SIEMENS

An Introduction to TTCN-3

Prof. Dr. Jens Grabowski
University Göttingen, Germany

Dr. Andreas Ulrich
Siemens AG, München, Germany

About the speakers 1(2)

- **Jens Grabowski**
  - Is professor for applied computer science at the Institute for Informatics of the University of Göttingen and head of the research group “Software Engineering for Distributed Systems”

  - Has the testing-oriented research interests
    - Test methodology, test specification, automatic and user-guided test generation, non-functional testing, testing languages
  - Is in the TTCN-3 business since the beginning

About the speakers 2(2)

- Andreas Ulrich
  - Is a Principal Engineer at Siemens' Corporate Technology Division, Software & Engineering Department in München, Germany
  - Provides consultancy services within the company in the area of testing and quality assurance for large software projects
  - Received his PhD in computer science from Magdeburg University in 1998
  - Is active in the research area of software testing
  - Member of the ETSI TTCN-3 maintenance group

Table of contents

PART I: Overall view of TTCN-3

What is TTCN-3
Main new aspects of TTCN-3
TTCN-3 series of standards
Concepts
TTCN-3 language elements
TTCN-3 test behavior specification
Attributes, grouping, and import

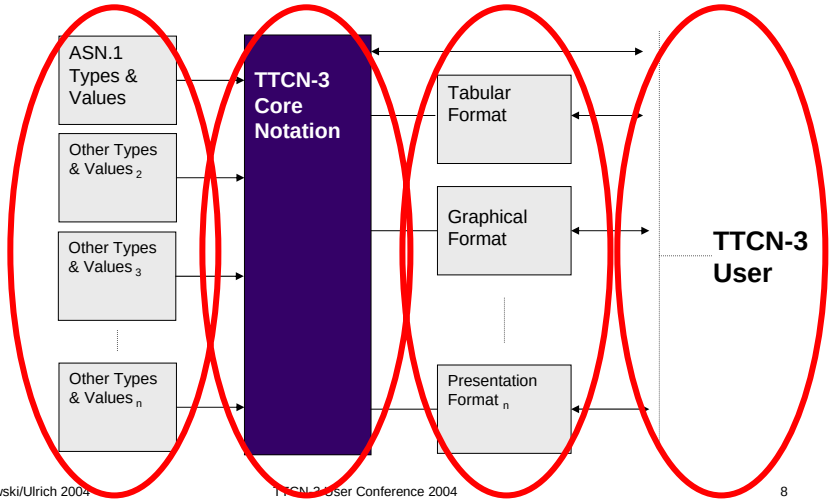
What is TTCN-3

- A standardised test specification and test implementation language
- Developed based on the experiences from previous TTCN versions
- Applicable for all kinds of black-box testing for reactive and distributed systems, e.g.
  - Telecom systems (ISDN, ATM);
  - Mobile (telecom) systems (GSM, UMTS);
  - Internet (has been applied to IPv6);
  - CORBA based systems.

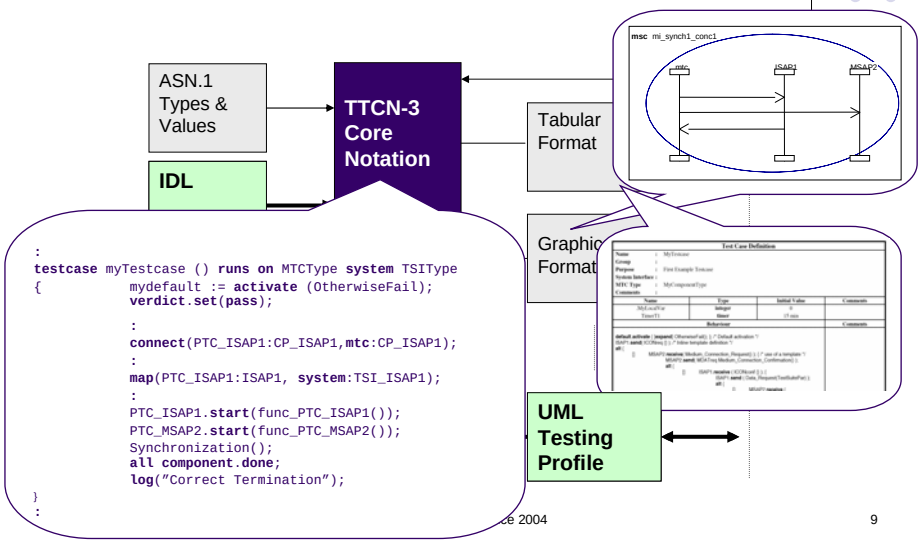
Main new aspects of TTCN-3

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

TTCN-3 series of standards 1(3)



TTCN-3 series of standards 2(3)



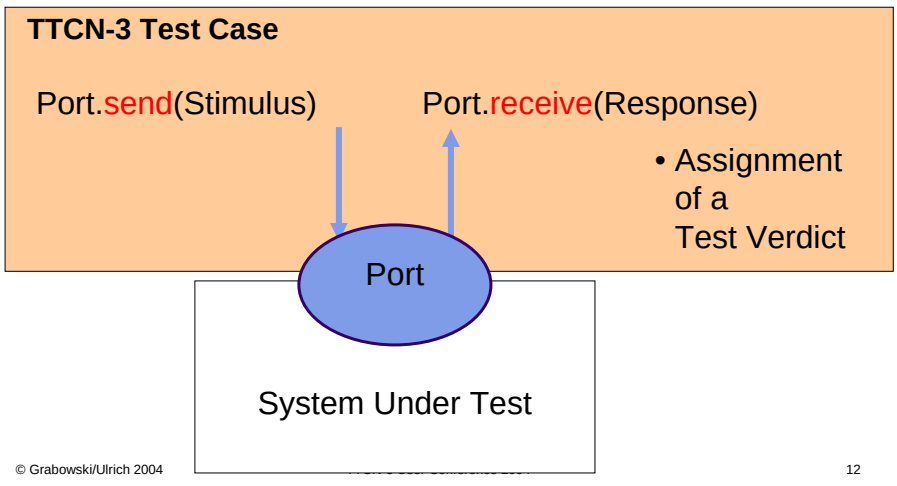
TTCN-3 series of standards 3(3)

- European Standard (ES) in 6 parts
  - ES 201 873-1: TTCN-3 Core Language
  - ES 201 873-2: TTCN-3 Tabular Presentation Format (TFT)
  - ES 201 873-3: TTCN-3 Graphical Presentation Format (GFT)
  - **ES 201 873-4: TTCN-3 Operational Semantics**
  - **ES 201 873-5: TTCN-3 Runtime Interface (TRI)**
  - **ES 201 873-6: TTCN-3 Control Interface (TCI)**
- Additional ETSI Technical Specification (TS)
  - TS 102 219: The IDL to TTCN-3 Mapping

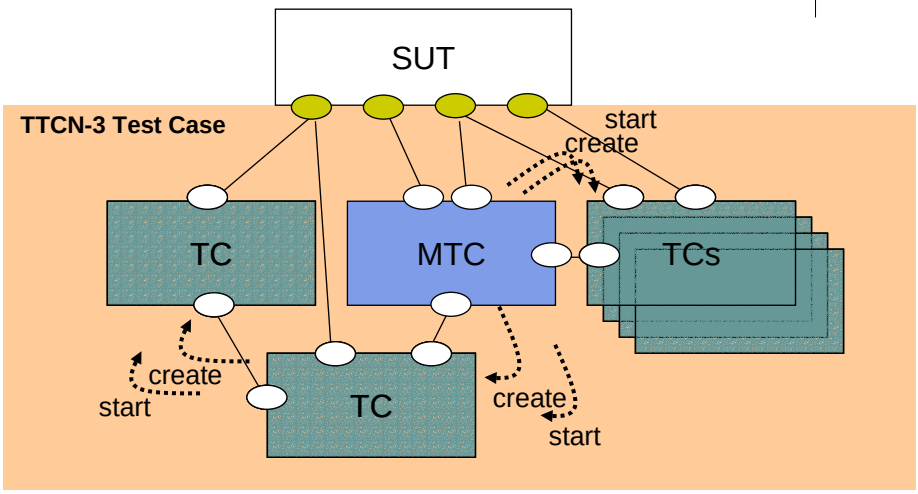
Concepts

- Black-Box Testing with TTCN-3
- Test Configuration
- Test Components
- Communication Ports
- Test Verdicts
- Main Elements of TTCN-3

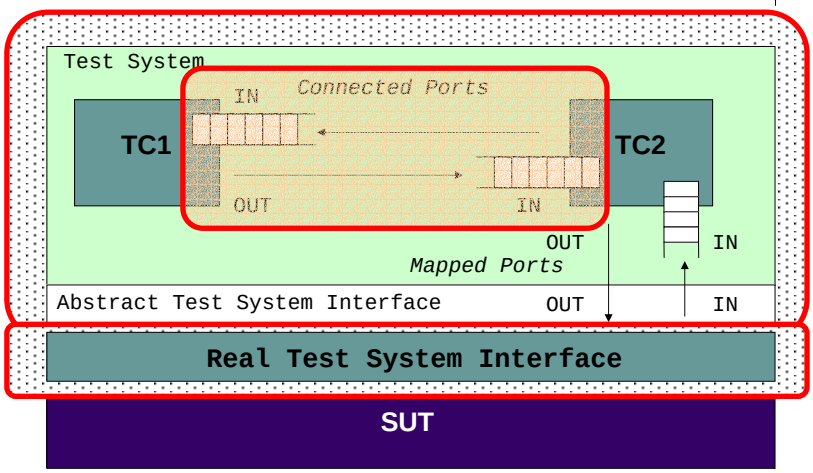
Concepts – Black-box testing with TTCN-3



Concepts – Test configuration 1(2)

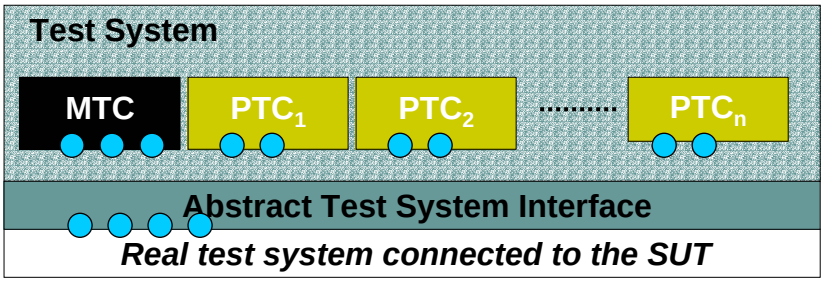


Concepts – Test configuration 2(2)



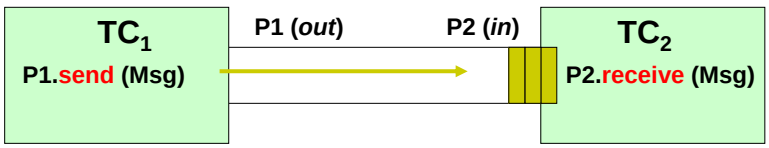
Concepts – Test components

- There are three ‘kinds’ of components
  - Abstract Test System Interface defined as component
  - MTC (Main Test Component)
  - PTC (Parallel Test Component)



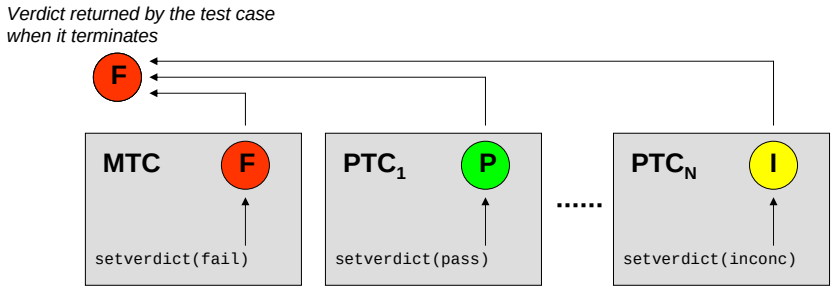
Concepts – Communication ports

- Test components communicate via ports
- A test port is modeled as an infinite FIFO queue
- Ports have direction (in, out, inout)
- There are three types of port
  - message-based, procedure-based or mixed

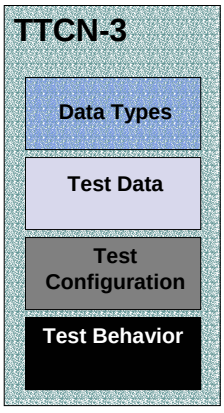


Concepts – Test verdicts

- Test verdicts: none < pass < inconc < fail < error
- Each test component has its own local verdict, which can be set (setverdict) and read (getverdict).
- A test case returns a global verdict



Concepts – Main elements of TTCN-3

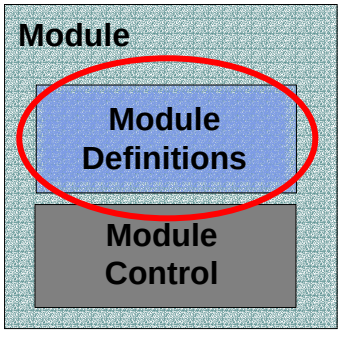


- Built-in and user-defined generic data types (e.g., to define messages, service primitives, information elements, PDUs).
- Actual test data transmitted/received during testing.
- Definition of the components and communication ports that are used to build various testing configurations.
- Specification of the dynamic test system behavior.

TTCN-3 language elements

- Module
- Module Definitions
  - Constants, Types, Templates
  - Port and Component Type Definitions
  - Functions
  - Altsteps
  - Test Cases
- Module Control

TTCN-3 language elements – Module 1(2)



- Modules are the building blocks of all TTCN-3 test specifications.
- A test suite is a module.
- A module has a definitions part and an (optional) control part.
- Modules can be parameterized.
- Modules can import definitions from other modules.

TTCN-3 language elements – Module 2(2)

```
module Example {
  modulepar {
    integer Par_One, Par_Two;
    boolean Par_Three := true
  }
  import from AnotherModule {
    …
  }
  … // all definitions

control {
  … // execution of test cases
}
}
```

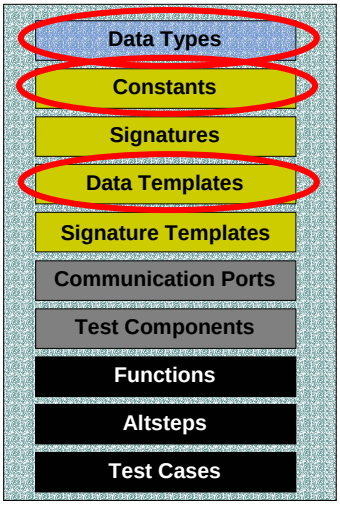
Module parameter definitions with and without default value

Import statement (more details later)

Definitions part

Control part

TTCN-3 language elements – Module definitions

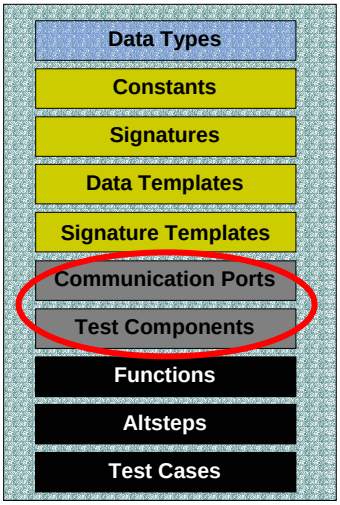


- Definitions are global to the entire module.
- Data Type definitions are based on TTCN-3 predefined and structured types.
- Templates define the test data.
- Ports and Components are used in Test Configurations.
- Functions, Altsteps and Test Cases define behavior.

TTCN-3 language elements – Constants, types, templates

```
const integer int_Const := 7;        // Normal constant
external const boolean bool_Const;   // External constant

type record Request   { // Structured type definition
  RequestLine      requestLine,
  ReqMessageHeader reqMessageHeader optional,
  charstring       crlf,
  charstring       messageBody optional
}

template Request Invite := { // template for the Request type
  requestLine      := Request_Line("INVITE"),
  reqMessageHeader := Req_Mes_Header("INVITE"),
  crlf             := CRLF,
  messageBody      := omit
}
```

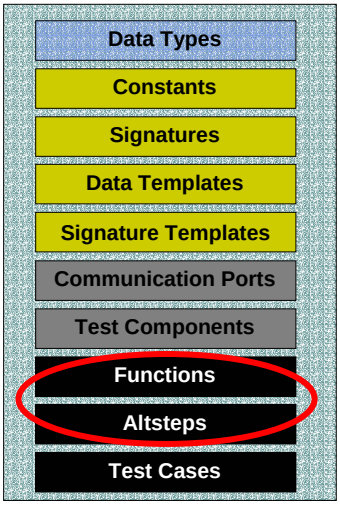

TTCN-3 language elements – Module definitions (recall)



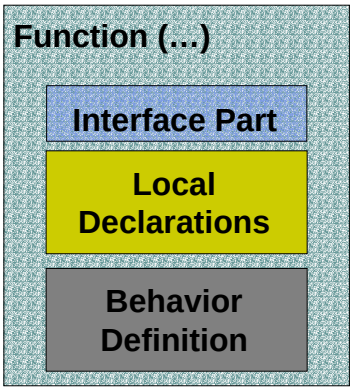
TTCN-3 language overview – Port & component type definitions

```
type port SipPortType message {
    inout Request, Response;
}

type component SipTestComponent {
    var integer Counter := 0;
    timer T1 := 0.5;
    timer T2 := 4.0;
    port SipPortType SIP_PCO
}
```


TTCN-3 language elements – Module definitions (recall)

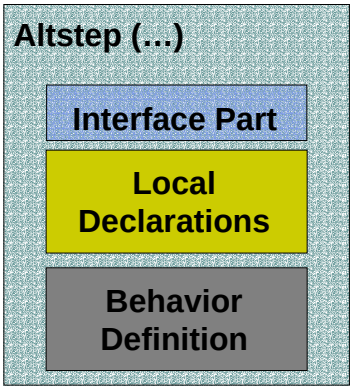


TTCN-3 language elements – Functions



- … allow to structure the test system behavior.
- … have
  - a (optional) interface part,
  - a declarations part and
  - a behavior definition part.
- … can be a 'pure' functions doing some data calculation or specify test behavior using communication operations such as send and receive.
- … can be
  - user-defined,
  - external or
  - pre-defined.

TTCN-3 language elements – Altsteps

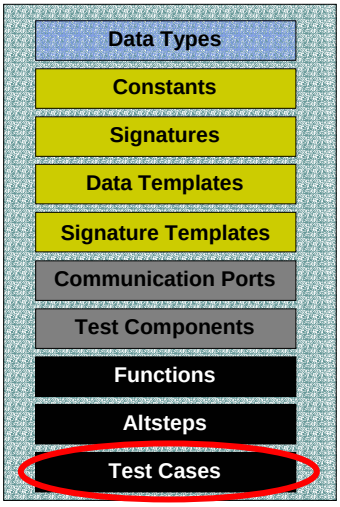


- … are a special kind of function and have therefore the same structure as a normal function.
- … allow to structure alternative behavior.
- … can be activated as default behavior.

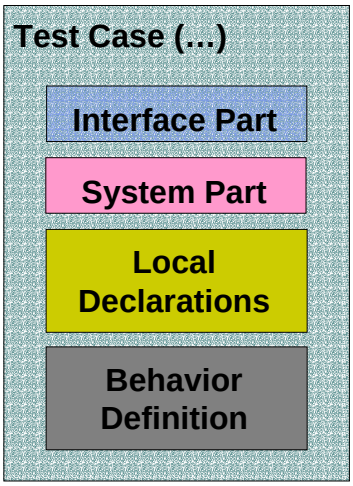
TTCN-3 language elements – Functions and altsteps

```
function postamble(charstring cseq) runs on SipTestComponent
{
    SIP_PCO.send(Bye_s_1(cseq));
}


altstep Default_1(charstring cseq) runs on SIPTestComponent {
  [] any timer.timeout {
       … // Behavior for unexpected timeout events
     }
  [] any port.receive {
       … // Behaviour for unexpected message arrivals
     }
}
```


TTCN-3 language elements – Module definitions (recall)



TTCN-3 language elements – Test cases 1(2)

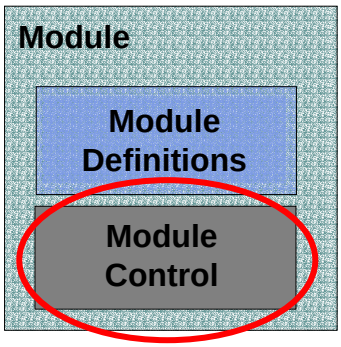


- Test cases are a special kind of functions, which are executed in the control part of a module.
- The interface part references the MTC on which the test case will run.
- The system part refers to the test system interface component. It is optional and can be omitted if the test case only consists of an MTC
- The behavior definitions specifies the behavior of the MTC

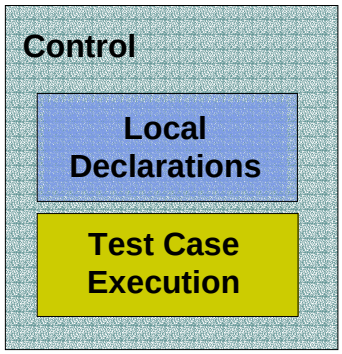
TTCN-3 language elements – Test cases 2(2)

```
testcase SIP_UA_REC_V_001() runs on SipTestComponent
  system configuration_01 {
    activate(Default_1("0"));
    map(self:SIP_PCO, system:SIP_PCO);
    SIP_PCO.send(Invite_s_1);
    T1.start;
    SIP_PCO.receive(Response_r_1);
    setverdict(pass);
    T1.stop;
    postamble("0");
    stop;
}
```

TTCN-3 language elements – Module (recall)



TTCN-3 language elements – Module control part 1(2)



- Module control is the 'dynamic' part of a TTCN-3 specification where the test cases are executed.
- Local declarations, such as variables and timers may be made in the control part.
- Basic programming statements may be used to select and control the execution of the test cases.

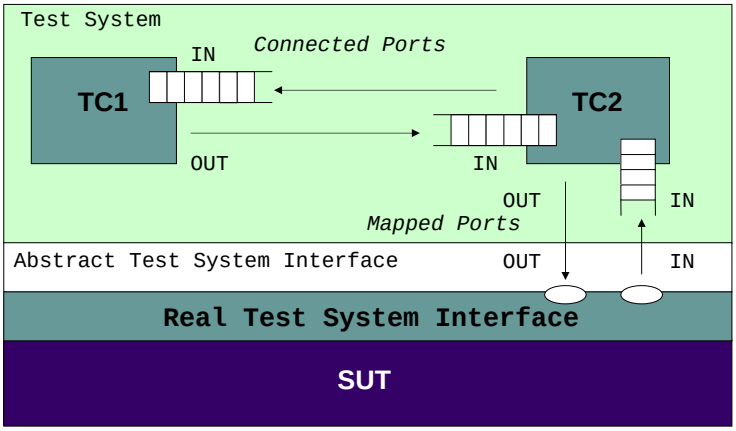
TTCN-3 language elements – Module control 2(2)

```
module … {
…
  control{
    var integer count;
    if(execute(SIP_UA_REC_V_001()) == pass) {
      // Execute test case 10 times
      count := 0;
      while( count <= 10) {
        execute(SIP_UA_REC_V_002());
        count := count + 1;
      } // end while
    } // end if
  } // end control
} // end module
```


TTCN-3 Test Behavior Specification

- Configuration Operations
- Procedure-based Communication
- Alternative Behavior
- Default Handling
- Overview

TTCN-3 test behavior spec. – Configuration operations 1(5)



TTCN-3 test behavior spec. – Configuration operations 2(5)

- Component Handling
  - Create operation

```
var MyCompType MyNewComp;
MyNewComp := MyCompType.create;
```

  - Start operation

```
MyNewComp.start(MyCompBehavior(…));
```

  - Stop operation

```
if (date == “1.1.2000”) { stop; }
```

TTCN-3 test behavior spec. – Configuration operations 3(5)

- Component Handling (continued)
  - Running operation

```
if (MyNewComp.running) {
        : // Do something
}
```

  - Done operation

```
all component.done;
```


TTCN-3 test behavior spec. – Configuration operations 4(5)

- Port Handling
  - Connect and Disconnect operations

```
connect(MyNewComp:Port1, mtc:Port3);
disconnect(MyNewComp:Port1, mtc:Port3);
```

  - Map and Unmap operation

```
map(self:Port2, system:PCO1);
unmap(self:Port2, system:PCO1);
```

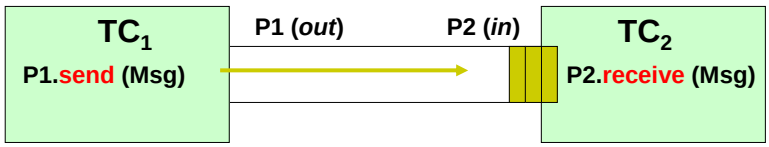
TTCN-3 test behavior spec. – Configuration operations 5(5)

- Port Handling (continued)
  - Start, Stop and Clear operations

```
MyPort.start;
MyPort.stop;
MyPort.clear;
```


TTCN-3 test behavior specification – Procedure-based communication 1(5)

- The details of message-based communication will be explained in the example (Part III).

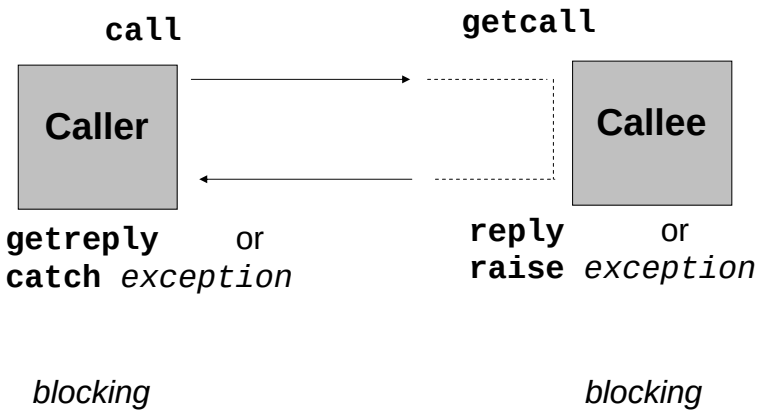


- Therefore, only procedure-based communication will be explained here.

TTCN-3 test behavior specification – Procedure-based communication 2(5)

```
signature MyRemoteProc
  (in integer Par1,
    out float Par2,
      inout integer Par3)
   return integer
   exception (ExceptType1, ExceptType2);

template MyRemoteProc Mytemplate := {
   Par1 := 7,
   Par2 := *,
   Par3 := MyConst
}
```


TTCN-3 test behavior specification – Procedure-based communication 3(5)



TTCN-3 test behavior specification – Procedure-based communication 4(5)

```
MyCL.call(MyProcTemp(5,MyVar), 0.03) to MyPartner {
   [] MyCl.getreply(MyProc:{MyVar1,MyVar2}) ->
        value MyResult param (MyPar1Var,MyPar2Var){}

   [] MyCL.catch(MyProc, MyExceptionOne) {
        stop;
      }
   [] MyCL.catch(MyProc, MyExceptionTwo) {}

   [] MyCL.catch(timeout) {
        setverdict(fail);
        stop;
      }
}
```


TTCN-3 test behavior specification – Procedure-based communication 5(5)

```
MyCL.getcall(MyProcTemp(5, MyVar)) -> sender MyPeer;
```

```
MyCL.reply(MyProc:{20, MyVar2} value 20) to MyPeer;
```

```
MyCL.raise(MyProc, MyVar + YourVar - 2) to MyPeer;
```

TTCN-3 test behavior spec. – Alternative behavior 1(2)

- … has to be specified whenever test component is ready to take a response from the SUT or a timeout.
- … is typically defined by several alternatives, which
  - are evaluated according to their appearance
  - may be guarded
  - can be part of an altstep, which may be called explicitly or activated as default.
- … forks the test behavior (the typical „tree“), but in TTCN-3 alternatives can be joined again after the end of an alternative behavior.
- All other cases can be handled in an else branch.

TTCN-3 test behavior spec. – Alternative behavior 2(2)

```
alt {
  [] L1.receive(MyMessage1) {
       :              // do something
     }
  [x>1]  L2.receive(MyMessage2) {}
  [x<=1] L2.receive(MyMessage3) {}
  [] MyTeststep;      // call of a teststep
  [else] stop         // else branch
}
```

TTCN-3 test behavior spec. – Default handling

- Default handle

```
var default MyDefault := null;
```

- Activation of an altstep as default

```
MyDefault := activate (MyAltstep());
```

- Default deactivation

```
MyDefault := activate (MyAltstep());
```


TTCN-3 test behavior spec. – Overview 1(6)

Statement	Keyword or symbol	Module control part	Functions and test cases
Basic program statements			
Expressions	**(…)**	Yes	Yes
Assignments	**:=**	Yes	Yes
Logging	**log**	Yes	Yes
Label and Goto	**label / goto**	Yes	Yes
If-else	**if (…) {…} else {…}**	Yes	Yes
For loop	**for (…) {…}**	Yes	Yes
While loop	**while (…) {…}**	Yes	Yes
Do while loop	**do {…} while {…}**	Yes	Yes
Stop execution	**stop**	Yes	Yes

TTCN-3 test behavior spec. – Overview 2(6)

Statement	Keyword or symbol	Module control part	Functions and test cases
Behavior statements and operations			
Alternative behavior	`alt {…}`	(Yes)	Yes
Repeat alternative	`repeat`	(Yes)	Yes
Interleaved behavior	`Interleave {…}`	(Yes)	Yes
Returning Control	`return`		Yes
Statements for default handling			
Activate a default	`activate`	(Yes)	Yes
Deactivate a default	`deactivate`	(Yes)	Yes

TTCN-3 test behavior spec. – Overview 3(6)

Statement	Keyword or symbol	Module control part	Functions and test cases
Configuration operations			
Create	`create`		Yes
Connect ports	`connect`		Yes
Disconnect ports	`disconnect`		Yes
Map ports	`map`		Yes
Unmap ports	`unmap`		Yes
Get MTC id	`mtc`		Yes
Get system id	`system`		Yes
Get own id	`self`		Yes
Start component	`start`		Yes
Stop component	`Stop`		Yes
Running comp.	`running`		Yes
Component done	`done`		Yes

TTCN-3 test behavior spec. – Overview 4(6)

Statement	Keyword or symbol	Module control part	Functions and test cases
Communication operations			
Send message	**`send`**		Yes
Call procedure	**`call`**		Yes
Reply to proc.	**`reply`**		Yes
Raise exception	**`raise`**		Yes
Receive message	**`receive`**		Yes
Trigger on mess.	**`trigger`**		Yes
Accept proc. call	**`getcall`**		Yes
Proc. reponse	**`getreply`**		Yes
Catch exception	**`catch`**		Yes
Check port	**`check`**		Yes
Clear port	**`clear`**		Yes
Start port	**`start`**		Yes
Stop port	**`stop`**		Yes

TTCN-3 test behavior spec. – Overview 5(6)

Statement	Keyword or symbol	Module control part	Functions and test cases
Timer operations			
Start timer	**`start`**	Yes	Yes
Stop timer	**`stop`**	Yes	Yes
Read elapsed time	**`read`**	Yes	Yes
Timeout event	**`timeout`**	Yes	Yes
Status check	**`running`**	Yes	Yes
Verdict operations			
Set local verdict	**`setverdict`**		Yes
Get local verdict	**`getverdict`**		Yes

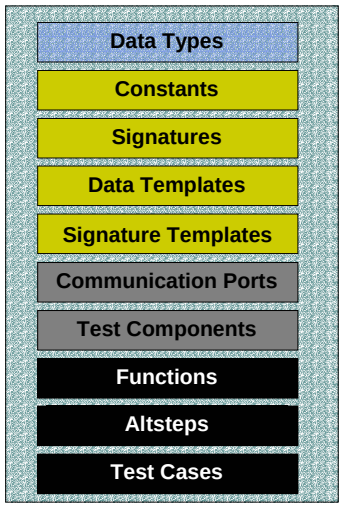
TTCN-3 test behavior spec. – Overview 6(6)

Statement	Keyword or symbol	Module control part	Functions and test cases
External actions			
Stimulate external action	`action`	Yes	Yes
Execution of test cases			
Execute test case	`execute`	(Yes)	Yes

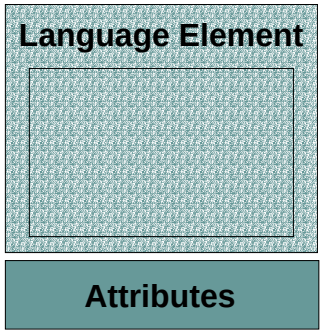
Attributes, groups, import – Overview

- Attributes
- Grouping
- Importing from other Modules
- Importing non TTCN-3 Definitions

Attributes, groups, import – Module Definitions (recall)



Attributes, groups, import – Attributes 1(2)



- Attributes can be assigned to all kinds of definitions, groups and modules.
- Kinds of attributes
  - Encoding information
    - encode – attribute
    - variant – attribute
    - Values are standardized for ASN.1 encoding
  - Presentation information
    - display – attribute
    - Values are standardized for the graphical and tabular presentation format.
  - User-defined
    - extension - attribute

Attributes, groups, import – Attributes 2(2)

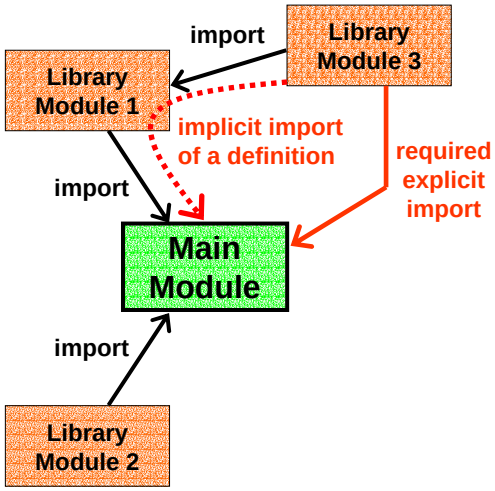
```
group MyPDUs {
  type record MyPDU1 { … }
  group MySpecialPDUs {
    type record MyPDU3 { … }
      with {extension "MyVerySpecialRule"}
    type record MyPDU4 { … }
  }
  with {extension "MySpecialRule"}
}
with {
  display "PDU";
  extension "MyRule"
}
```


Attributes, groups, import – Grouping

```
group Logical_Group {
  import from …
  modulepar …
  const …
  type …
  function …
  altstep …
  testcase …
  …
  group Sub_Logical_Group {
    import from …
    …
  }
}
```

- TTCN-3 grouping mechanism allows to structure the module definitions part logically.
- Groups may also be structured into groups.
- Groups are no scope units.
- Groups can be used for the assignment of attributes to all definitions of the group.
- Groups of definitions can be imported.

Attributes, groups, import – Import from other modules 1(2)



- Main Module contains the control part, which specifies test suite execution.
- Modules may reuse definitions from other (library) modules.
- Implicit import of definitions via chains of imports is not allowed, i.e., an explicit import has to be added.
  - Reason: A module shall know all modules which it depends on.

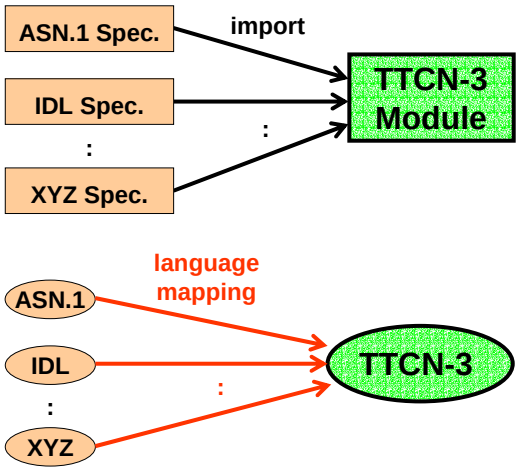
Attributes, groups, import – Import from other modules 2(2)

```
import form ModuleOne {
  modulepar ModPar2;
  type RecordType_T2
}

import from ModuleTwo
recursive {
  testcase T_case
}

import from ModuleThree
all except {
  template all
}
```

- Import allows to import
  - single definitions
  - definitions of a certain kind, and
  - groups of definitions

  from other modules.
- Definitions may be imported recursively.
- If several definitions are addressed, certain definitions can be excluded by using an except directive.

Attributes, groups, import – Importing non TTCN-3 definitions



- Importing non TTCN-3 definitions requires a language mapping onto TTCN-3.
- The language mapping defines the meaning of non TTCN-3 definitions in TTCN-3 modules.
- The language mapping may provide TTCN-3 additional features for imported definitions (e.g., operations for ASN.1 data types).

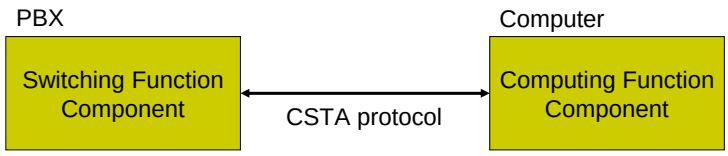
PART II: The test application

The CSTA example

Test purposes

The CSTA example 1(4)

- CSTA = Services for Computer-Supported Telecommunications Applications
  - Defines OSI layer 7 communication between
    - Computing network (a PC in the simplest case) and
    - Telecommunication network (a PBX)

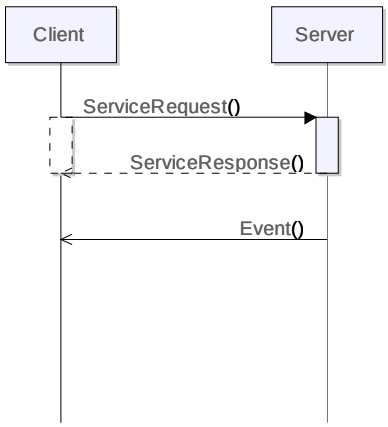


The CSTA example 2(4)

- CSTA standard
  - Standardized by Ecma International
    - European association for standardizing information and communication systems
    - http://www.ecma-international.org/
  - Developed in versions (= phases)
  - Current standard: CSTA phase III
    - Services: Ecma-269 (Jun. 2000)
    - ASN.1-based protocol: Ecma-285 (Jun. 2000)
    - XML-based protocol: Ecma-323 (Dec. 2002)
    - WSDL-based protocol: Ecma-348 (Jun. 2003)
- **Here**: CSTA III XML

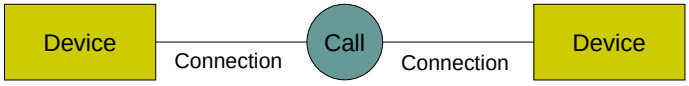
The CSTA example 3(4)

- CSTA communication
  - Message-based communication
    - XML coded
  - Client/server relationship
    - Service request
    - Service response
  - Events
  - Roles for client and server are interchange-able



The CSTA example 4(4)

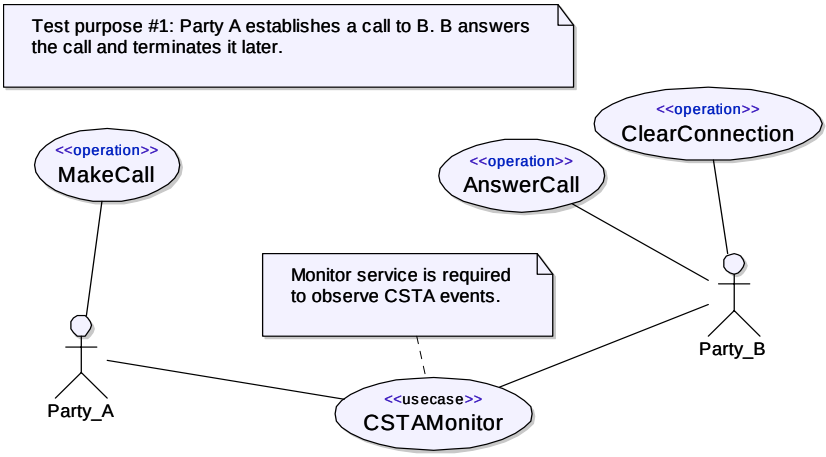
- CSTA device
  - Allows users to access telecommunication services
  - Either physical (stations) or logical ones (e.g. call groups)
- CSTA call
  - Is a communication relationship between one or more devices
- CSTA connection
  - Is a relationship between a CSTA device and a call, in which the device is involved



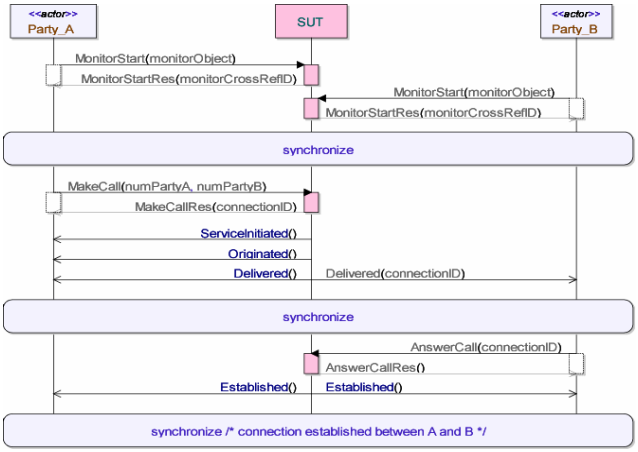
Test purposes – Overview 1(7)

- CSTA service testing of switching function operations
- A telephony application shall apply some switching function services
  - Test purpose #1: Basic phone call between 2 parties
  - Test purpose #2: Conference call among 3 parties

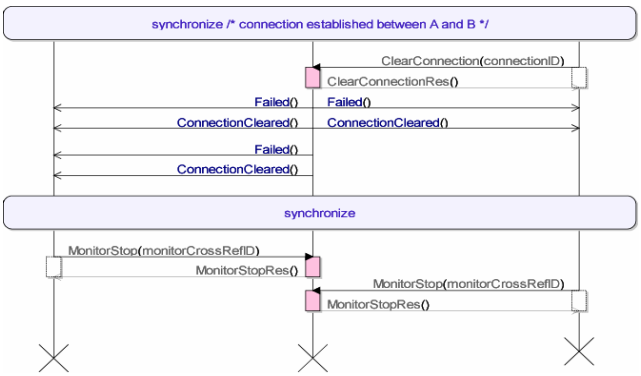
Test purposes – Use case #1 2(7)



Test purposes – Sequence chart #1a 3(7)



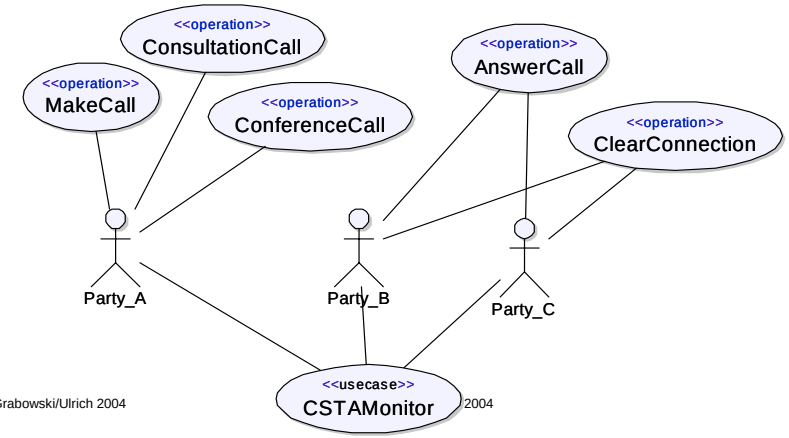
Test purposes – Sequence chart #1b 3(7)



Test purposes – Use case #2

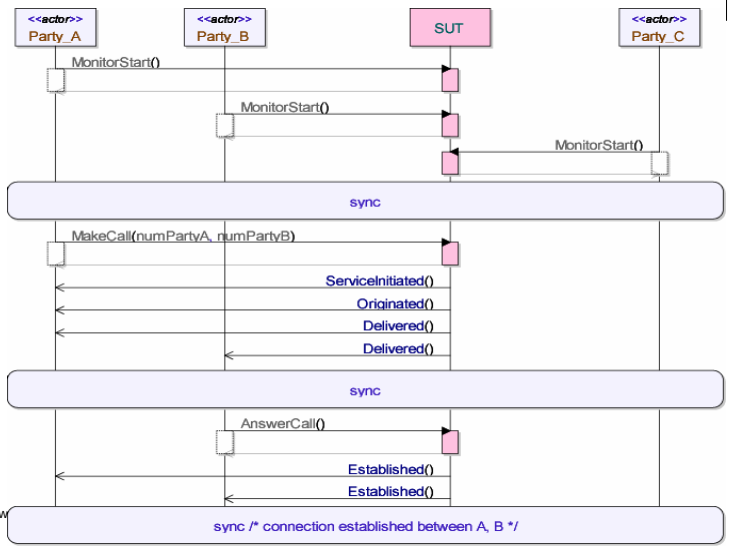
4(7)

Test purpose #2: Party A calls B. Then A initiates a consultation call to C and joins both parties, C and B, in a conference call.

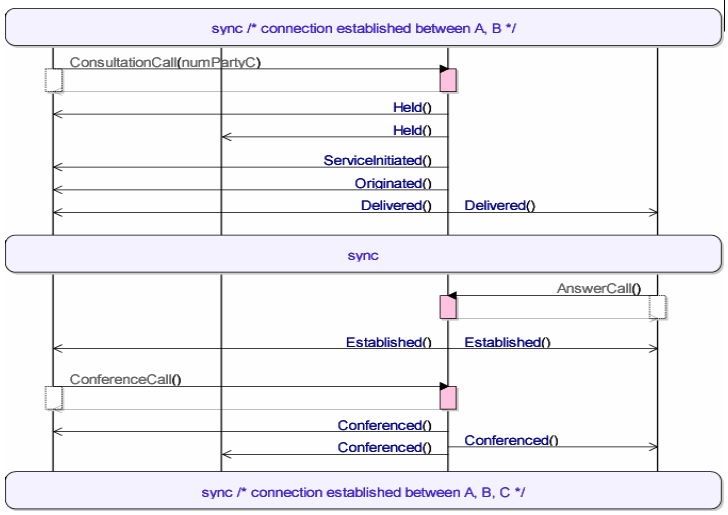


Test purposes – Sequence chart #2a

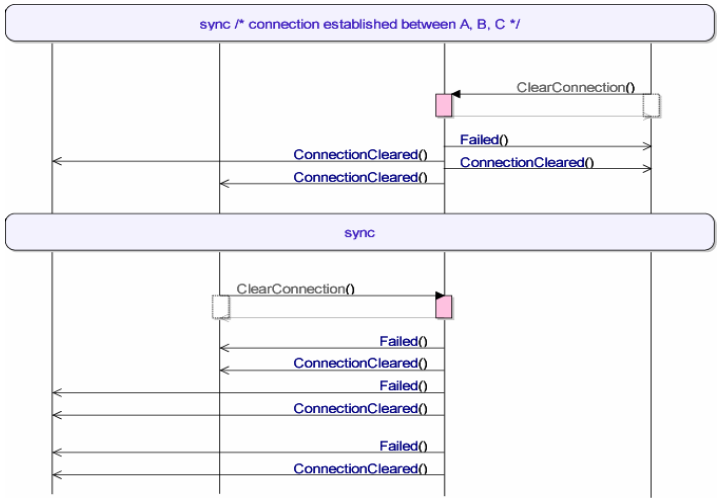
5(7)



Test purposes – Sequence chart #2b 6(7)



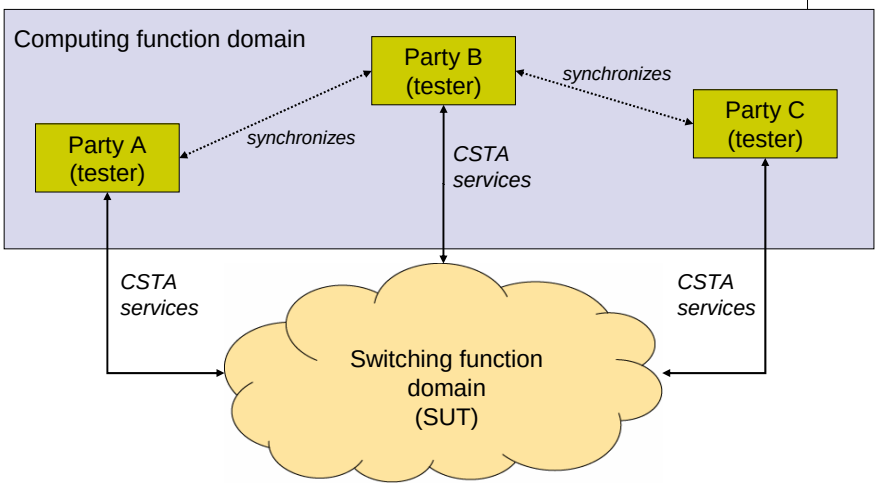
Test purposes – Sequence chart #2c 7(7)



PART III: TTCN-3 en detail

Test architecture
Test data definitions
Test behavior description
Overall view of the test suite

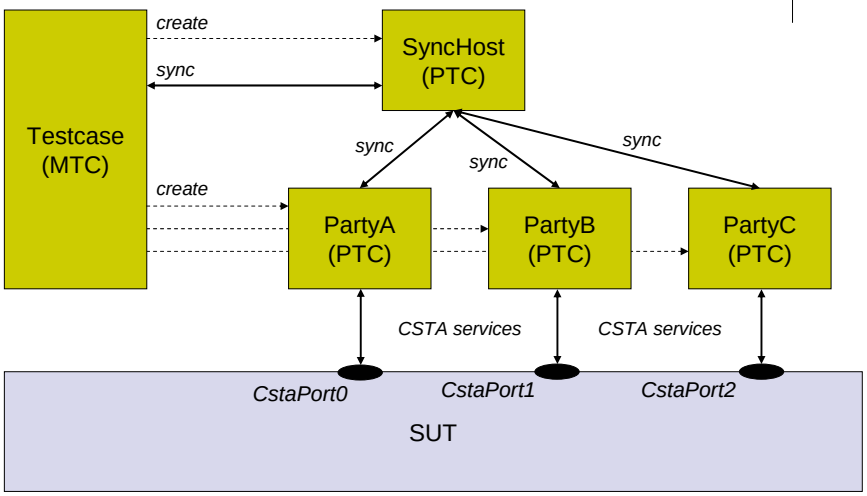
Test architecture – Service testing 1(4)



Test architecture – TTCN-3 design 2(4)

- The SUT comprises all components (PBXs) that make up the Switching function domain
  - Is the **system** component in TTCN-3
- A Tester reflects a party in the Computing function domain
  - Implemented as a PTC in TTCN-3
  - Each tester manages its own CSTA message port
- Synchronization among testers
  - Implemented by means of an additional PTC “SyncHost” and applying the publisher/subscriber design pattern
- Creation of PTCs and mapping/connecting ports is done exclusively in the MTC (**testcase**)

Test architecture – TTCN-3 design 3(4)



Test architecture – TTCN-3 design approach 4(4)

- Define the following parts of a TTCN-3 test suite
  1. Ports and components SUT, MTC and (optional) PTCs
  2. In/out messages and procedure calls exchanged between tester components and SUT
  3. Tester-internal messages
     - E.g. synchronization messages
  4. Data templates of in/out messages
  5. Behavior definitions
     - Test cases
     - Functions and altsteps that run on components
     - Functions that manipulate data (without “runs on” attribute)
     - Consider definition of proper module parameters
  6. Control part to execute the defined test cases
- Distribute all definitions of your TTCN-3 project over a proper set of TTCN-3 modules

Test data definitions – CSTA messages 1(7)

- CSTA message types are given in XML schema definitions (XSD)
- Mapping required from XSD to TTCN-3 data types
  - Currently no standardized mapping rules defined
  - Project-specific, ad-hoc mappings prevail
- Message == record/union
  - Record may contain optional elements

```
type record MakeCall {
   DeviceID callingDevice,
   DeviceID calledDirectoryNumber,
   …
   CSTACommonArguments extensions
                           optional
};
type integer DeviceID;
type record CSTACommonArguments {
   CSTASecurityData security
                           optional,
   CSTAPrivateData privateData
                           optional
};
type union CSTAPrivateData {
   octetstring string,
   bitstring private
};
```

Test data definitions – Party component 2(7)

- A Party is defined as a component (PTC)
  - CSTA port “cstaPort”
    - Accepts all in/out messages
  - Sync port
  - Timer “cstaTimer”
    - Timeout value set to “CstaTimeout” (module parameter)
  - Variable “syncHost”
    - Stores reference to the SyncHost component
    - Initialized to “null”

```
type port CstaPortType message {
   inout all;
}
type port SyncPortType message {
   inout SyncMsg;
}

type component PartyType {
   port CstaPortType cstaPort;
   port SyncPortType syncPort;
   timer cstaTimer := CstaTimeout;
   var SyncHostType syncHost :=
                                 null;
}
```


Test data definitions – SUT and MTC definitions 3(7)

- The SUT is defined as another TTCN-3 component
  - Provides 3 separate CSTA ports for communication with parties
  - Implementation of these ports is done in TRI test adaptor
- The MTC is defined as component “MTC”
  - Provides a sync port to access and control the SyncHost component

```
// ports are already defined
type port CstaPortType message {
   inout all;
}
type port SyncPortType message {
   inout SyncMsg;
}

type component SUT {
   port CstaPortType cstaPort0;
   port CstaPortType cstaPort1;
   port CstaPortType cstaPort2;
}

type component MTC {
   port SyncPortType syncPort;
}
```

Test data definitions – Templates 4(7)

- Templates define values and/or placeholders for each message field
- Out-message template
  - May contain parameters
  - Optional fields may be omitted (“omit”)
- In-message template
  - May contain parameters
  - May use placeholders
    - “?”: element **must** occur
    - “*”: element **may** occur in the received message

```
template MakeCall makeCallDefault
( in DeviceID sender_,
  in DeviceID receiver_) :=
{
  callingDevice := sender_,
  calledDirectoryNumber :=
                          receiver_,
  …
  extensions := omit
}
template MakeCallResponse
         makeCallResponseDefault :=
{
  callingDevice := ?,
  mediaCallCharacteristics := *,
  initiatedCallInfo := *,
  extensions := *
}
```


Test data definitions – ASN.1 in TTCN-3 5(7)

```
ACSE-1 DEFINITIONS ::= BEGIN

ACSE-apdu ::= CHOICE
{
  aarq  AARQ-apdu,
  …
}
AARQ-apdu ::= SEQUENCE
{
  protocol-version  BIT STRING,
  application-context-name
                    OBJECT IDENTIFIER,
  called-AP-title   AP-title OPTIONAL,
  …
  calling-authentication-value
            Authentication-value OPTIONAL,

  …
  user-information
        Association-information OPTIONAL
}

END
```

```
import from ACSE_1 language "ASN.1:1997"
all with { encode "BER:1997" };


// corresponding TTCN-3 template
template ACSE_apdu acseAssociateRequest
  (charstring authenticationValue) :=
{
  aarq := {
    protocol_version := '0'B,
    application_context_name :=
                    objid {1 3 12 0 218},
    called_AP_title := omit,
    …
    calling_authentication_value := {
      charstr := authenticationValue
    },
    …
    user_information := {
      …
    }
  }
}
```

Test data definitions – Module parameters 6(7)

- Module parameters are similar to constants
- Used here to describe test suite parameters
  - Timeout values
  - Characterization of a party
    - Name of the party
    - Phone number
    - Authentication string
- Can be imported like other definitions
- Tool providers may provide means for parameterization during test runtime

```
modulepar {
  // describes the test purpose
  charstring TestPurpose := "...";
  // timeout of the entire test case
  float TestCaseTimeout := 360.0;
  // CSTA connection timeout
  float CstaTimeout := 5.0;

  // total number of parties, 1..5
  integer TotalPartyNumber := 3;

  // party A
  charstring PartyNameA := "partyA";
  charstring PartyAuthA := "...";
  integer PartyNumberA := 1111;

  …
}
```


Test data definitions – Import statements 7(7)

- Introduces definition identifiers from other modules into the current module
  - All definitions of a module
  - Explicit identifiers
- Often extended by encoding information
  - Predefined: “BER:1997”
  - User-defined: “Ecma323”

```
module cstaInitialization
{
  import from BaseDefinitions {
     type  PartyType;
     type  PeerSystemStatus;
     function convertSystemStatus;
  };

  import from ECMA323 {
     type  SystemStatus,
           SystemStatusResponse,
           SystemStatusVal;
  } with { encode "Ecma323"; };

  import from ACSE_1
    language "ASN.1:1997" all
    with { encode "BER:1997" };
}
```

Test behavior description – Behavior functions (a) 1(9)

- Behavior functions run on a component (“runs on”)
  - Invoked at a component’s start or anytime during execution
  - May carry in/out/inout parameters
  - May define local variables at the beginning
- xmlMakeCall()
  - Requests the MakeCall service using template “makeCallDefault”
  - Waits for the service response using template “makeCallResponseDefault”
  - Returns the connection ID of the new call contained in the response message

```
function xmlMakeCall
( in DeviceID from_, in DeviceID to_,
  out ConnectionID connId )
runs on PartyType
{
  var MakeCallResponse mcrMsg;
  if(getverdict != fail) {
    cstaPort.send(makeCallDefault(
                          from_, to_));
    cstaTimer.start;
    alt {
    [] cstaPort.receive(
                makeCallResponseDefault)
       -> value mcrMsg {
         cstaTimer.stop;
         connId :=  mcrMsg.callingDevice;
       }
    }/*tla*/
  }/*fi*/
}
```


Test behavior description – Behavior functions (b) 2(9)

- xmlCallClearedEvent()
  - Waits for the occurrence of the CallCleared event that matches template “callClearedEventDefault”
  - If it does not occur before “cstaTimer” times out, a default altstep will be activated then (outside the function definition)
  - Returns the connection ID contained in the event

```
template CallClearedEvent
  callClearedEventDefault ( … ) := { … }

function xmlCallClearedEvent
( in MonitorCrossRefID monitorId,
  out ConnectionID connId )
runs on PartyType
{
  var CallClearedEvent event;

  if(getverdict != fail) {
    cstaTimer.start;
    alt {
    [] cstaPort.receive(
               callClearedEventDefault(…)
                    ) -> value event {
         cstaTimer.stop;
         connId := event.clearedCall;
       }
    }
  }
}
```

Test behavior description – Behavior functions (c) 3(9)

- scriptA()
  - The overall function that implements the behavior of party A
  - Activates/deactivates default altsteps at the beginning and the end
  - If a message received in xmlMakeCall() does not match within this function, matching is attempted first in "cstaEventCollector", then in "cstaDefaultHandler" (reverse activation order!)

```
function scriptA() runs on PartyType
{
  var default dh := activate(
              cstaDefaultHandler());
  var default cstaEventCollector :=
                          activate(
          xmlDefaultEventHandler());
  …

  xmlMakeCall(
         PartyNumberA, PartyNumberB,
         connId);

  …

  deactivate(cstaEventCollector);
  deactivate(dh);
}
```


Test behavior description – Altsteps 4(9)

- Altsteps
  - Define a set of alternative behavior if a receive operation in an associated alt statement fails
- cstaDefaultHandler()
  - If activated, it matches any CSTA error message, any unidentified message and a timeout of the CSTA timer
  - "stop" statement stops execution of the component, on which the altstep was activated
  - Otherwise execution continues after associated alt statement

```
altstep cstaDefaultHandler()
runs on PartyType
{
[] cstaPort.receive(CSTAErrorCode:?)
    {
      cstaTimer.stop;
      setverdict(fail);
      stop;
    }
[] cstaPort.receive {
      …
    }
[] cstaTimer.timeout {
      log("Timeout received!");
      setverdict(inconc);
      // continue test execution
}   }
```

Test behavior description – Testcases 5(9)

- A testcase is the initial function that must always exist in an executable TTCN-3 module
- cstaTestCase()
  - Static test configuration
  - Is defined on type “MTC” and uses the ports defined in “SUT”
  - Creates all test components of the test suite
  - Connects/maps to internal/external ports
  - Starts all components and waits for their termination
  - Disconnects/unmaps all ports

```
testcase cstaTestCase()
runs on MTC system SUT
{
  var SyncHostType sh :=
                      SyncHostType.create;
  var PartyType partyA :=
                       PartyType.create;
  connect(self:syncPort, sh:syncPort);
  connect(partyA:syncPort, sh:syncPort);
  map(partyA:cstaPort, system:cstaPort0);

  sh.start(syncHostMain());
  partyA.start(testScriptInit(1, sh));
  partyA.done;

  unmap(partyA:cstaPort,
                     system:cstaPort0);
  disconnect(partyA:syncPort,
                           sh:syncPort);
  disconnect(self:syncPort, sh:syncPort);
}
```


Test behavior description – Local concurrency 6(9)

- Concurrent messages received at the same port of a component must be handled properly
- Example from test purpose #2 of party A
  - After the ConsultationCall service the Held event occurs concurrently with the sequence of ServiceInitiated, Originated, Delivered events

```
// behavior of party A from #2
// after ConsultationCall
...
interleave {
[] cstaPort.receive(HeldEvent:?)
   { ... }
[] cstaPort.receive(
            ServiceInitiatedEvent:?)
   {
      cstaPort.receive(
                  OriginatedEvent:?)
      { ... }
      cstaPort.receive(
                   DeliveredEvent:?)
      { ... }
   }
}
...
```

Test behavior description – Implementing the TPs 7(9)

- Test purposes (TPs) given as sequence charts are implemented for each party separately as “scriptA()”, “scriptB()” etc.
  - Resembles a projection of actions in a sequence chart on the selected party
  - Relies on an existent synchronization mechanism between parties; here: function “sync()”
  - Projection could be done automatically
    ➔ Test script generation

Test behavior description – Implementing TP #1 8(9)

```
function scriptA() runs on PartyType {
  var MonitorCrossRefID monitorId;
  var ConnectionID connId;

  xmlMonitorStart(PartyNumberA, monitorId);
  sync(); // null state
  xmlMakeCall(PartyNumberA, PartyNumberB, connId);
  xmlServiceInitiatedEvent(monitorId, connId);
  xmlOriginatedEvent(monitorId, connId);
  xmlDeliveredEvent(monitorId, connId);
  sync(); // connected state

  xmlEstablishedEvent(monitorId, connId);
  sync(); // connected state
  log("connection established for A");


  xmlFailedEvent(monitorId, connId);
  xmlConnectionClearedEvent(monitorId, connId);
  xmlFailedEvent(monitorId, connId);
  xmlConnectionClearedEvent(monitorId, connId);
  sync(); // null state
  xmlMonitorStop(monitorId);
}
```

```
function scriptB() runs on PartyType {
  var MonitorCrossRefID monitorId;
  var ConnectionID connId;

  xmlMonitorStart(PartyNumberB, monitorId);
  sync(); // null state
  xmlDeliveredEvent(monitorId, connId);



  sync(); // alerting state
  xmlAnswerCall(connId);
  xmlEstablishedEvent(monitorId, connId);
  sync(); // connected state
  log("connection established for B");
  sleep(2.0);
  xmlClearConnection(connId);
  xmlFailedEvent(monitorId, connId);
  xmlConnectionClearedEvent(monitorId, connId);


  sync(); // null state
  xmlMonitorStop(monitorId);
}
```

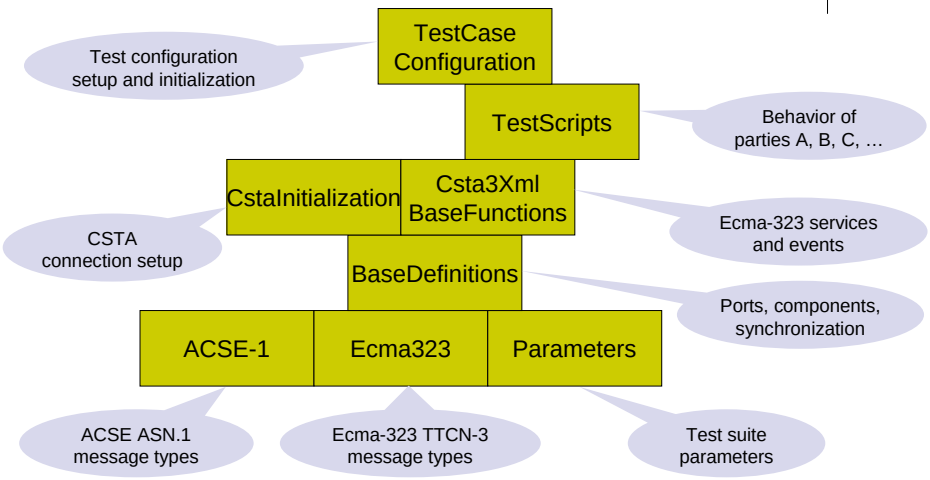
Test behavior description – Implementing TP #2 9(9)

```
function scriptA()
runs on PartyType {
  xmlMonitorStart();
  sync(); // null state
  xmlMakeCall();
  xmlServiceInitiatedEvent();
  xmlOriginatedEvent();
  xmlDeliveredEvent();
  sync(); // connected state

  xmlEstablishedEvent();
  sync(); // connected state
  log("connection A with B");
  xmlConsultationCall();
  xmlHeldEvent();
  xmlServiceInitiatedEvent();
  xmlOriginatedEvent();
  xmlDeliveredEvent();
  sync(); // connected state

  xmlEstablishedEvent();
  log("connection A with C");
  xmlConferenceCall();
  xmlConferencedEvent();
  sync(); // connected state
  log("Connection for A, B, C");
  …
}
```

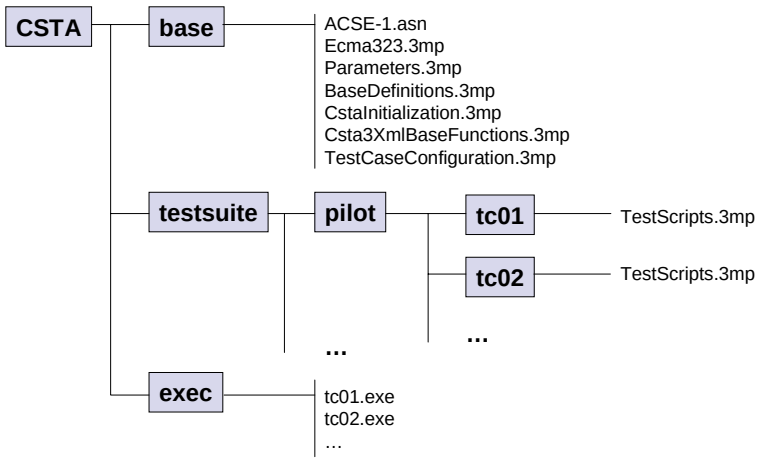
```
function scriptB()
runs on PartyType {
  xmlMonitorStart();
  sync(); // null state



  xmlDeliveredEvent();
  sync(); // alerting state
  xmlAnswerCall();
  xmlEstablishedEvent();
  sync(); // connected state
  log("connection B with A");

  xmlHeldEvent();



  sync(); // hold state



  xmlConferencedEvent();
  sync(); // connected state
  log("Conference for B, A, C");
  …
}
```

```
function scriptC()
runs on PartyType {
  xmlMonitorStart();
  sync(); // null state



  sync(); // null state


  sync(); // null state



  xmlDeliveredEvent();
  sync(); // alerting state
  xmlAnswerCall();
  xmlEstablishedEvent();
  log("connection C with A");

  xmlConferencedEvent();
  sync(); // connected state
  log("Conference for C, A, B");
  …
}
```


Overall view of the test suite – Test suite modules 1(2)



Overall view of the test suite – File directory structure 2(2)



PART IV: Conclusions and outlook

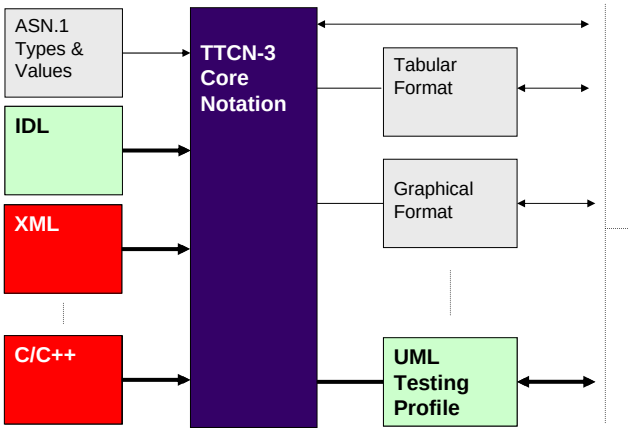
On the user's acceptance of TTCN-3

Conclusions

TTCN-3 extensions

TTCN-3 tool providers

On the user's acceptance of TTCN-3 ingredients 1(4)



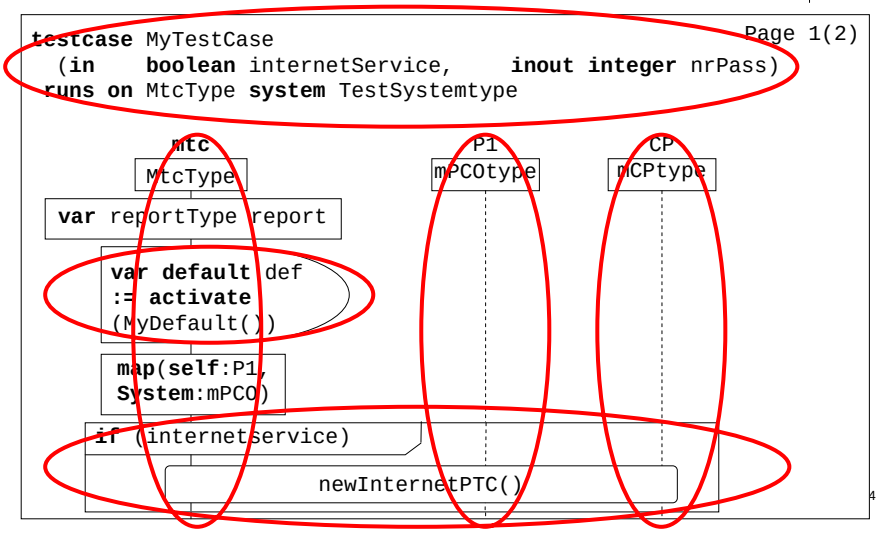
On the user's acceptance of TTCN-3 ingredients 2(4)

- Well accepted
  - TTCN-3 core notation
  - ASN.1 handling
- Strong interest
  - UML testing profile (work not yet completed)
  - XML mapping (work not yet completed)
  - C/C++ mapping (work will start)
- No feedback
  - IDL mapping

On the user's acceptance of TTCN-3 ingredients 3(4)

- Small or no interest
  - Tabular presentation format
  - Graphical presentation format
    - Users don't see an advantage of the one-to-one mapping
    - The UML testing profile (U2TP) mapping to TTCN-3 is no one-to-one mapping, but in some points more like test generation

On the user's acceptance of TTCN-3 ingredients 4(4)



Conclusions

- TTCN-3 finds its way into practice
- Lots of interest in industry as well as in academia in TTCN-3
- Stimulates further work and research
- Still issues to be improved
- Still possibilities to influence the future of TTCN-3

TTCN-3 extensions

- Planned for next edition of TTCN-3 standard in Dec. 2004
- Work items
  - Language extensions mechanisms
    - Packages and profiles
  - Extended communication mechanisms
    - Broadcast / multicast
    - Synchronization / coordination
  - Real-time extensions
    - Absolute time support
    - Time-constrained operations
  - Better performance testing support
    - Implicit test configuration
    - Implicit communication
    - Performance measurement

TTCN-3 tool providers

- Tool Provider
  - Testing Technologies
  - Telelogic
  - Danet
  - DaVinci Communication
  - Strategic Test Solutions
  - Open TTCN
  - Internal
    - Nokia
    - Ericsson
    - Motorola
- Test Devices
  - Alcatel A1100
  - Navtel InterWatch
  - Nethawk
  - Tektronix G20

Literature on TTCN-3

Standard documents
Overview articles
Graphical presentation format
TTCN-3 control and runtime interface
IDL to TTCN-3 mapping
TTCN-3 real-time extensions
UML Testing Profile

Literature on TTCN-3 1(6)

- Standard documents: (can be found on the TTCN-3 homepage: http://www.etsi.org/ptcc/ptccttcn3.htm)
  - ES 201 873-1: TTCN-3 Core Language
  - ES 201 873-2: TTCN-3 Tabular Presentation Format (TFT)
  - ES 201 873-3: TTCN-3 Graphical Presentation Format (GFT)
  - ES 201 873-4: TTCN-3 Operational Semantics
  - ES 201 873-5: TTCN-3 Runtime Interface (TRI)
  - ES 201 873-6: TTCN-3 Control Interface (TCI)
  - TS 102 219: The IDL to TTCN-3 Mapping
- Example test suites:
  - See: http://www.etsi.org/ptcc/ptccttcn3.htm
  - E.g., http://www.etsi.org/ptcc/ptccsip_osp.htm

Literature on TTCN-3 2(6)

- Overview articles
  - Jens Grabowski, Dieter Hogrefe, György Réthy, Ina Schieferdecker, Anthony Wiles, Colin Willcock. *An introduction into the testing and test control notation (TTCN-3).* Computer Networks, Volume 42, Issue 3, Elsevier, Amsterdam, Juni 2003, 375-403.
  - Jens Grabowski, Anthony Wiles, Colin Willcock, Dieter Hogrefe. *On the Design of the new Testing Language TTCN-3.* '13th IFIP International Workshop on Testing Communicating Systems' (Testcom 2000), Ottawa, 29.8.2000-1.9.2000, Kluwer Academic Publishers, August 2000.
- Graphical presentation format
  - P. Baker, E. Rudolph, I. Schieferdecker. *Graphical Test Specification - The Graphical Format of TTCN-3*. Proc. of the 10th SDL Forum 2001, Copenhagen, June 2001.
  - E. Rudolph, I. Schieferdecker, J. Grabowski: *HyperMSC - a Graphical Representation of TTCN*. Proc. of the 2nd Workshop of the SDL Forum, Society on SDL and MSC, Grenoble, June 2000.

Literature on TTCN-3 3(6)

- TTCN-3 control and runtime interface
  - S. Schulz, T. Vassiliou-Gioles. Implementation of TTCN-3 Test Systems using the TRI. Testing Internet Technologies and Services - The IFIP 14th International Conference on Testing of Communicating Systems March, 19th - 22nd, 2002, Berlin, Kluwer Academic Publishers, March 2002.
  - I. Schieferdecker, T. Vassiliou-Gioles. *Realizing distributed TTCN-3 test systems with TCI*. IFIP 15th Intern. Conf. on Testing Communicating Systems - TestCom 2003, Cannes, France, May 2003.
- IDL to TTCN-3 mapping
  - Michael Ebner, Aihong Yin, Mang Li. *A Definition and Utilisation of OMG IDL to TTCN-3 Mappings*. TestCom 2002: Testing Internet Technologies and Services -- The IFIP 14th International Conference on Testing of Communicating Systems March, 19th - 22nd, 2002, Berlin, Kluwer Academic Publishers, March 2002.

Literature on TTCN-3 4(6)

- TTCN-3 real-time extensions
  - Zhen Ru Dai, Jens Grabowski, Helmut Neukirchen. *Timed TTCN-3 - A Real-Time Extension for TTCN-3.* TestCom 2002: Testing Internet Technologies and Services -- The IFIP 14th International Conference on Testing of Communicating Systems March, 19th - 22nd, 2002, Berlin, Kluwer Academic Publishers, March 2002.
  - Zhen Ru Dai, Jens Grabowski, Helmut Neukirchen. *TimedTTCN-3 Based Graphical Real-Time Test Specification.* Testing of Communicating Systems (Editors: D. Hogrefe, A. Wiles). Proceedings of the 15th IFIP International Conference on Testing of Communicating Systems, Sophia Antipolis, France, May 2003. LNCS 2644, Springer, May 2003.
  - Helmut Neukirchen, Zhen Ru Dai, Jens Grabowski. *Communication Patterns for Expressing Real-Time Requirements Using MSC and their Application to Testing.* Testing of Communicating Systems. Proceedings of the 16th IFIP International Conference on Testing of Communicating Systems, Oxford, UK, March 2004. LNCS 2978, Springer, March 2004.

Literature on TTCN-3 5(6)

- UML Testing Profile
  - Online Resources: http://www.fokus.gmd.de/u2tp/
  - U2TP consortium. *UML 2.0 Testing Profile Specification*. OMG Adopted Specification ptc/03-08-03. (for download see: http://www.fokus.gmd.de/u2tp/)
  - Zhen Ru Dai, Jens Grabowski, Helmut Neukirchen, Holger Pals. *From Design to Test with UML -- Applied to a Roaming Algorithm for Bluetooth Devices.* Testing of Communicating Systems. Proceedings of the 16th IFIP International Conference on Testing of Communicating Systems (TestCom2004), Oxford, United Kingdom, March 2004. LNCS 2978, Springer, March 2004.
  - Ina Schieferdecker, Zhen Ru Dai, Jens Grabowski, Axel Rennoch. *The UML 2.0 Testing Profile and its Relation to TTCN-3.* Testing of Communicating Systems (Editors: D. Hogrefe, A. Wiles). Proceedings of the 15th IFIP International Conference on Testing of Communicating Systems (TestCom2003), Sophia Antipolis, France, May 2003. LNCS 2644, Springer, May 2003, pp. 79-94.

Literature on TTCN-3 6(6)

- Further resources
  - More can be found in the proceedings of the TestCom conferences and on the homepages of the TTCN-3 team members (e.g., I. Schieferdecker and J. Grabowski) and the TTCN-3 tool providers.

Thank you for your attention!