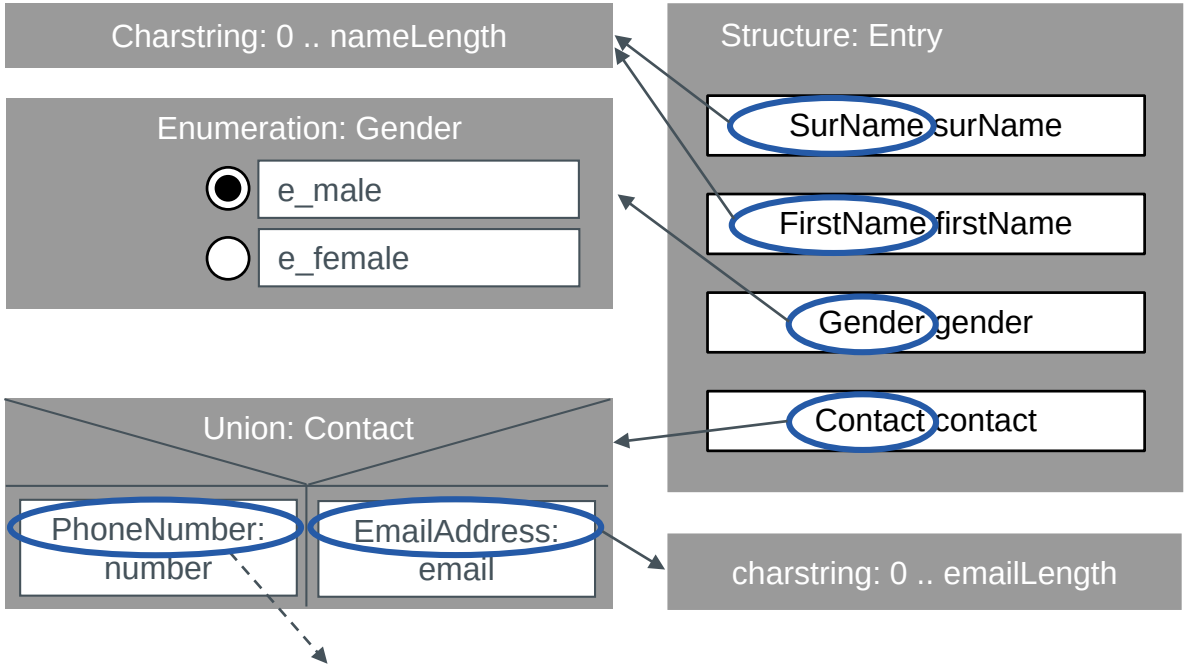
The codec and value interface



- Facts on the TTCN-3 type and value interface
  - TE maintains abstract type and data presentation
  - Codec translates between abstract and concrete presentation

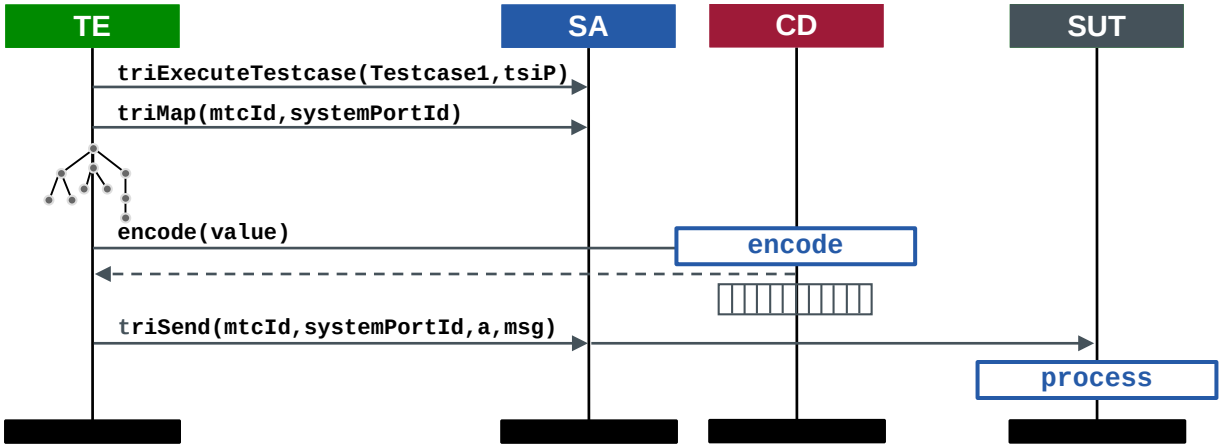
- Management of different codecs
- Complete set of provided (**`TciCDProvided`**) operations
  - **`TriMessageType encode (in Value value)`**
  - **`Value decode (in TriMessageType message, in Type hyp)`**

Data types used



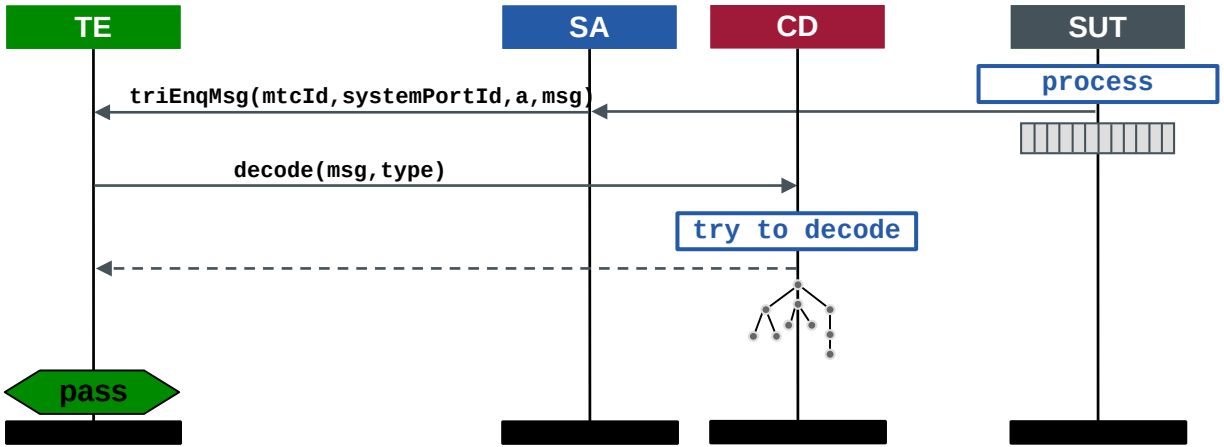
Dynamics of the codec (sending)

```
testcase Testcase1() runs on DNSTester system TSI {
        map(mtc:P, system:P);
        P.send(query);

        P.receive(answer);

        setverdict(pass);
}
```



Dynamics of the codec (receiving)

```
testcase Testcase1() runs on DNSTester system TSI{
        map(mtc:P, system:P);
        P.send(query);
        P.receive(answer);

        setverdict(pass);
}
```



The **decodingHypothesis**

- The interpretation of an arbitrary bitstring is context sensitive
- Example: What is **'56455300'O** ?
  - Four bytes as one octetstring: **'56455300'O**
  - An integer: **1447383808**
  - A charstring: **"YES"**
- Decode() can be read as follows
  - Try to decode the provided bitstring, with the appropriated decoding rules into a value of given type
  - If you succeed, return the value
  - If you fail, return **NULL**

Access to TTCN-3 data types and values

- Formal definition
  - Specification of abstract data types
    - Type for TTCN-3 types
    - Different ADT types for TTCN-3 values
  - Set of high-level operations define the functionality
- Practical usage
  - TTCN-3 environments provide functions/operations to
    - Access TTCN-3 types
    - Read existing TTCN-3 values and
    - Create new TTCN-3 values
  - Underlying philosophy behind the ADT operations is an object oriented model

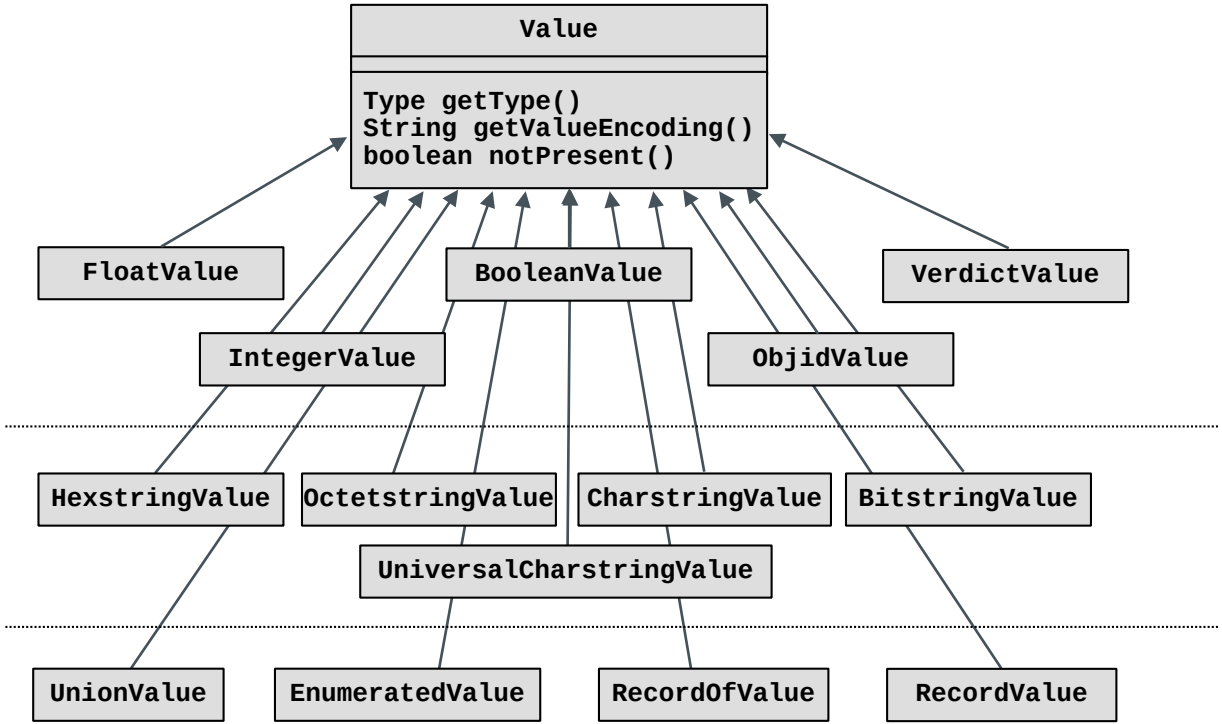
Abstract data type: **Type**

```
type record Entry {
   SurName surName,
   FirstName firstName,
    Gender gender,
   Contact contact
}

type union Contact {
   PhoneNumber number,
   EmailAddress email
}
```


Type	
TciModuleId	getDefiningModule()
String	getName()
TciTypeClass	getTypeClass()
String	getTypeEncoding()
Value	newInstance()

- **Type** represents every TTCN-3 type
- Only types defined in TTCN-3 modules can be accessed
- No creation of user defined types at the TCI
- ... but creation of new instances of given type!

ADT: Value

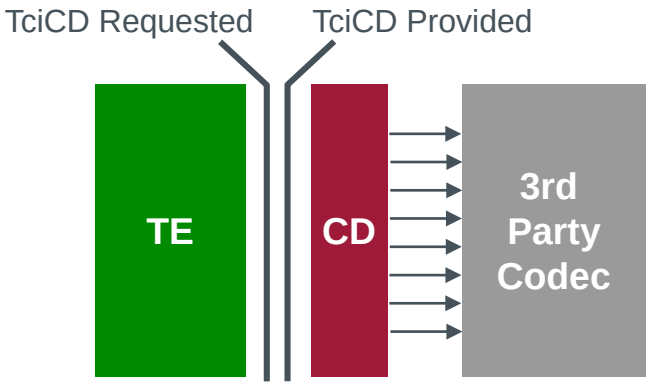

Value
Type getType()
String getValueEncoding()
boolean notPresent()
FloatValue
BooleanValue
VerdictValue
IntegerValue
ObjidValue
HexstringValue
OctetstringValue
CharstringValue
BitstringValue
UniversalCharstringValue
UnionValue
EnumeratedValue
RecordOfValue
RecordValue

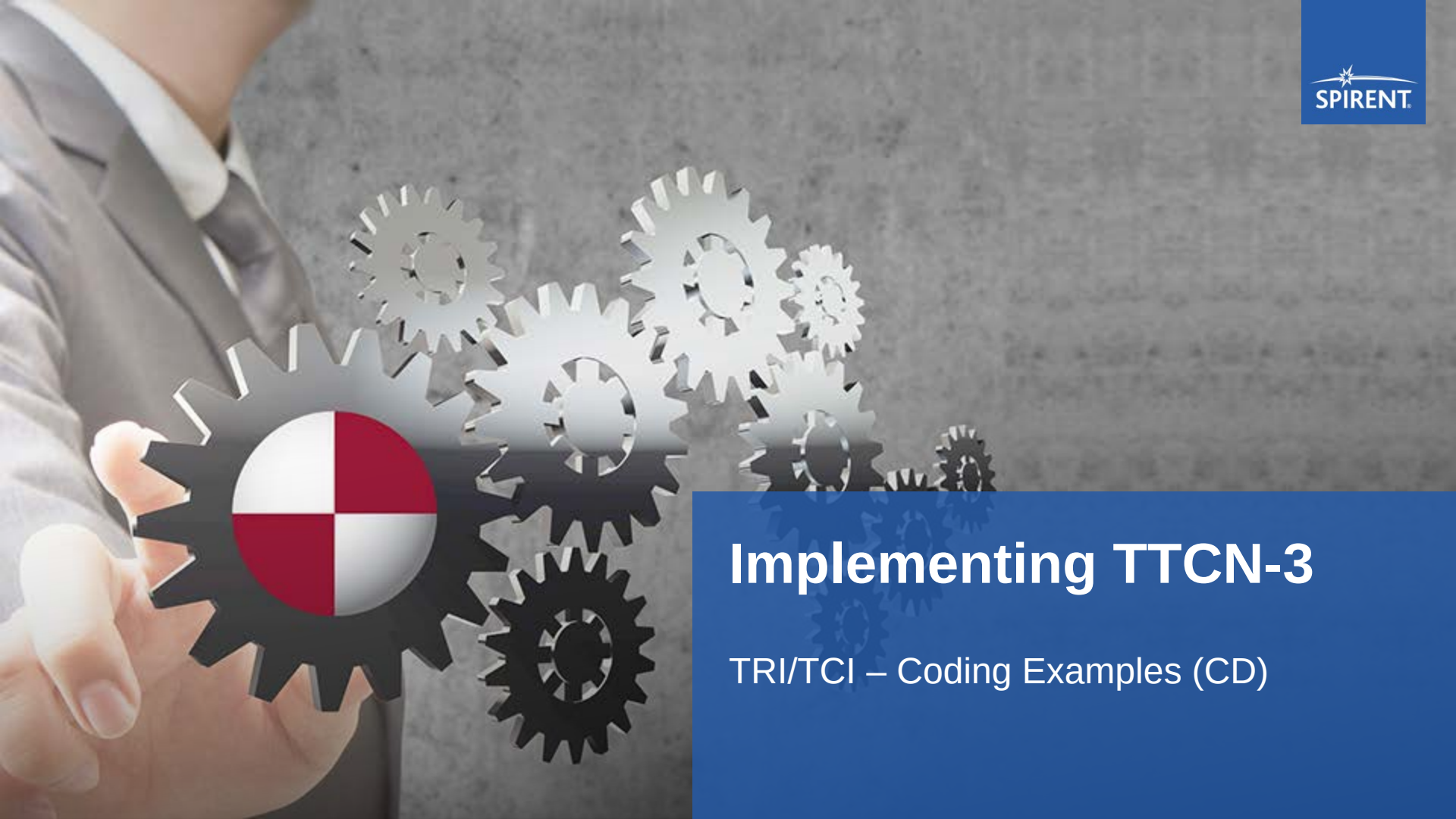
The codec interface

- Complete set of required (**`TciCDRequired`**) operations
  - **`Type getTypeForName(…)`**
  - **`Type getInteger()`**
  - **`Type getFloat()`**
  - **`Type getBoolean()`**
  - **`Type getChar()`**
  - **`Type getUniversalChar()`**
  - **`Type getObjid()`**
  - **`Type getCharstring ()`**
  - **`Type getUniversalCharstring ()`**
  - **`Type getHexstring ()`**
  - **`Type getBitstring()`**
  - **`Type getOctetstring ()`**
  - **`Type getVerdict()`**
  - **`void tciErrorReq(…)`**

The codec and value interface

- 3rd party codec provides
  - Operations to construct values
  - Operations to query values
  - Operations to encode values
  - Operations to decode bitstring TE
- CD provides
  - Operations to construct values
  - Operations to query values
- CD implementation maps TCI value structures into codec value structures





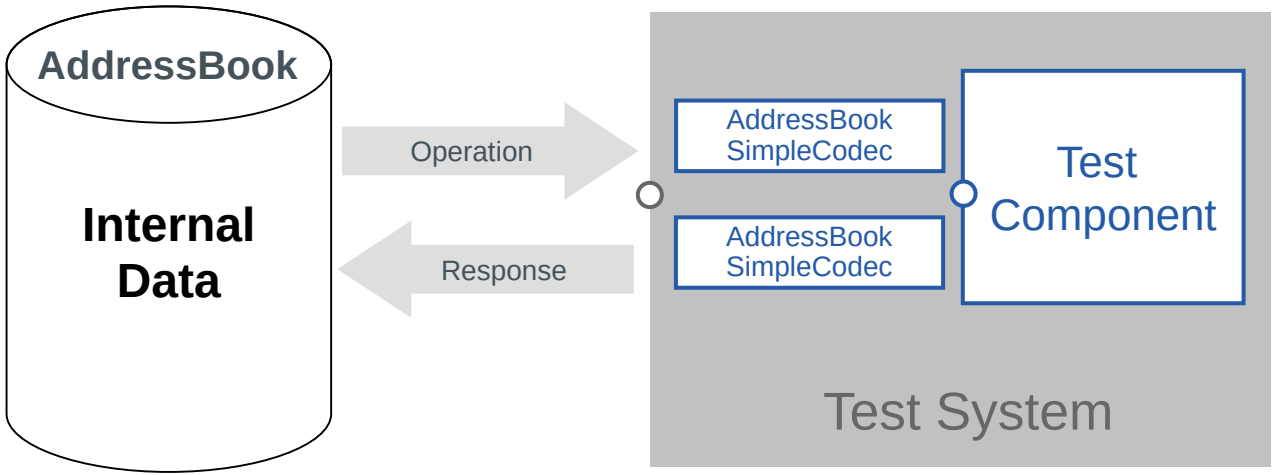
Implementing TTCN-3

TRI/TCI – Coding Examples (CD)

The codec

- In order to send real data over a TTCN-3 port, encoding functionality has to be provided to the TTCN-3 runtime behavior
- Incoming data has to be decoded into a TTCN-3 value, to allow matching against TTCN-3 template definitions
- Two entities have to be implemented
  - Encoder – TTCN-3 data to real world data
  - Decoder – real world data to TTCN-3 data

Codec practically



`AddressBookSimpleCodec` – TTCN-3 data to a bitstring

`AddressBookSimpleCodec` – response to TTCN-3 data

→ TTCN-3 data as defined in the TTCN-3 module

Relevant TTCN-3 definitions

```
type record Entry {
        SurName surName,
        FirstName firstName,
        Gender gender,
        Contact contact }

template Entry firstEntry := {
        surName := "Borowski",
        firstName := "Dirk",
        gender := e_male,
        contact := { email := "borowski@testingtech.com" }
}

type port addressBookPort message {
        out addEntry, getEntry;
        out clear_;
        in getEntryReply,userExists, sizeLimitReached, notFound;
}
```

Declaration of an Entry

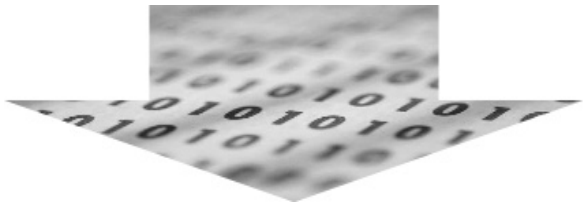
Declaration of Database Entry

Declaration of Interface

Encoding

- Must implement the interface **`org.etsi.ttcn.tci.TciCDProvided`**
- Encodes a TTCN-3 record into a message

```
template Entry firstEntry := {
   surName := "Borowski",
   firstName := "Dirk",
   gender := e_male,
   contact := { email := borowski@testingtech.com
   }
}
```



0100010100010111000101010010100010110100010010101010101

Encoding PDU type

```
public TriMessage encode(Value template) {
    // choose the PDU type
    if (template.getType().getName().equals("Entry")) {
        return encodeEntry((RecordValue) template);
    }
    return super.encode(template);
}
```

- Encodes only Entry types as defined in the TTCN-3 module
- Uses the helping classes to encode the data

Encoding type Entry

```
private TriMessage encodeEntry(RecordValue value) {
      bitpos = 0; // initialize bit position
      ByteArrayOutputStream out = new ByteArrayOutputStream();

      encodeCharstring(value, "surName", out);
      encodeCharstring(value, "firstName", out);
      encodeGender((EnumeratedValue) value.getField("gender"), out);
      encodeContact((UnionValue) value.getField("contact"), out);
      return new TriMessageImpl(out.toByteArray());
}
```

- Encodes the sub-fields of type Entry into a byte array
- Uses the helping method to encode the data

Encoding type Charstring

```
private void encodeCharstring(RecordValue value, String fieldName,
                              ByteArrayOutputStream out) {
     CharstringValue cs = (CharstringValue)
        value.getField(fieldName);
     // enocde charstring length with a 4 byte integer
     int length = cs.getLength();
     IntegerValue lengthValue = (IntegerValue)
        RB.getTciCDRequired()
     .getInteger().newInstance();
     lengthValue.setInt(length);
     super.encodeInteger(out, lengthValue);
     // encode the charstring itself
     super.encodeCharstring(out, cs);
}
```

- Add 4 bytes to indicate the charstring length
- Several other approaches are possible
- In many cases the length information is included in TTCN-3 type definition
- Use encodeCharstring in super class AbstractBaseCodec

Encoding type Gender and Contact

```
private void encodeGender(EnumeratedValue value,
                          ByteArrayOutputStream out) {
     // Enumeration is also implemented as IntegerValue, therefore
     // can be casted to IntegerValue
     out.write(((IntegerValue) value).getInt());
}
private void encodeContact(UnionValue value,
                          ByteArrayOutputStream out) {
     String variantName = value.getPresentVariantName();
     if (variantName.equals("number")) {
     out.write(0); // marker for 1st variant - PhoneNumber
     encodePhoneNumber(out,(RecordOfValue)value.getVariant(variantName));
     } else {
     out.write(1); // marker for 2nd variant – EmailAddress (charstring)
     encodeCharstring(out,(CharstringValue)value.getVariant(variantName));
     }
}
```

- Enumerated is implemented as integer, also user-assigned
- For each union variant a marker as integer is defined

Encoding type PhoneNumber

```
private void encodePhoneNumber(ByteArrayOutputStream out, RecordOfValue value) {
      // encode the length (element amount)
      int length = value.getLength();
      IntegerValue lengthValue = (IntegerValue) RB.getTciCDRequired()
      .getInteger().newInstance();
      lengthValue.setInt(length);
      super.encodeInteger(out, lengthValue);
      // encode each digit as one byte integer
      for (int i = 0; i < length; i++) {
            out.write(((IntegerValue) value.getField(i)).getInt());
      }
}
```

- Encoding RecordOfType add a byte as type length information
- Single elements/digits are encoded in a loop

Decoding

- Must implement the interface **`org.etsi.ttcn.tci.TciCDProvided`**
- Decodes a received bytestring into the TTCN-3 data

```
01000101
00010111
00010101
00101000
10110100
01001010
10101010
```

```
type charstring FirstName
    length (0 .. 20) ;

type record userExists {
    FirstName firstName
}
```

Decoding PDU type

```
public Value decode(TriMessage message, Type decodingHypothesis) {
      // initialize bit position
      bitpos = 0;
      if (decodingHypothesis.getName().equals("Entry")) {
            RecordValue result = (RecordValue)
               decodingHypothesis.newInstance();
            byte[] encodedMessage = message.getEncodedMessage();
            decodeEntry(encodedMessage, result);
            return result;
      }
      return super.decode(message, decodingHypothesis);
}
```

- If the decodingHypothesis is the PDU type, start decodeEntry
- A new instance is generated over <Type>.newInstance()
- If the message does not fit to the given decoding hypothesis → return null

Decoding type Entry

```
private void decodeEntry(byte[] message, RecordValue result) {
      int length;
      // surName
      CharstringValue surName = decodeCharstring(message);
      result.setField("surName", surName);
      // firstName
      CharstringValue firstName = decodeCharstring(message);
      result.setField("firstName", firstName);
      // gender
      int gender = message[bitpos / 8];
      bitpos += 8; // skip one decoded byte
      IntegerValue genderValue = (IntegerValue)
         result.getField("gender");
      genderValue.setInt(gender);
      result.setField("gender", genderValue);
      // contact
      UnionValue contact = (UnionValue) result.getField("contact");
      decodeContact(message, contact);
      result.setField("contact", contact);
}
```

- Decode different fields

Decoding Charstring

```
private CharstringValue decodeCharstring(byte[] message) {
      // read the charstring length
      int length =
         super.createIntegerValue(message).getInt();
      CharstringValue surName =
         super.createCharstringValue(message, length * 8);
      return surName;
}
```

- Read first the charstring length
- Use super class createCharstringValue

Decoding subtype Union (Contact)

```
private void decodeContact(byte[] message, UnionValue contact) {
      int variant = message[bitpos / 8];
      bitpos += 8; // skip one decoded byte
      if (variant == 0) {
      RecordOfValue number = (RecordOfValue)
         contact.getVariant("number");
      decodeNumber(message, number);
      contact.setVariant("number", number);
      } else {
      CharstringValue email = decodeCharstring(message);
      contact.setVariant("email", email);
      }
}
```

- Depending on the varian the union type is decoded as number or email

Decoding subtype RecordOf (Number)

```
private void decodeNumber(byte[] message, RecordOfValue number) {
      int length = super.createIntegerValue(message).getInt();
      number.setLength(0); // initialize value
      for (int i = 0; i < length; i++) {
      IntegerValue digitValue =
         (IntegerValue)number.getElementType().newInstance();
      int digit = message[bitpos / 8];
      bitpos += 8; // skip one decoded byte
      digitValue.setInt(digit);
      number.appendField(digitValue);
      }
}
```

- Read the first byte to get the length of the number list (RecordOf)
- Create new integer instance and assign it with digit value, append to the list

Managing the codecs

```
public TciCDProvided getCodec(String encodingName) {
        // ...
        // Some frame work code

        // You always need a constructor with RB
        TciCDProvided codec = new AddressBookSimpleCodec(RB) ;

        // store the codec for later usage
        encoders.put(encodingName, codec);

        return codec;
    }
```

- Tool dependent functionality, not specified in the standard
- Spirent brokers the **codecs** in the Test Adapter
- **`encodingName`** taken from TTCN-3 encoding attributes

Summary

- TTCN-3 provides two different interfaces
  - TTCN-3 Runtime Interface
  - TTCN-3 Control Interface
- Provide a complete, independent set of operations for implementing TTCN-3 test suites
- Some components
  - are provided by the TTCN-3 tools,
  - others are provided by TTCN-3 solutions
  - and others have to be implemented manually.
- Result: High flexibility and reuse of software components within the test system development