The TTCN-3 Language

An Introduction

Speaker

Theofanis Vassiliou-Gioles

CEO Testing Technologies, Berlin (Germany)

vassiliou@testingtech.de

www.testingtech.de

Agenda

- Motivation
- The test specification language TTCN-3
- Implementing TTCN-3 test specifications

The TTCN-3 Language

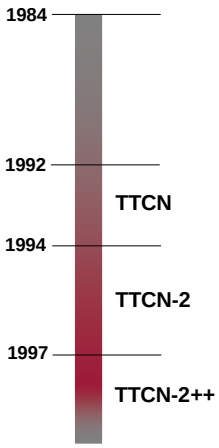
Introduction to TTCN-3

Motivation

Design Principles of TTCN-3

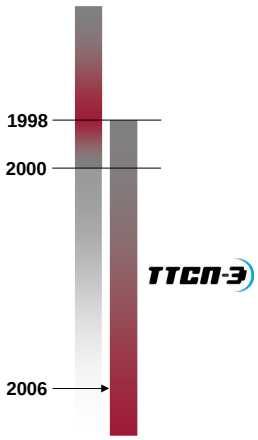
- One test technology for different tests
  - Distributed, platform-independent testing
  - Integrated graphical test development, -documentation and -analysis
  - Adaptable, open test environment
- One test technology for distributed IT and Telco systems

History (1)



- TTCN (1992)
  - Published as an ISO standard
  - Tree and Tabular Combined Notation
  - Used for protocol testing
    - GSM, N-ISDN, B-ISDN
- TTCN-2/2++ (1997)
  - Concurrent tests
  - Modularization
  - Manipulate external data
  - Rather for conformance testing

History (2)



- TTCN-3 (2000)
  - Testing and Test Control Notation
  - ETSI STFs
  - Proper language
    - Well defined syntax and semantics
  - Enhanced communication, configuration and control
  - Standard test specification
    - SIP, SCTP, HiperLan, HiperAccess, IPv6 ...
- TTCN-3 (2006): Edition3

➡ Changing requests, extension proposals

The TTCN-3 Language

Introduction to TTCN-3

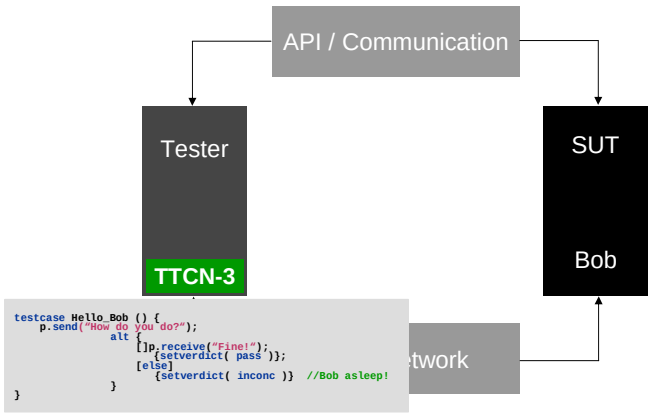
Basic Concepts

What is TTCN-3?

- Testing and Test Control Notation
- Internationally standardized testing language for formally defining test scenarios and their implementation
- Designed purely for testing
- Taking the best bits of TTCN-2 and combining them with a new, more powerful textual notation

```
testcase Hello_Bob () {
    p.send(“How do you do?“);
        alt {
            []p.receive(“Fine!“);
                {setverdict( pass )};
            [else]
                {setverdict( inconc )}  //Bob asleep!
        }
}
```


TTCN-3 Execution



Application Areas

- New application areas
  - Software testing
  - Text-based protocols …
- Additional communication paradigm
  - Message-based communication
  - Procedure-based communication
- Different kinds of testing
  - Functional testing
  - Conformance testing
  - Scalability testing …
- Covering larger range within development cycle
  - From unit to integration testing

Main Aspects of TTCN-3

- Triple C
  - **C**onfiguration: Dynamic concurrent test configurations with test components
  - **C**ommunication: Various communication mechanisms (synchronous and asynchronous)
  - **C**ontrol: Test case execution and selection mechanisms
- Features
  - Well-defined syntax, static and operational semantics
  - Different presentation formats
  - Module concept
  - Extendibility via attributes, external function, external data
  - Harmonized with ASN.1

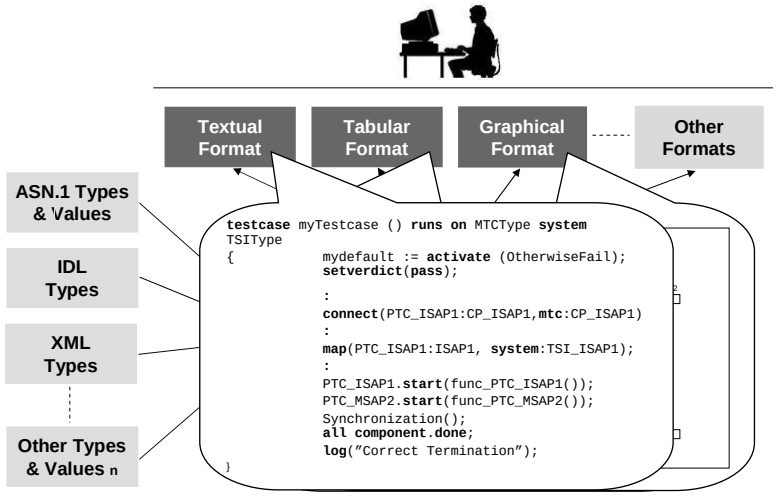
Differences TTCN-2 / TTCN-3

- Configuration
  - Static configuration with configuration tables
  - Dynamic configuration with arbitrary amount of components
  - Differentiation between PCOs and CPs
  - One port concept
- Communication
  - Asynchronous communication only
    - Abstract Service Primitives
    - Protocol Data Unit
  - Procedure and message-based communication
    - Procedures
    - Messages
- Control
  - Static selection of test cases via selection expression
  - Complete high level control flow mechanisms

Differences TTCN-2 / TTCN-3

- Externalisation
  - Test suite operations
  - External function
  - PICS / PIXIT
  - Module parameters
- Data types, values
  - TTCN-2 / ASN.1
  - TTCN-3, ASN.1, IDL, XML, ...
- Modularisation
  - Possible but seldom used
  - Central concept
- Extensibility
  - Not possible
  - Attributes, languages
- Methodology
  - Conformance Testing Methodology Framework (CTMF) (ISO 9646)
  - No specific
- Presentation
  - Tabular, machine processable
  - Textual, graphical, tabular, ...
- Implementation
  - No runtime interfaces
  - TTCN-3 Runtime Interfaces, TTCN-3 Control Interfaces
- Acronym
  - Tree and Tabular Combined Notation
  - Testing and Test Control Notation

TTCN-3

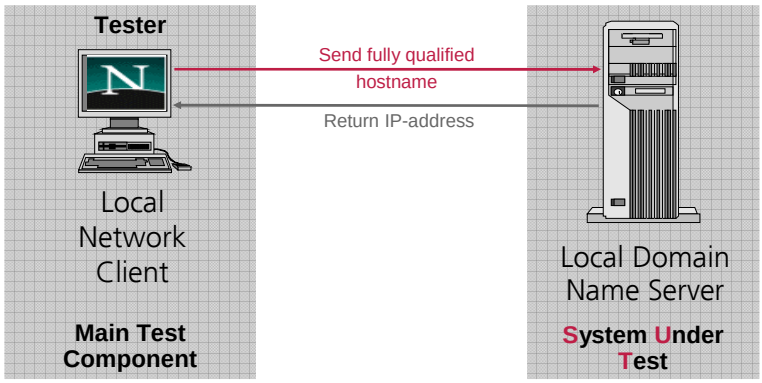


TTCN-3 Edition 3 Standards

- ETSI ES 201 873-1 TTCN-3 Core Language (CL)
- ETSI ES 201 873-2 TTCN-3 Tabular Presentation Format (TFT)
- ETSI ES 201 873-3 TTCN-3 Graphical Presentation Format (GFT)
- ETSI ES 201 873-4 TTCN-3 TTCN-3 Semantics
- ETSI ES 201 873-5 TTCN-3 TTCN-3 Runtime Interface (TRI)
- ETSI ES 201 873-6 TTCN-3 TTCN-3 Control Interfaces (TCI)
- ETSI ES 201 873-7 Integration of ASN.1
- ETSI ES 201 873-8 Integration of IDL
- ETSI ES 201 873-9 Integration of XML

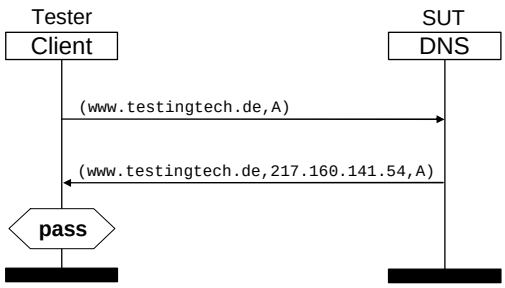
- Standard available for download at http://www.etsi.org/ptcc
- Testing Tech tools support Edition 3

TTCN-3 By Example



DNS Server – Test Purpose

- The Internet's Domain Name System (DNS) service offers hostname to IP Address resolution
- Communication is message based (UDP)
- We want to test the correct resolution www.testingtech.de => 17.160.141.54



TTCN-3 Modules

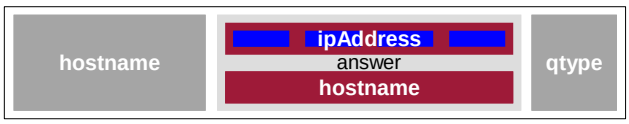
- Main building block of TTCN-3 is a module
  - Unit of compilation
  - Contains definitions
  - And an optional control part0

```
module DNS {

 // module definitions

 // module control (optional)

}
```


Module Definitions (1)

- Module definitions
  - **Type definitions**
  - Port definitions
  - Component definitions
  - Test case
  - Templates
  - Control part

```
type record DNSQuery {
        charstring  hostname,
        AnswerType  answer optional,
        QueryType   qtype
}
type union AnswerType {
        Byte        ipAddress[4],
        charstring  hostname
}
type integer Byte (0 .. 255);
type enumeration QueryType {
        A, NS, CNAME, MX
}
```



Module Definitions (2)

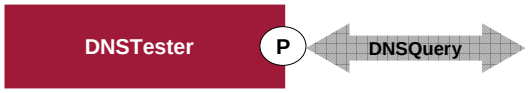
- Module definitions
  - Type definitions
  - **Port definitions**
  - **Component definitions**
  - Test case
  - Templates
  - Control part

Port definitions

```
type port DNSPort message {
    inout DNSQuery

    // a port may send/receive messages
    // of more than one type
}
```

Component definitions

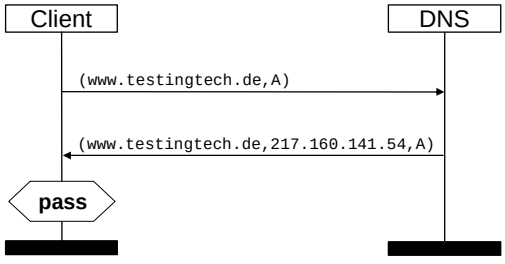
```
type component DNSTester {
    port DNSPort P

    // a component may have more than one port
}
```



Module Definitions (3)

- Module definitions
  - Type definitions
  - Port definitions
  - Component definitions
  - **Test case**
  - Templates
  - Control part

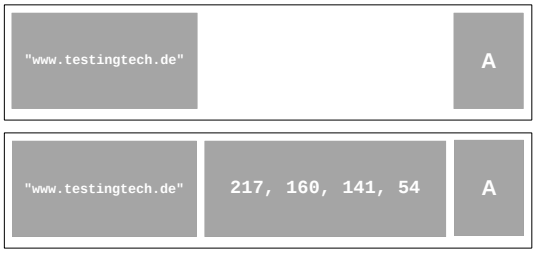
```
testcase Testcase1() runs on DNSTester {
        P.send(query);
        P.receive(answer);
        setverdict(pass);
}

// there may be more than one in a module
```



Module Definitions (4)

- Module definitions
  - Type definitions
  - Port definitions
  - Component definitions
  - Test case
  - **Templates**
  - Control part

```
type record DNSQuery {
  charstring hostname,
  AnswerType answer optional,
  QueryType qtype
}
type union AnswerType {
  IPAddress ipAddress,
  charstring hostname
}
template DNSQuery query := {
  hostname := "www.testingtech.de",
  answer := omit,
  qtype := A
}
template IPAddress ipAddress := {
  ipaddress := {217,160,141,54}
}
```

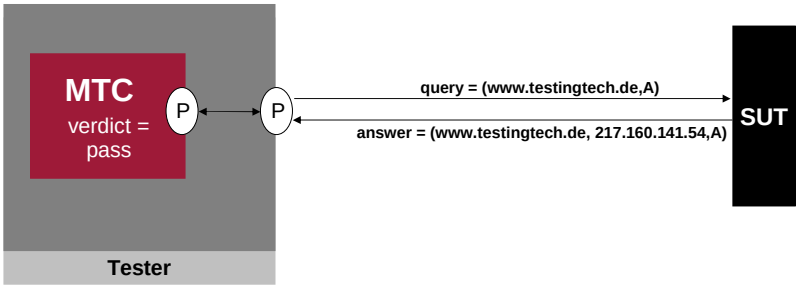


Module Definitions (5)

- Module definitions
  - Type definitions
  - Port definitions
  - Component definitions
  - Test case
  - Templates
  - **Control part**
    - Controls the execution of test cases

```
control {

    execute(Testcase1(), 5.0);
    while( /* condition */) { };

    // more testcases might follow
    // C-like control structures available

}
```

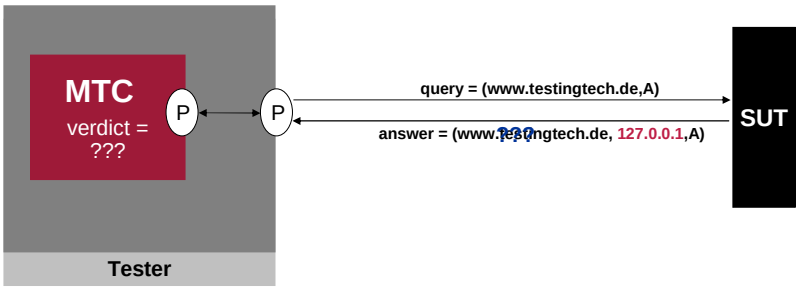
Execution of a Test Case




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

}
```

Is this test case definition adequate?

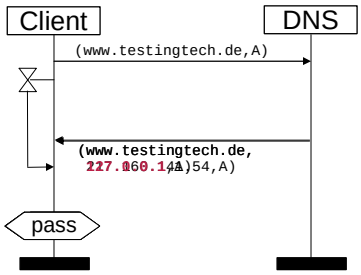
Dealing with Erroneous Behavior (1)



- **`P.receive (answer)`** blocks until it receives a message that matches answer.
- Any other message does not unblock the tester, which then blocks forever.
- If no message is received, the tester will also block forever.

Dealing with Erroneous Behavior (2)

```
testcase Testcase2() runs on DNSTester {
    timer t := 5.0;
    P.send(query);

    t.start;

    alt {
        [] P.receive(answer) {
                setverdict(pass);
            }
        [] P.receive { // any message
                setverdict(fail);
            }
        [] t.timeout {
                setverdict(inconc);
            }
    }
    stop;
}
```



Code Reusability – Altsteps and Defaults

```
alt {
  [] P.receive(answer) {
        setverdict(pass);
  }
  [] P.receive { // any message
        setverdict(fail);
  }
  [] t.timeout {
        setverdict(inconc);
  }
}
```

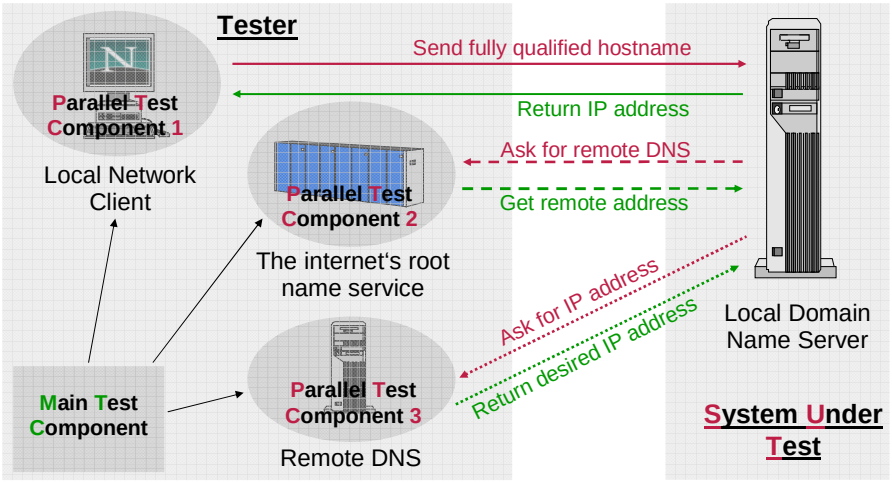
refactor

```
altstep RefactoredAltstep()
        runs on DNSTester {
    [] P.receive { // any message
          setverdict(fail);
    }
    [] t.timeout {
          setverdict(inconc);
    }
}
```

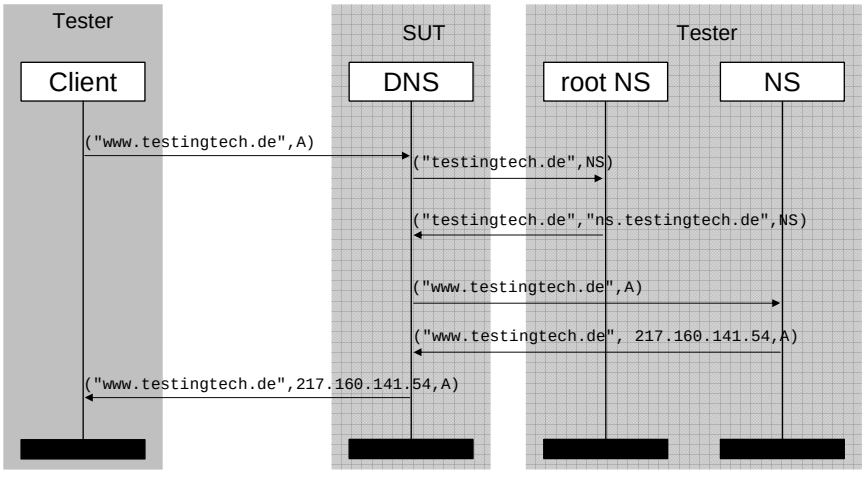
becomes

```
var default d := activate(RefactoredAltstep());
P.send(query);
t.start;
P.receive(answer);
setverdict(pass);
```

Non-Local DNS Query (1)



Non-Local DNS Query (2)



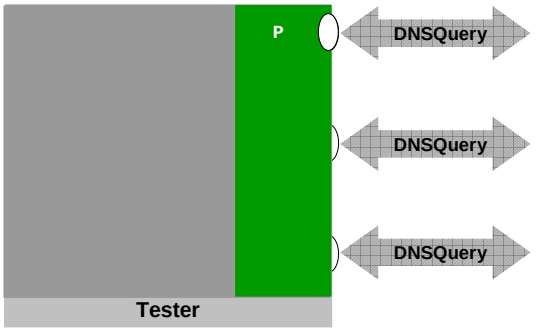
From Simple To Complex Test Scenarios

- Test system needs more interfaces
  - Test system interface has to be extended
- Additional test behavior needed at additional test interfaces
  - Behavior of Local Network Client already covered in **`Testcase2`**
  - Behavior of RootNS and NS required
- Test case that combines all pieces

Parallel Test Components (1)

- Test system interface

```
type component DNSTestSystemInterface {
    port DNSPort CLIENT;
         port DNSPort ROOT;
    // a component may have more than one port
}
```



From Test Case to Test Function

- Functions define the behavior of the parallel test components

```
testcase Testcase2() runs on DNSTester {
  var default d := activate(RefactoredAltstep());
  timer t := 5.0;
  P.send(query); t.start;
  P.receive(answer);
  setverdict(pass);
  stop;
}
```

becomes

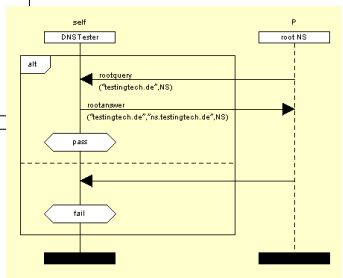
```
function ClientBehaviour() runs on DNSTester {
  var default d := activate(RefactoredAltstep());
  timer t := 5.0;
  P.send(query); t.start;
  P.receive(answer);
  setverdict(pass);
  stop;
}
```


Additional Test Behavior

- Simple „react-on-request“ behavior

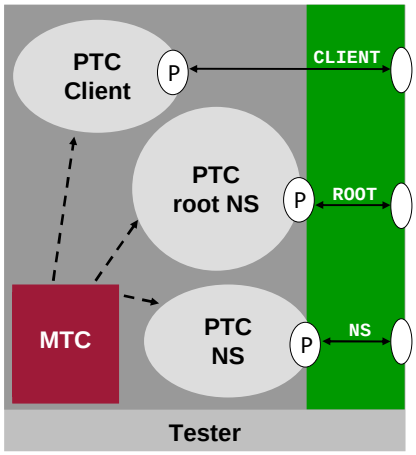
```
function RootBehaviour() runs on DNSTester {
    alt {  [] P.receive(rootquery) {
                  P.send(rootanswer);
                  setverdict(pass);
           }
           [] P.receive {
                  setverdict(fail);}
    }
}
```

```
function NSBehaviour() runs on DNSTester {
    alt {  [] P.receive(nsquery) {
                  P.send(nsanswer);
                  setverdict(pass);
           }
           [] P.receive {
                  setverdict(fail);}
    }
}
```



Dynamic Configuration

```
testcase Testcase3() runs on MTC
         system TestSystemInterface {

   var DNSTester ClientComp, RootComp,
       NSComp;

   ClientComp := DNSTester.create;
   RootComp   := DNSTester.create;
   NSComp     := DNSTester.create;

   map(ClientComp:P, system:CLIENT);
   map(RootComp:P,   system:ROOT);
   map(NSComp:P,     system:NS);

   ClientComp.start(ClientBehaviour());
   RootComp.start  (RootBehaviour());
   NSComp.start    (NSBehaviour());

   ClientComp.done;
   // block until ClientComp is done
   stop;
}
```



- **Re-configuration during run time is possible**

Procedure-based Communication (1)

- DNS also allows queries over TCP/IP connections
  How can this be adequately tested?
  - Signature definitions

```
signature DNSCall(
        inout charstring hostname,
        out   AnswerType ans,
        inout QueryType  qtype
);
```

  - Ports for procedure-based communication

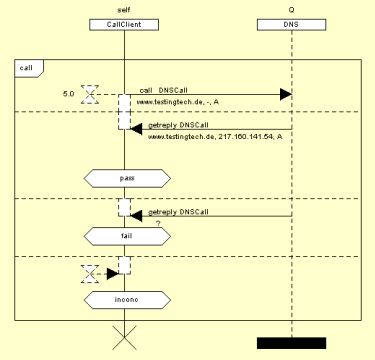
```
type port DNSCallPort procedure {
        out DNSCall
}
```

  - Component

```
type component DNSCallComponent  {
        port DNSCallPort Q
}
```

Procedure-based Communication (2)

```
testcase Testcase4() runs on DNSCallComponent {
   Q.call(DNSCall:{"www.testingtech.de", -, A}, 5.0);
   {
       [] Q.getreply(DNSCall:{"www.testingtech.de",{217.160.141.54},A}) {
            setverdict(pass)
       }
       [] Q.getreply(DNSCall: ?) {
            setverdict(fail)
       }
       [] Q.catch(timeout) {
            setverdict(inconc)
       }
   }
   stop;
}
```



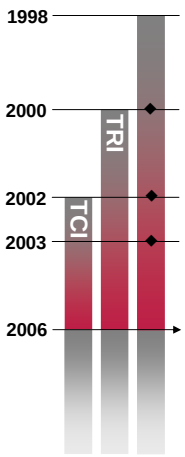
The TTCN-3 Language

Introduction to the Runtime Environment (TRI/TCI)

TTCN-3

- The Testing and Test Control Notation
- The standardized test specification and test implementation language
- Wider scope of application
  - Applicable to many kinds of test applications, not just conformance
    - i.e. also for development, system, integration, interoperability, scalability … testing
  - Applicable in the telecom and datacom domain
  - Used both
    - For standardized test suites and
    - As a generic solution in software development

History of TTCN-3

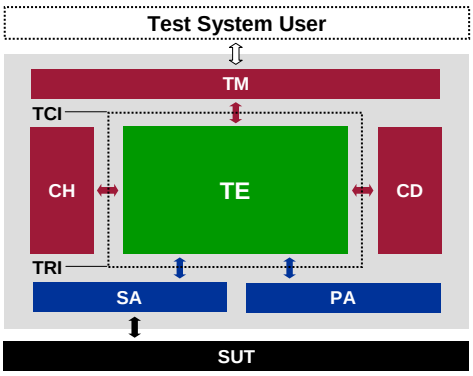


- TTCN-3 was published as ETSI standard in Oct. 2000
- Works on standardizing the runtime interfaces started immediately
- Main drivers
  - NOKIA, Ericsson
  - Testing Technologies, Telelogic
  - FOKUS, TU Berlin
- TRI - TTCN-3 Runtime Interface
  - First draft released Oct. 2001
  - First version released 2002
  - Current version dated 02/2003
- TCI – TTCN-3 Control Interfaces
  - Work for TCI started after v1.0 of TRI
  - v1.0 of TCI released 03/2003
- Ongoing Task in the maintenance group

Standard Overview

1.	Core Language	TTCN-3 Concepts and Syntax
2.	Tabular Presentation Format	Presentation Formats
3.	Graphical Presentation Format	
4.	Operational Semantics	Semantics Execution Environment
5.	TTCN-3 Runtime Interfaces (TRI)	
6.	TTCN-3 Control Interfaces (TCI)	
7.	ASN.1 to TTCN-3	Language Mappings
8.	IDL to TTCN-3	
9.	XML to TTCN-3	
10.	…	

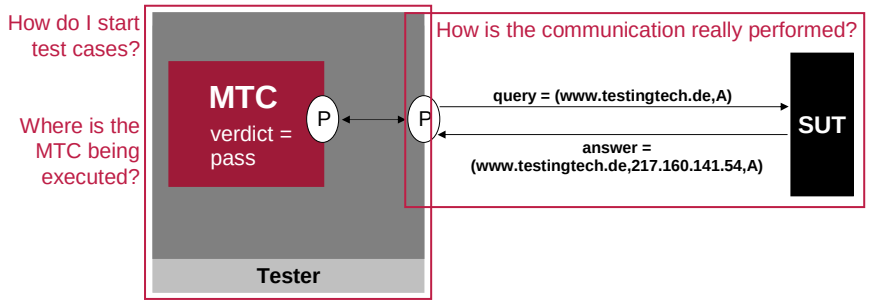
A TTCN-3 Test System



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

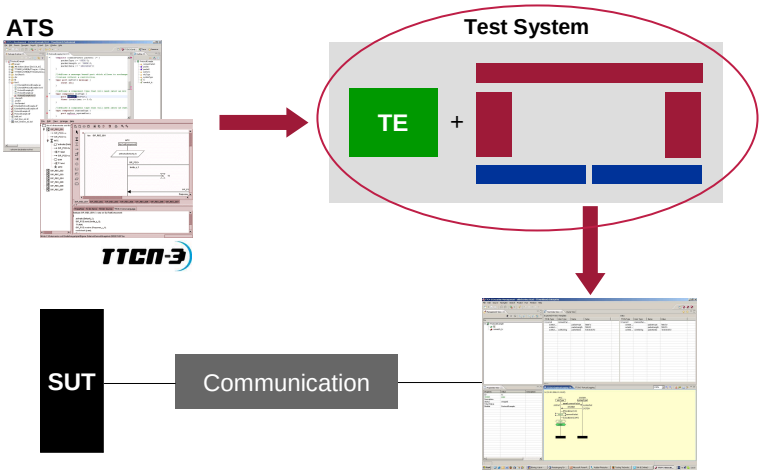
ETSI ES 201 873-1 TTCN-3 Core Language (CL)
ETSI ES 201 873-5 TTCN-3 Runtime Interface (TRI)
ETSI ES 201 873-6 TTCN-3 Control Interfaces (TCI)

Example – Test Case




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


Implementation



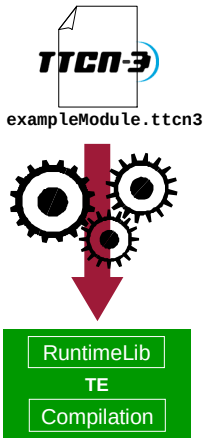
Steps to Implement TTCN-3

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

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



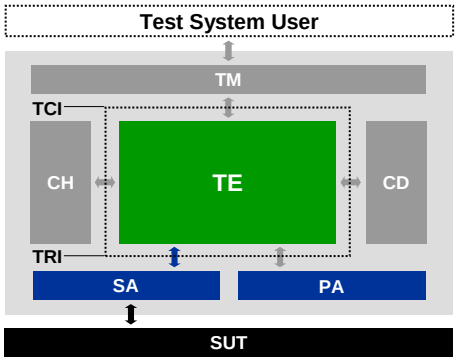
Steps to Implement TTCN-3

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

Steps To Implement TTCN-3

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

TRI – Communication Adaptation



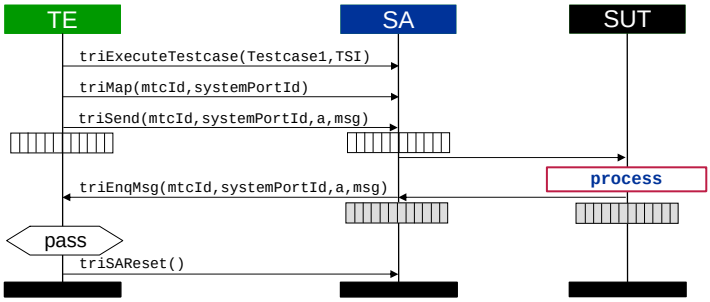
- Facts on the TTCN-3 Runtime Interfaces (TRI)
  - Standardized (part 5)
  - Language independent specification
  - Multi-vendor support

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

Dynamics of TRI SA

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



THANK YOU!

Questions?