4

Application of a LOTOS based Test Environment* on AAL5

J. Burmeister and A. Rennoch

GMD FOKUS, Research Institute for Open Communication Systems
Hardenbergplatz 2, D – 10623 Berlin, Germany
phone: +49 30 25499 - 241, fax: +49 30 25499 - 202
email: burmeister@fokus.gmd.de

Abstract

This paper provides a first experience report on a formal based testing methodology, which we applied on the ATM Adaptation Layer Type 5 (AAL5). A feasibility study is given, which shows the applicability of a formal description technique, namely LOTOS [2], to be suitable as a testing notation. In this approach it is one of our intentions to be able to check a test suite against a formal specification without leaving the language (and its formal semantics) domain.

The methods and tools presented in this paper are addressing the following aspects:

- validation of (possibly manual written) abstract test cases with formal data specification,
- the generation of the corresponding executable test cases,
- the test operation, and
- the visualized animation of the test execution trace.

As a prerequisite, a formalized, but simplified, protocol or service specification is expected, which will be also used for the analysis of the test execution. The generation (creation) of the LOTOS test cases is not addressed here.

Keywords: AAL5 Testing, Protocol Conformance Testing, Quality of Service Verification, Formal Description Techniques, LOTOS, TTCN, ASN.1.

1 Introduction

It was our intention to develop a platform which supports the standardized conformance testing methodology (CTMF) on the basis of a formal description technique (LOTOS).

*This testing environment has been partially funded by the European RACE Project R2088 TOPIC (Tool Set for Protocol and Advanced Service Verification in IBC Environments).

A. Cavalli et al. (eds.), *Protocol Test Systems VIII*

Although the ISO 9646 [4] standard provides its own testing notation, a formal description technique (FDT) has been selected due to some reasons presented in the following. The platform should be open for experiments which go beyond PCT (Protocol Conformance Testing) and TTCN [4, 1]. The use of an FDT like LOTOS makes a link to FMCT (Formal Methods in Conformance Testing, ISO/IEC JTC1/SC21 P54), a joint ISO/ITU-T project, which has already identified the needs of formal methods in protocol and service (conformance) testing.

Future requirements for ODP testing and/or QoS verification (addressed in QoS Framework Project ISO/IEC JTC1/SC21 P57) needs more powerful expressiveness than TTCN is currently able to provide. Unfortunately, TTCN does not have a sound formal semantics, which makes its extension nearly impossible. On the other side, any new test notation has also to integrate the features of the CTMF standard.

Our approach represents an alternative to the development of TTCN compiler which is currently lacking of a sound formal basis for the relationship between formal (FDT) protocol and (TTCN) test specifications. We've avoided any additional model for the test specification and can validate LOTOS test and protocol specifications without leaving the underlying semantics model.

In the following, if LOTOS is referred to as a test notation, only a certain subset of this language will be addressed w.r.t our intention to derive the executable test cases (ETC) automatically as far as possible. Specially, the abstract data types (ADT) are reduced to those, which can be derived from an ASN.1 [3] specification.

The rest of this paper is divided as follows: first, the used methodology is defined (chap. 2), then, the supporting tool environment is introduced (chap. 3). In chapter 4 the applicability of the methods and tools are demonstrated, followed by some further application ideas. And finally, the conclusions are given.

2 Methodological Approach

The objective of the LOTOS based test environment is to automate the generation of the executable test case (ETC) or suite (ETS) from a given abstract test case (ATC) or suite (ATS). Therefore, it is also intended to provide some support in the adaption of the different IUT (Implementation Under Test) interfaces, which are in general distributed, and which need coordination. To be flexible as possible, the test environment is designed to the distributed test method, which is also of important relevance for ODP service testing. For test analysis purposes, post-animation of the behavior of the test system is proposed to get a detailed view of the observations, and to identify those parts of the formal specification which are fulfilled or not by the IUT.

Due to a consequent formal approach in the code generation from an abstract test case, LOTOS and ASN.1 has been chosen as the front end to the user. Data specified in ASN.1 can be mapped into both some implementation languages like C, and also into LOTOS abstract data types, which then allows an implementation semantics for ADTs. Furthermore, an extension of ASN.1 is selected (RO-Notation from ISODE [10]) to define the abstract interfaces, i.e. the data exchange parameters at the abstract communication points.

LOTOS has been chosen due to its programming language design (although it is not a programming language), its checking and simulation features, and its parameterization capability. It has been already recognized, that the application of full LOTOS is not required to use it as a test notation, but some restrictions on that language enables the application of algorithm and tools to make LOTOS fit for being a test notation like TTCN.

Not at least, the LOTOS based test environment shows the integration of different available tools towards a powerful test tool set, which also integrates platform features like remote procedure calls (RPC under SunOS) and possibly ROSY/PEPSY (ISODE).

Figure 1 gives an overview of our methodological approach in technical terms, i.e. by illustrating the logical relationship between relevant documents (specifications and programs). The rectangle in the middle of the figure comprises the executable programs, which realizes the test execution phase:

- the test system, i.e. a master tester with its upper/lower tester agents and individual timer servers, and
- the system under test (SUT), which includes the IUT and its environment (e.g. communication services).

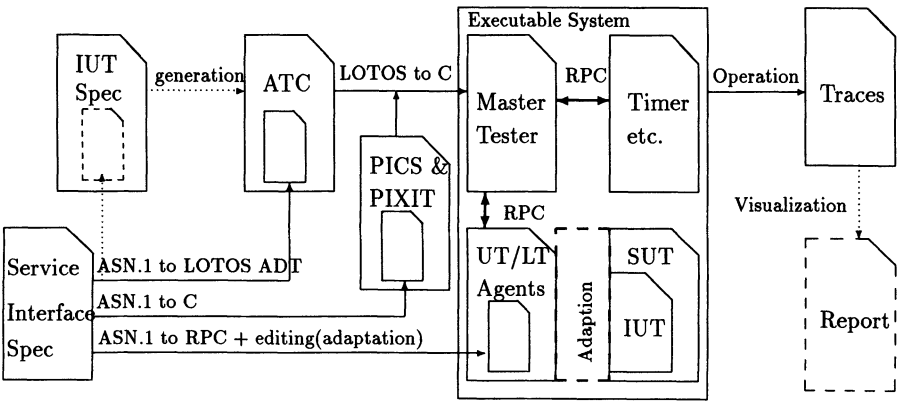

Figure 1: LOTOS based Test Approach

On the left side, which presents the preparation phase, the IUT specification and the service interface specification constitutes the starting point for the production and completion of the ATC and the PICS/PIXIT documents, which are input for the test system compilation.

The right side of the figure illustrates the test traces from the test operation, which will be used for visualization and test report production.

2.1 Preparation Phase

In this paper the term preparation refers to the tester preparation w.r.t. the IUT interfaces and the integration of additional context information (like PICS and PIXIT, Protocol Information Conformance Statement and eXtra information for Testing). It is not referred to the generation of abstract test cases, which is out of the scope of this paper.

In the following three aspects of preparation are discussed.

Abstract Service Interface Definition

For each test case (suite) an abstract service interface definition has to be specified formally, which in fact will be defined once. This task may be part of standardization bodies in the future.

The definition, usually specified in ASN.1, comprises the used data types and the communication points. And using the RO-Notation, the location of the data exchange can be described in an abstract style. An example is given later in chapter 4.

Adaptation of the IUT Interface

It is obvious, that for each different implementation of an IUT some implementation code must be generated manually for its adaptation. Since the LOTOS based test environment is designed to be an open tool, which should be able to adapt to a wide range of communication interfaces (in C), it uses and works only on (a subset of) abstract data types. The approach here is to provide a basic implementation representation (in C) of the abstract interface specification to reduce the manual involvement only to the implementation of the mapping between this (C) representation and the individual IUT interface.

The creation of the (C) representation has to be done only once for each protocol/service specification, but the final mapping to the concrete service primitive calls has to be done for each implementation.

Parameterization of a Test Case

Abstract test cases are abstract in the sense, that they do not refer to concrete objects like machines, implementation languages, and communication interfaces. If an abstract test case will be made executable, some specific values from the testing environment and the IUT must be taken into account. Normally, all these information are collected in the PIXIT document. Host names of the machines, time limitations, buffer and data constraints, and other information and requirements are to be provided.

Using LOTOS as a test notation, the user can easily integrate all requested parameters in the specification, as it can be seen later in chapter 4.

2.2 Execution Phase

The interpretation semantics of TTCN is weakly formalized in a procedural program notation like PASCAL. Based on a snapshot of the queues at the PCOs, an interpretation along the events is defined. It is not clear what will happen in case of events to be ignored.

If they are not removed from the input queues, any stored event might match to a future expectation, which was not intended. On the other hand side a queue might contain too many events due to the fact, that the frequency of taking a snapshot can be too low. In fact, the snapshot semantics is firstly to take a snapshot, secondly, to try to match the events, and thirdly to take a next snapshot. It is not based on what to do with the most recent observed event.

Other features of TTCN are the OTHERWISE construct, and that timers are dealt separately. Again, it is not clear how these features can be reflected properly in the executable code, e.g. where to locate the timers in a distributed environment. This comes along with the problem of the evaluation of the distributed input queues of the different PCOs. Although timeouts are defined, in time critical applications the time for the coordination and transmission of the observed events via e.g. a ferry clip must be taken in to account.

In this paper a slightly different execution method for abstract test cases is proposed. FDTs, which are already available since 1988, can be used to derive a so-called master tester, which is primarily independent from any test method. With some restrictions on the FDTs it is possible to compile the specification without loosing their usability as a test notation.

In our approach, LOTOS has been selected for the test specification notation (as well as for protocol and service specification). The following concepts and requirements for test notations are supported by LOTOS:

parameterization: The definition of external gates corresponds to the PCOs. Other specification parameters, which have unknown values during the test case generation, must be instantiated in the executable test, which are e.g. concrete host names (locations), timer values or variances of them, data to fill buffers etc. .
If a test specification consists of a test suite, the LOTOS parameterization technique can be also used to select and/or exclude certain test cases with respect to a given PICS document.

verdict assignment: At certain places within a specification, verdicts are required. This can be expressed by the `exit` statement (together with an appropriate `verdict` data type) and the functionality of processes and the specification itself. In case of a conformance test, also the final verdict can be directly specified.

structuring: The process definitions are suitable to build hierarchies for test groups, test cases, and test steps. Required loops are presented by recursive instantiations of processes.

constraints: Like in procedural programming languages, temporary variables can be declared, and values can be assigned. The `guards` in combination with the events express constraints (conditions) on parameter values.

test event: In general, an event is expressed by an `action` in LOTOS. Normally, a test event is composed of the PCO, where the event must occurs, and a data unit to be send or to be await. In LOTOS, the PCO is defined by the **gate**, and similar to TTCN, the sending and the expectation (receiving) of an event is expressed by '!' and '?' resp. .
The semantics of LOTOS is based on synchronous communication of events. Using LOTOS here as a test notation, we can retain this semantics in the sense, that the communication via PCOs (and thus with the environment) must be implemented synchronously. That means, that a sending event is always successful (synchronization with the environment), and a receiving event is then successful, if it has gotten the required (and possibly by a **guard** constrained) data.
By this concept, we define the basic different interpretation of an abstract test case in opposite to TTCN. A master tester has to react on each exchange of data at any PCO immediately.

test sequence: The sequential ordering of events is provided by the `action denotation` of LOTOS.

alternative: Alternate behavior of an abstract test step is supported by two operators of LOTOS. The disable operator `[>` is suitable to express the interruption of a test step due to some more general events like timeouts or disconnecting events. And the choice operator `[]` will be used in the sense of an `if-then-else` construct, either constrained by the environment, which offers an appropriate event, or conditioned by **guard** expressions.

timer: Timers are not directly supported by LOTOS. Nevertheless, on the abstraction level of abstract test cases, a timer can be seen as a service, which provide appropriate functionalities at certain interfaces. The event at these interfaces can be expressed exactly in the same style as it will be done for the PCOs.

otherwise: Unless reliable timer can be used, timeouts in combination with the disable operator will catch every deadlock situation, which might come from a blocking of receiving events which do not get anything. Thus, it's just a question of specification style.

data presentation: As to the behavior description feature of LOTOS, its expressiveness of abstract data must be restricted. Since ASN.1 is one of the most popular data description techniques for communication protocols and services, the LOTOS ADTs are reduced to those which are related to ASN.1 data types.
The advantage of using ASN.1 is the abstraction from programming languages, and it is suitable for the expression of concrete structures necessary for protocol and service data unit descriptions. Several relationship definitions between ADTs and ASN.1 are already defined.

The above listed requirements for a test notation are completely supported by a well-defined subset of LOTOS, which supports therefore the automated generation of an executable test case from an abstract one.

2.3 Analysis Phase

Each ATC should be related to a specific path in the given specification. Verdicts are assigned with respect to the preamble, the test steps, and the test events of the test case. A concrete verdict will be given depending on the actual behavior of the IUT. The relationship between the test sequence and the corresponding path in the specification has to be provided manually and may expressed via comments in the ATC. After the test run, an appropriate test report has to be provided. In case of IUT failures the implementation supplier is interested in those parts of the specification, which have not been satisfied by the IUT.

If the protocol or service has been specified formally, the observed trace (also expressed in an FDT) can be combined with the formal protocol or service specification. Using appropriate visualization tools, a post-animation of the test operation is possible, which shows the final result, and which also indicates the test steps, which has been undertaken, and which part of the specification has been either fulfilled (finishing with `pass`) or couldn't be verified (finishing with `inconclusive` or `fail`).

Similar to the validation of the abstract test suite against the specification, the obtained traces identify the specification parts, which have been verified in the test operation. The post-animation of the test run helps to reconsider the test without its repetition. The methodological approach here is to combine the specification together with the obtained traces plus some testing environment services like e.g. timers and verdict processors.

3 Tool Set Implementation

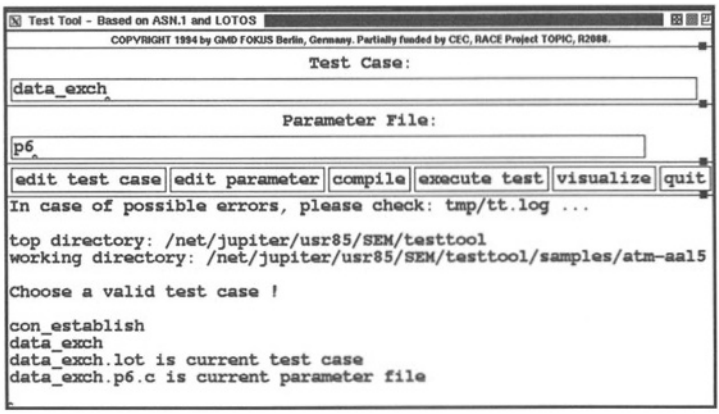

Figure 2: Graphical User Interface of the Test Tool `tt`

The current status of the test tool set (`tt`) supports the user in various ways in the three testing stages of preparation, execution, and analysis. Namely, three tools

have been developed, and some tools of the TOPO tool set have been selected (previous versions are part of LITE [8]). A user interface is provided, which provides all these tool functionalities without presenting them explicitly. This user interface is a rapid prototyping of an X11 window application, and can be replaced easily by other (modified) user interface implementations (see fig. 2). In fact, a command line interface would fulfill the same tasks, so this tool environment could also run independent of any window surface.

In the following, the tools and their functionalities are briefly explained w.r.t. the three testing stages. They are following the methods given in the previous chapter. The automation process requires a certain file subsystem strategy. Since the executable test system will be derived from a selected abstract test case, all required information must be located in certain directories under particular names. Here, we abstract from file name conventions, but indicate the logical file categories.

3.1 Preparation Tools

The preparation phase is supported by two tools dealing with the data specifications, the interface specification, and the data transmission between distributed hosts. The presented tool environment supports the presentation of abstract data using ASN.1. More concrete, the RO-Notation from the ISODE platform has been identified as a suitable abstract description of data together with the interface description. Such an abstract description of data and interfaces allows the use of tools which provides encoding and decoding facilities to exchange data between machines (hosts), hence, using a distributed platform like ISODE or ANSAWare [12].

In our approach, we use an ASN.1 to C Compiler (a tool called `asn`), whose output is suitable for `rpcgen` [9], a tool under SunOS. The same ASN.1 compiler generates the LOTOS ADTs as required for the abstract test case. The used mapping mechanism have been reported in the TOPIC deliverable *Verification Tools (V2)* [6].

The current status of the tools does not support the automated generation of appropriate operations on data yet (e.g. comparison), neither in LOTOS ADT nor in C. They must be provided manually, but can be automated in the future.

Another task, which must be provided manually, but which cannot be automated, is the mapping between abstract represented data in C and the concrete data types used by the IUT at the PCOs.

3.2 Execution Tools

The transformation of the ATC into in ETC is supported by a set of tools, managed by a user interface, which combines the different functionalities (see fig 2). The testing tool set (`tt`) needs a directory for the configuration of the executable test of the IUT. This directory must be divided into the following subdirectories:

- a temporary (`tmp`) directory, which contains all the generated and copied files to produce the master tester;
- a PIXIT (`pixit`) directory, which contains the generated (RPC) coordination interfaces and the abstract data types to be used in the abstract test cases;

- a test suite (`testcases`) directory, which consists of the different test cases for a certain protocol or service specification, and which also contains a LOTOS specification of the testing environment, which will be completed and used later for the visualization of the test run (see next section);

- a binary (`bin`) directory containing the executable interfaces (RPC demons) to the IUT and/or underlying service (PCOs), which are generated from the interface specifications given in the PIXIT directory; other binaries may also located here;

- optional directories (`include` and `lib`) which refer to additional implementation details of the environment (libraries and header files) and the IUT to access the PCOs. Further directories are omitted, which store the source code (`src`) of the IUT (if available at all), documentations (`doc`), and/or manuals (`man`).

The main feature of the testing tool is to generate and run the Master Tester. For that purpose, the PIXIT documentation must be available from the preparation stage, i.e. the LOTOS abstract data types are generated, and the interface routines of the PCOs are prepared. Then, `tt` combines the selected test case with the data types, checks the obtained specification with TOPO tools, and takes the generated so-called *common representation* of LOTOS to produce the Master Tester using a LOTOS to C compiler. In fact, basically the behavior part of LOTOS is compiled as follows, and the data elements are mapped to those C data types and operations already generated by the ASN.1 tool.

The generated code implements the test events with RPCs, which are in fact implemented as synchronous calls. A racing strategy has been implemented, which discards all other events, if one of the set of alternative events has been occurred. Note, that the parallel operator is not implemented; firstly, it is not a requirement for conformance test notations as listed above (but may be one in future to test e.g. platforms and distributed services), and secondly, only the interleaving operator `|||` and the partial synchronized parallel operator `|[..]|` make sense, whereby the latter one uses internal (hidden) gates to synchronize.

The generated trace consists of a sequence of actions together with the verdict of the test case, which can be easily transformed into a LOTOS behavior expression (see log file in chapter 4).

3.3 Analysis Tools

The analysis of the test run is supported by a visualization tool. The input specification for the DEMON [11] visualization tool is composed of the IUT specification, the observed trace, and some additional specification parts like e.g. used timers. Furthermore, some visualization information must be integrated as annotations. In this approach, the test operator and the IUT supplier can analyze the observed communication events. In case of a failure, a backwards reference to the specification is feasible to indicate the part of the specification, which has not been implemented correctly in the IUT. Note, that the visualization information does not change the semantics of the specification itself like e.g. other annotations will do for code generation of e.g. a state machine.

An excerpt of the visualization specification as used for our AAL5 experimentation looks as follows:

```
specification aal_env_spec[U (*# region 2 #*), ... , T2 (*# region 5 #*),
                           info (*# region 5 #*) ]:noexit
  (*# user_defined_icons
   region_definitions
   GRID R3 50 665 8 8 1.0; # timer & verdict
...
   GRID R7 140 30 8 8 2.0; # aal
  #*)

   type number is ... endtype

   type primitives is
     sorts primitive
     opns
        a_connect_req, ... , a_close_req,
        pass, ..., tt_ti_start, tt_ti_expired :-> primitive
   endtype

   behavior
       test_system[U,L,T1,T2,info]
     ||
       (aal[U,L] (1) ||| timer1[T1] ||| timer2[T2] ||| verdict[info])
  where
     process aal[U,L](PN:number):noexit:=
       (*# region 7 #*)
     (  U ! 1 ! a_connect_req ; (* The appropriate AAL5 Specification *) ... )
     endproc

   process test_system[U,L,T1,T2,info]:noexit:=
       (*# region 1 #*)
#include "auto.lot"
       (* Integration of the observed trace expressed in LOTOS *)
     stop
   endproc

   (* Specification of the environment: *)

   process verdict [info]:noexit:=
         (*# region 3 #*)
       info ! 1 ! pass ; stop
     [] info ! 1 ! fail ; stop
     [] info ! 1 ! inconclusive ; stop
   endproc

   process timer1[T1]:noexit:=
         (*# region 3 #*)
       T1 ! 1 ! tt_ti_start ; timer1[T1]
     [] T1 ! 1 ! tt_ti_expired ; timer1[T1]
   endproc
   process timer2[T2]:noexit:= ... endproc
endspec
```

Another LOTOS compiler will be used, which generates the appropriate input code for DEMON, using the annotations '(*#' and '#*)' for visualization support. The current state of the art of this tool supports only very simple data types, but the tool is powerful enough to express at least the used abstract primitives.

In case of the AAL testing environment, we consider four independent processes (AAL5, 2 timers, verdict processor), which are synchronized by the test system, whose behavior is defined by the actual observed trace (`auto.lot`, derived from the log file). The visualization is presented in the following chapter.

4 Experiments with AAL5

First experiences have been made within the RACE Project TOPIC. Here, the service over XTPX [7] has been tested. After extending the testing tool set, tests have been applied on the AAL5 service [13].

We have used Sun work stations under SunOS 4.1.x. Two machines were equipped with FORE ATM adapter cards, together with the AAL software. The machines were connected via an ATM FORE ASX 200 switch (a third machine). One of the applied tests just only checks the sending and receiving of data:

```
specification test_data_exchange [T1,T2,L,R]
  (remote: CHARACTERSTRING, any_qos, sel_qos: DL_qos,
   data_flow: DL_dataflow, data: CHARACTERSTRING, data_len: INTEGER,
   to1, to2: INTEGER) : exit(verdict)

library Verdict endlib
library Timer endlib
library tt_primitive endlib
#include "../pixit/datatypess.adt"

behaviour
 test_data_exch [T1,T2,L,R]
  (remote,any_qos,sel_qos,data_flow,data,data_len,to1,to2)
 where
  process test_data_exch [T1,T2,L,R]
   (remote: CHARACTERSTRING, any_qos: DL_qos, sel_qos: DL_qos,
    data_flow: DL_dataflow, data: CHARACTERSTRING, data_len: INTEGER,
    to1, to2: INTEGER) : exit(verdict) :=
   T1 ! tt_ti_start ! to1 ;
   ( R ! a_connect_req ! SEQ_A_Con_Send(dest      (remote),
                                        qos       (any_qos),
                                        dataflow  (data_flow));
       L ! a_connect_ind ! data_flow ? re: A_Con_Send;
        L ! a_connect_resp ! SEQ_A_Con_Receive(qos      (sel_qos),
                                               dataflow (data_flow));
         R ! a_connect_conf ? re: A_Con_Receive;
          T2 ! tt_ti_start ! to2 ;
          ( R ! a_data_req ! SEQ_A_Data_Send(buf (data), len(data_len)) ;
             L ! a_data_ind ? re: A_Data_Send;
             exit(pass)
          [>
            T2 ! tt_ti_expired  ? x:BOOLEAN ;
```

```
            exit(fail)
          )
    [>
      T1 ! tt_ti_expired  ? x:BOOLEAN ;
      exit(inconclusive)
    )
  endproc
endspec
```

The identifiers (e.g. `a_data_request`) of this specification have been taken in particular from the abstract service interface definition (after its mapping into LOTOS ADTs (`datatypes.adt`)). A small excerpt of this interface definition looks as follows:

```
AAL5-CT DEFINITIONS ::=
BEGIN ...
DL-dataflow ::= ENUMERATED
 { simplex(0),
   duplex(1),
   multicast(2) }
...
A-Data-Send ::= SEQUENCE
 { buf        CHARACTERSTRING,
   len        INTEGER }
...
a-data-req                          -- Abstract Data Request Primitive
    OPERATION
    ARGUMENT   A-Data-Send
 -- RESULT None
    ERRORS     { ... }
    ::= 5
...
END
```

The abstract test specification has been parameterized with different values for time out and data formats. Arbitrary user data have been taken, because no specification was available during the experimentation phase. An example parameterization for selected PIXIT values is given in the sequel, provided in ASN.1. The identifier names like `DL-dataflow` have been taken from the above abstract service interface definition:

```
PIXIT-NO-6 DEFINITIONS ::=
BEGIN
  IMPORTS DL-qos, DL-dataflow, simplex FROM AAL5-CT ;
remote CHARACTERSTRING ::= "\000\000\000\001\361\044\015\013"
any-qos DL-qos ::= { { 256 , 128 } , { 128 , 64 } , { 2 , 1 } }
sel-qos DL-qos ::= { { 256 , 128 } , { 128 , 64 } , { 2 , 1 } }
data-flow DL-dataflow ::= simplex
data-len INTEGER ::= 4096
data CHARACTERSTRING ::= "\
012345678901234567890123456789012345678901234567890123456789012345678901234567890123\
...
012345678901234567890123456789012345678901234567890123456789012345678901234567890123"
to1 INTEGER ::= 400  -- ms
to2 INTEGER ::= 200  -- ms
END
```

Note, that at the current state of the art the ASN.1 tool does semantics checks on data type definitions only, and therefore data values must be compiled manually into C, e.g. the `remote` value:

```
CHARACTERSTRING remote = { 8 , "\000\000\000\001\361\044\015\013" } ;
```

In the PIXIT example above the general timeout (`to1`) has been set to 400 ms, whereby the data transmission phase (`to2`) has been limited to 200 ms. As you can simply identify, the ATM host address is given as an extra information for testing, since the abstract test case does not know concrete addresses of the testing environment.

Furthermore, the parameterization of the PCO (`L, R`) and timer gate (`T1, T2`) locations shows the independence of these interfaces from the generated (master) tester location. Due to the generation of RPCs, the (master) tester could be located on a fourth machine.

Assuming, that the first timeout value `to1` has been chosen to small, the verdict will lead to an inconclusive verdict. The appropriate log file looks as follows:

```
(* Started: Mon Mar 27 18:41:21 (723 ms) 1995 *)
T1!tt_ti_start!to1
R!a_connect_req!SEQ_A_Con_Send(dest(remote), qos(any_qos), dataflow(data_flow))
T1!tt_ti_expired?x:BOOLEAN
exit(inconclusive)
(* Finished: Mon Mar 27 18:41:22 (190 ms) 1995 *)
(* Finished with verdict inconclusive *)
```

This log file can be combined with the formal service specification or, if not available, with a simplified version, and run by the visualization tool DEMON. The post-animation demonstration is presented in figure 3. The example illustrates a timeout during the connection establishment, i.e. here it results with an inconclusive verdict. Therefore, the test operator and the IUT supplier may conclude, that either the preamble does not work properly, or the connection establishment is not correct implemented.

The presented test approach is not fully applicable to do performance measurements. E.g. specially in the realm of AAL testing, the exchange of data within the IUT is much more faster as it is provided by the tester coordination procedures (like RPCs). Furthermore, as long as timers have to be integrated like additional services, a certain delay must be taken into account. Hence, like in TTCN, timers should be used only to limit the duration of a set of events.

On the other hand side, long time statistical measurements are feasible. E.g. the transmission of data by an appropriate loop within a certain time frame can be expressed easily.

5 Conclusions and future Plans

This paper has shown the feasibility of using an FDT (LOTOS) as a suitable test notation. It has focussed on the distributed testing platform which implements certain features of a restricted subset of LOTOS. The problem of test case generation has been left out; and to validate a test specification against the protocol or service specification, certain constraints like e.g. a common data type part specification must be fulfilled. In this paper, ASN.1 is proposed to define the minimal set of (abstract) data types.

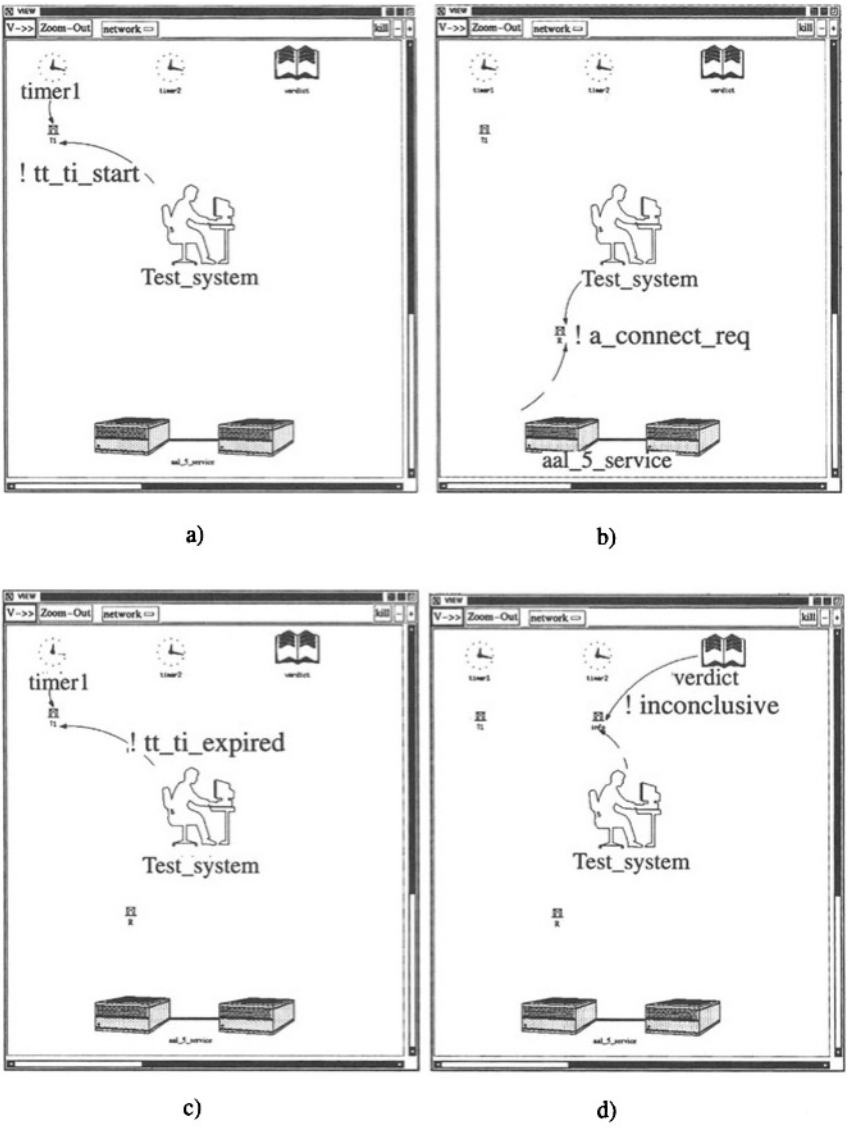

Figure 3: AAL5 Test Trace Visualization

A set of advantages makes LOTOS to be an alternative to TTCN:

1. it is based on a sound formal method and its semantics is well defined;
2. the relationship between an abstract test case and the corresponding specification parts (if it is also formalized by an FDT) can be verified (specially in case of manual generation);
3. the derivation of the executable test suite is supported due to the sound semantics definition of LOTOS;
4. the parameterization of an abstract test case is an integrated part of the notation;
5. PCOs are not restricted to service access points of the IUT, and may also include other test environment components like timer, and, in future, e.g. traffic generator and CPU loader.

Since this feasibility study has been realized by a prototype, certain limitations have to be accepted. E.g. the availability of formal specification and its corresponding (formal) test suites are very rare, especially on new protocols and services like ATM and the adaptation layers. In fact, very rudimentary ATM-AAL specification and test specifications have been developed manually to demonstrate the distributed testing platform.

The current testing tool set covers the conformance testing methodology from generated abstract test cases, their execution and analysis, up to a preliminary test report generation.

Furthermore, extensions are feasible, which make this tool set and its testing approach also suitable for QoS testing. PCOs can be introduce in an abstract test suite to control any traffic generator or CPU load, and recursive process instantiations can be used to specify loops for statistical measurements. On the other hand side, language extensions of the FDTs are desirable w.r.t. their testing purpose and for their applicability to quality testing aspects including time and resource constraints on events.

Acknowledgment

This testing environment has been partially funded by the European RACE Project R2088 TOPIC (Tool Set for Protocol and Advanced Service Verification in IBC Environments, see also WWW: `http://www.fokus.gmd.de/step/topic`). We would like to thank J. de Meer, technical leader of the TOPIC project, who invented and discussed with us the basic ideas of the distributed test system, and also B. Stepien as a project consultant form the University of Ottawa, Canada, who worked on the implementation of the visualization concepts.

References

[1] B. Baumgarten, A. Gießler; OSI Conformance Testing Methodology and TTCN, North-Holland, Amsterdam, 1994.

[2] ISO, IS 8807, Information Processing Systems, Open Systems Interconnection, *LOTOS–A Formal Technique Based on the Temporal Ordering of Observational Behaviour*, July, 1988.

[3] ISO/IEC DIS 8824-1, *ASN.1. Abstract Syntax Notation One – Part 1, Specification of Basic Notation*, 1992.

[4] ISO/IEC IS 9646, Information Retrieval, Transfer and Management for OSI: *Conformance Testing Methodology and Framework*, (1991).

[5] RACE Project R2088 TOPIC, *Integration of the Toolset Prototypes (V2)*, Deliverable 16, Ref. R2088/GMD/SEM/DS/L/016/b4, 1994.

[6] RACE Project R2088 TOPIC, *Verification Tools (V2)*, Deliverable 18, Ref. R2088/GMD/SEM/DS/L/018/b1, 1994.

[7] RACE project R2088 TOPIC, *Experience Report of the Verification Demonstrator*, Deliverable 20, Ref. R2088/CLE/TEE/DS/P/020/b1, December 15th. 1994.

[8] ESPRIT Project 2304 LOTOSPHERE, *LITE User Manual*, Ref. Lo/WP2/N0034/V08, 1992.

[9] Sun Microsystems, *Network Programming Guide*, Part Number 800-3850-10, 1990.

[10] ISODE, The ISO Development Environment: User's Manual Vol. 1, *Application Services*, & Vol. 4, *The Applications Cookbook*, 1991

[11] DEMON, *Reference Manual V3.0*, Mari Computer Systems Ltd, 1993.

[12] Architecture Projects Management Limited, *ANSAware Version 4.0 Manual Set*, Cambridge, UK, March 1992.

[13] FORE Systems Inc. , *ATM Devices and Network Interfaces, Manual Rel. 3.0*, June 1994.