Test Data Filtering

A. Rennoch, J. de Meer, I. Schieferdecker

GMD Fokus, TIP
Kaiserin-Augusta-Allee 31, D-10589 Berlin
www.fokus.gmd.de/tip

Abstract: The paper discusses the need of test data filtering within traditional conformance testing methodology and advanced QoS data stream controlling. The common approach is the introduction of a data profile declaration. This profile declaration is useful for the (semi-)automatic generation of e.g. TTCN data constraints or object interface filters. The B-ISDN ATM Adaptation layer specification Type 1 is used as an application example in both application areas.

Keywords: Conformance testing, stream controlling, TTCN, ODL

1 Introduction

Standardized Conformance Test Suites consist of three major parts:

(a) the test behaviour sequences,

(b) the data structure and type definitions, and

(c) the data value constraints, i.e. the concrete description of test data to be sent (selected values) to the IUT and to be received (expected values) from the IUT.

A lot of effort has been undertaken to automize the generation of test suites by means of formal descriptions of the so-called System under Test (SUT). Notice, SUT is a notion of the ODP Con-

formance Testing Methodology [7] which describes the compound system in which the implementation to be tested resides. From a practical point of view there exists long-term experience with commercial tools [1] on automatic generation of test behaviour from formal specifications, e.g. FSM based [5]. Work has been done also on developing mapping rules from formal data type specification written in SDL [2] or ODL [3] to TTCN or ASN.1 respectively. Unfortunately less support is available on developing constraints on data for test case generation.

Practical experiences in test suites development projects at our FOKUS test suite factory group show that especially this part of the test suite is important for the variety of test cases. Looking at today's test suites a lot of test cases are similar in behaviour but differ in the handling of data, e.g. only one parameter component differs only by one distinct value. The whole range of the value space of a parameter normally is not required to be tested. Tool support in the development of data constraints to avoid exhaustive data type testing consists currently only in the generation of an empty TTCN constraint skeleton, i.e. a constraint table which includes the component (TTCN field) names (with comments) and the data type, but not any coded data values.

The derivation of a set of significant test data from the unconstrained set of all test data (due to the value definition space) could be understood as a process of test data filtering. An infinite number of possible test data is due to the infinity number of elements in some data components, e.g. integer fields in PDUs or ASPs. Then we have to filter some test data used for the test cases (see Figure 1). In addition to this filtering of test data for test suite development we have taken a similar view on data filtering in the context of QoS measurements and controlling of continuous data streams [4]. In the latter case we have to specify which cells of a data stream have to be considered for QoS testing. Also, a test data filter is needed to distinguish stream elements selected for QoS controlling from those traffic elements which must not be considered in the QoS control process. From the technical point this view is similar to the idea of TTCN data constraints for observing test events which allow the identification and distinction of incoming data. In this context we put emphasise on receiving continuous streams and an integration of the constraint specification within the object interface definition.

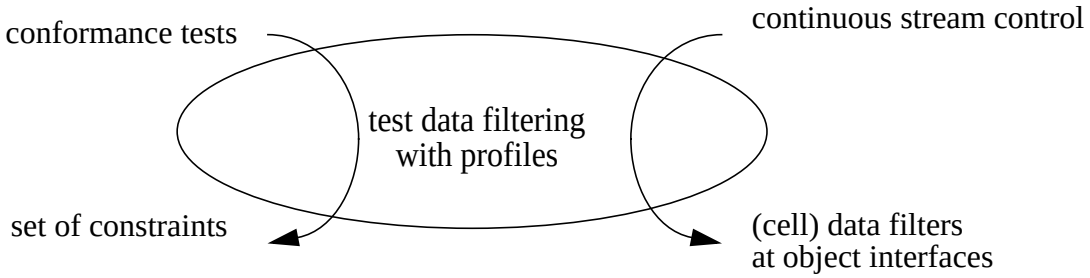

Figure 1: Different views on test data filtering

In the following we have to distinguish between the logical description of a data filter, called 'profile' and the instantiations of the profile in (TTCN) test data constraints or its usage at stream interfaces. Both approaches make use of data (structure) type definitions and the profiles. The instantiation of profiles take place in the context of (TTCN) test suite development only.

Section 2 presents various concepts for the definition of profiles. In sections 3 and 4 we apply these concepts to an example in the context of ATM, the B-ISDN ATM Adaptation layer specification Type 1, whereas section 3 integrate the ideas into TTCN and section 4 addresses the control of cell streams by the profile.

2 Profile declaration concepts

This section presents a series of abstract concepts needed to describe a profile for a data filter. Each of the concepts could be applied to a field or substructure of a data type. In each of the rules a data type field or data substructure get one or more instantiated rules assigned. We will use the following syntax for our rules:

EXPLANATION:

Data type field name : PROFILE,

for TTCN data structures and more abstract:

Substructure name : PROFILE.

The filtering of test relevant data addresses the question of how to restrict data value spaces by constraints. The theoretical data value space results from all combinations of values of all involved parameters. By the specification of appropriate constraints the value space becomes dramatically reduced. State of the art e.g. provided by the ASN.1 notation is explicit enumeration of the encoded values of the data type.

First we want to restrict the range of value intervals and to limit value lists, e.g. we map a value set to a restricted range or subsets:

(1) RANGE:
Field1 : [1.0, 2.0],
i.e. if *Field1* is of type Real, only real numbers between 1.0 and 2.0 are possible.

(2) SUBSET:
Field2 : (2, 4, 6, ...),
i.e. if *Field2* is of type Integer, only even integers are possible.

Further we need possibilities to predefine a limited number of discrete values instead of unlimited value sets.

(3) VALUELIST:
NetworkAddress : (1.2.3, 1.4.6)
i.e. the network address space is limited to two specific addresses 1.2.3 and 1.4.6.

It must be noted that this approach has already been applied in the context of validating formal specification in order to reduce the number of environment signals [2].

The explicit fixing of data values reduces massively - in the sense of ASN.1 - the amount of possible data constraints on ‘protocol data units’ (PDUs) or on ‘abstract service primitives’ (ASPs) and hence, leads to a smaller set of data value combinations. In the context of testing some parameters, e.g. network transit lists (i.e. address values out of an unlimited address space), can not be restricted. They are subject of the testing environment, i.e. they have to be

regarded as parameters to the test, and therefore part of a PICS (protocol implementation conformance statement) or PIXIT (protocol extra information for testing) document. For this case, we propose to mark the test suite parameters before the test data constraint development starts. In the test data constraint development process we have to consider one variable instead of an (unrestricted) value space of the definition set (with lots of values) of the parameter.

(4) PROFILE PARAMETER:
Field1 : *tsp_param1*,
i.e. the value space for *Field1* is mapped to one profile parameter *tsp_param1*.

Often, we can observe that test data constraints differ in one parameter value only. The test case purpose (i.e. the subject of a test case body) is focusing on testing one parameter (say: one data constraint field value) only, e.g. a limiting value or an invalid value. We propose to mark such special values before the test data constraint development process starts. This should remind test suite developers or any (semi-) automatic test data development tool to consider such special values and to produce appropriate data constraints to be used in test case behaviour. In such complex sub-structured test data constraints need to be copied, the (sub-)constraints are to be renamed (i.e. new cross-references are necessary) and the single modified test data values have to be introduced, e.g.:

(5) SUBSTRUCTURES:
PDU1.Parameter3.Field2.Subfield1 : 5,

i.e. the integer value 5 is required in the specified PDU sub-field. This should lead to an individual data constraint in addition to a basic set of value combinations due to the other profile rules.

Further the specification of the test data restrictions could be supported using a random selection of a number of valid or invalid (i.e. values which are not within the definition set of the corresponding data type) value:

RANDOM SELECTION:

(6) *Field1* : ANY

(7) *Field2* : TAKE *n*

(8) *Field3* : TAKE *n* INVALID.

We intend to implement some kind of control program which takes a profile declaration and delivers a set of test data constraints to be used within test case behaviour. Profile declarations could be considered as some intermediate form between encoded data (TTCN send constraints) on one side and data structure definitions (general TTCN receive constraints which allows ‘any’ or ‘not present’ only) on the other side.

Due to complex structure of some PDUs the presence of (sub)fields might depend on field values or the presence of other (sub)fields. For this reason we need an introduction of crossreferences with the semantics of ‘if’ and ‘exclude’, e.g.:

CONDITIONAL SELECTION:

(9) *Field1* : IF *Field2* = ’1’,
i.e. *Field1* is present only if *Field2* has the value ‘1’.

(10) *Field3* : EXCL *Field4*,
i.e. *Field3* is present only if *Field4* is not included.

We have to notice a distinction between the constraints for sending data units and those of receiving data units. In case of receiving data units unconstrained fields will get the ANY value and would not cause the generation of the variety of constraints with the different values allowed by the field data type.

Looking at current TTCN (part 3 of [7]) we found language concepts to restrict integer values to ranges, value lists and even complementary value sets. Test suite developers may also use the ANY, OMIT and ANYorNONE features instead of values and could use all these suite operations defined in the test suite but no means for an explicit restriction of the amount of values and value dependencies. To our knowledge fourthcoming versions of TTCN will not address constraint specifications compared to our proposal.

3 TTCN Test data constraint generation example

The data structure of AAL1 cells is simple and should serve as a first example. AAL1 PDU consists of a PDU header (2 x 4 bits), a pointer field (1 octet) and the payload field (46 octets) [6]. The structure of a convergence sublayer PDU with pointer information (*CS_PDU_P*) is presented in Table 1.

	Field name		length
PDU header:	SN	CSI	1 bit
		SeqCount	3 bits
	SNP	CRC	3 bits
		Parity	1 bit
	Pointer	Parity	1 bit
		Offset	7 bits
	Payload		46 octets

Table 1: AAL1 PDU structure *CS_PDU_P*

An example profile declaration for the AAL1 cell is given in the Figure 2. In contrast to standard TTCN we prefer to collect all statements on data selection in one table instead of using so many sub-constraints for subfields even if the data type definition is more substructured. Therefore we need means to reference subfields. In our example we have used the ‘dot‘-notation proposed in the rule 5 (see above). For simplification we omit the current field names, i.e. we use ‘.SeqCount‘ instead of ‘SN.SeqCount‘. The reference for CRC and EvenParity must be interpreted as test suite operations.

With the profile declaration as presented in Figure 2 we solve four final AAL1 data constraint specifications for sending data units (see Figure 3). This is due to the fact that with the exception

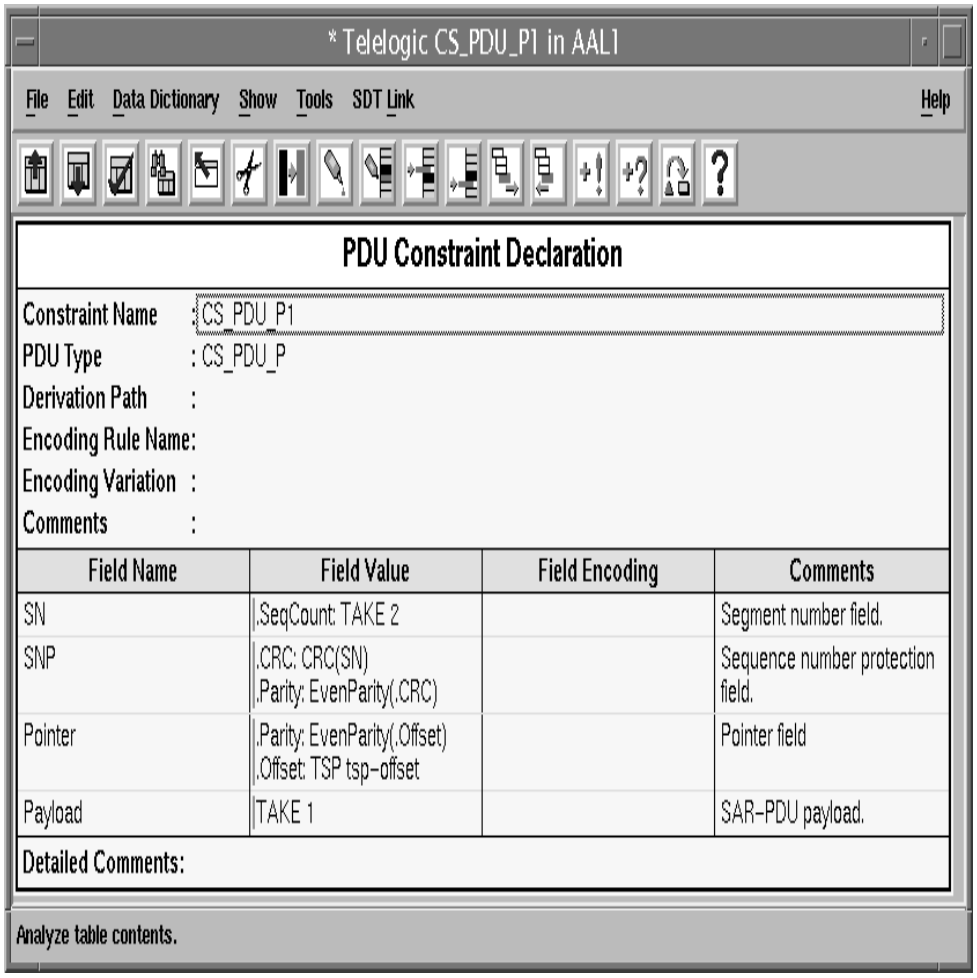

Figure 2: A profile declaration example

of the SN components all the other subfields SNP, Pointer and Payload could take one value only. The corresponding specification is either explicitly for Payload (TAKE 1) and implicitly for the subfield of SNP and Pointer due to the reference of test suite operations (EvenParity, CRC) or the usage of a test suite parameter (tsp-offset). The final value for these subfields depends on the corresponding test suite operation results and the related test suite parameter. But both SN subfields allow two values: SN.CSI has not been restricted, i.e. the subfield type has two values (‘0’B and ‘1’B) and SN.SeqCount is explicitly restricted by TAKE 2.

Due to the usage of TAKE *n* there is no explicit value declaration for SN.SeqCount and Payload, i.e. some values have to be selected at random from a default value list.

Considering the generation of TTCN data constraints for receiving data units we may derive one single constraint declaration from the profile declaration example given in Figure 2. In this context there will be no difference to the sending of data units w.r.t. the usage of test suite operations and test suite parameters. But we will generate ANY constraints (“*”) for data fields with undefined values, i.e. SN.CSI, SN.SeqCount and Payload, instead of using TAKE *n*.

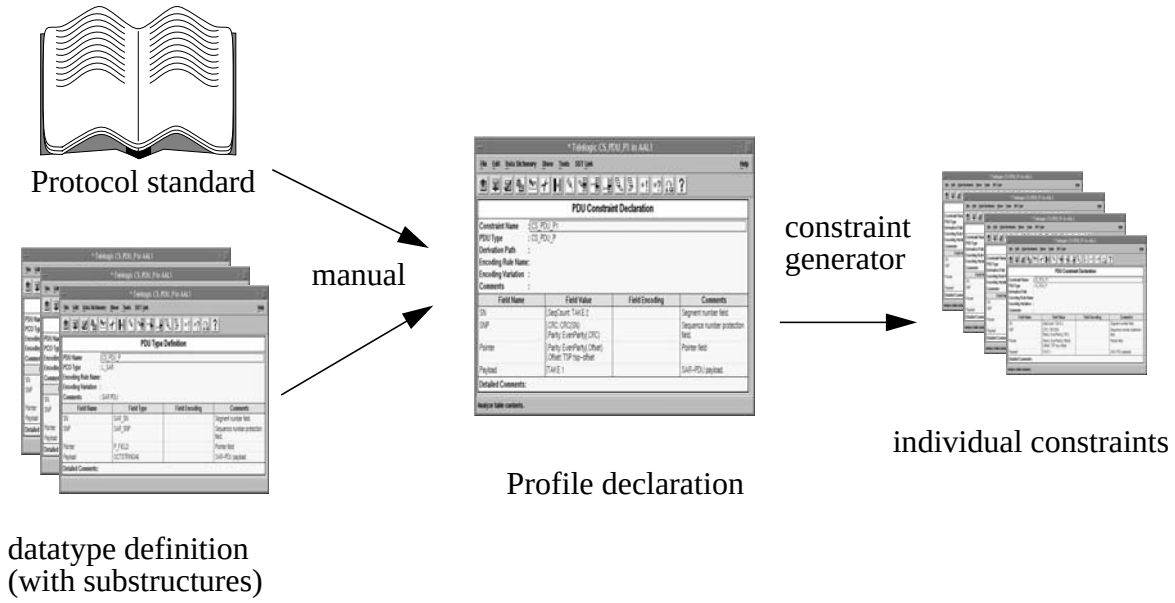

Figure 3: Data constraint generation

4 An interface profile notation

We want to extend a model that considers already continuous data streams and performance characteristics by the approach of specifying data type profiles, respectively test data constraints. Whereas data type profiling is related to the filtering technique applied to streams, data constraining is a value set reduction technique preferably used in testing. The notational approach $Q_{uite}ID_{ea}L$ of specifying "Object Interface Declarations" (the language is a superset of ITU-T's ODL notation) adopts from the TTCN inherent data type specification notation ASN.1 the concept of constraining value sets. This concept has been improved by the approach of filtering profiled data or even profiled behaviour. Hence, profiling is suitable for both, the filtering of specified data or events from a stream and the reduction of possibly infinite data sets or behaviour to practically handable sets.

A declaration of ASN.1 test data type supports - for the purpose of restricting large data sets - the use of specific constraints. These constraints however, are rather primitive. One can explicitly specify the coded pattern of a data value one is looking for or, unrestricted data. This is already similar to a very primitive filter. All the data which does not coincide with the specified constraints, is allowed to pass, respectively will not be tested. Similar, as constraints in terms of ASN.1, a profile constrains the possibly infinite set of data, simply by augmenting the declaration of the data type at an interaction point by the profile. An activated filter can use the profile for several purposes, i.e. either to wait for the occurrence of the profiled data value or to check more sophisticated, conformance rules against the passing data. Filters are also useful for providing statistics, e.g. counting the number of occurrences of specific data values.

For the purpose of testing it would be extremely convenient if these filters could be used by the testing machine for test relevant data observations, instead of annoying generation of test data

values mostly done by hand. Thus, $Q_{uite}ID_{ea}L$ supports the declaration of appropriate profiles and filters at an object's interface. Considered from the point of testing methodology these filters are very tiny testers distributed to interfaces where needed. The filters may interact with a test coordinator which might further be able to program the policy of testing ran by the filters. Where no filter can be implemented at the interfaces, the profiles are still useful to derive limited test data value sets.

In Figure 4 the syntax of a data type specification improved by the notion of 'auxiliary tools' is presented. Auxiliary tools could either apply the old ASN.1 style of specifying constraints to data types, or the improved notation of applying $Q_{uite}ID_{ea}L$ data type profiles. These profiles can be applied by filters plugged-in at an object interface. In order to improve expressiveness to performance constraints, filters might use tools such that as timers or counters in addition to data type profiles for the elaboration of statistical information. A filter may realize a certain policy for the selection of test relevant data, e.g. to achieve a certain coverage of tested data type.

dt_spec ::= PCO send/receive_ indicator pdu_type auxiliary_tools

auxiliary_tools ::= $Q_{uite}ID_{ea}L$_filters | ASN.1_constraint

$Q_{uite}ID_{ea}L$_filters ::= policy(dt_profile [statistic_tools])

dt_profile ::= range | enumeration | expression

ASN.1_constraint ::= coded_value | any_value

Figure 4: Data Type Profiling Concepts

The $Q_{uite}ID_{ea}L$ options for profiling data types are in their effects comparable to the examples applicable to TTCN which are presented in the sections before. With the exceptions of value ranges and enumeration of values, i.e. value list, the various options are represented in terms of expressions of prepositional logics. For example, the constraint specification rule (6) '*Field1: ANY*' of the category *'random selection'* is specified by the $Q_{uite}ID_{ea}L$ profile expression of

$$SN.\text{SeqCount\_profile}\{\ (x \in \text{SeqCount\_type}) \wedge (\#\text{SeqCount} = 1)\ \}.$$

This expression is equivalent to the meaning: "take *any one value* from the valid set of type *SeqCount_type* and assign it to field *SeqCount*". Notice the sign '#' counts the number of selections of values from type *SeqCount_type*". Similar, the conditional selection constraint specification rule (9) is translated into the $Q_{uite}ID_{ea}L$ profile expression of

$$\text{Payload\_profile}\{\ (\text{SeqCount} = 1) \rightarrow (x \in \text{Payload\_type}) \wedge (\#\text{Payload} = \text{n})\ \},$$

which means: "take *some n values* from the valid set of type *Payload_type* and assign them to field *Payload*, provided field *SeqCount* contains the value 1".

The identifiers *SeqCount* and *Payload* are the identifiers of entries in the larger data structure declaration *CS_PDU_P*. This data structure is a parameter of an interface operation. Figure 5

```
interface AAL1 {

   typedef boolean SeqCount_type;
   typedef octet Payload_type;

   profiledef SN.SeqCount_profile {...}
   profiledef Payload_profile {...}

   typedef struct CS_PDU_P { SeqCount_type SeqCount profile SN.SeqCount_profile;
                             Payload_type Payload profile Payload_profile; ...}
   ...}
```

Figure 5: Interface Data Type Declaration with Profiles

contains an example interface specification that includes data type declarations augmented with profiles. The profiles are separated from the basic ODL data type declaration by the keyword 'profile' and are referred to by names. At an additional part of the interface specification the concrete profile declarations are to be specified. In Figure 5, the profile declaration capability has been added to the type declaration clause of ODL. The profile declarations are useful for reducing the amount of unconstrained typed test data. Active filters inserted at object interfaces are parameterized by profiles and can thus control various flows of data, i.e. either to extract them from the flow or to let them pass.

5 Conclusion

The introduction of (test) data profiles to identify subsets for relevant (test) data instead of considering unlimited data value spaces has been proposed. We have defined basic profile rules which are meaningful in the context of test data constraint generation and data stream controlling.

Filters allow to shrink huge amounts of data to a handsome amount. By parameterization filters with flexible profiles, the same filter is able to realize varying testing policies. In order to control or to test distributed systems, filters and data type profiles must be able to be placed at adequate locations. These locations are the interfaces of distributed objects. Consequently, we have improved ITU-T's notation for object declarations ODL by compatible concepts to specify behaviour at "points of control and observations" (PCOs) augmented with filters referring to data type profiles.

Implementing some tool support for TTCN test data constraint generation is expected to speed up the development of basic TTCN test cases. We plan to implement such a prototype of the constraint generator in the next future to identify further needs for the profile declaration rules.

An additional topic in this project is the expansion of the constraint generation approach to performance constraints as defined in PerfTTCN [8] and $Q_{uite}ID_{ea}L$ [4].

6 References

[1] M. Schmitt, A. Ek, B. Koch, J. Grabowski, D. Hogrefe: Autolink - Putting SDL-based test generation into practice. In: A. Petrenko, N. Yevtushenko: IWTCS volume 11, Tomsk (Russia), August 1998.

[2] Telelogic: TAU manuals, version 3.4, Malmö (S) 1998.

[3] M. Li, I. Schieferdecker, A. Rennoch: Testing the TINA retailer reference point. ISADS'99, Tokyo, March 1999.

[4] J. de Meer, A. Rennoch, A. Puder: Towards a QoS binding notation. In: H. König, P. Langendörfer: FBT'98, Cottbus, June 1998.

[5] Q. Tan, A. Petrenko: Test generation for specifications modelled by I/O automata. In: A. Petrenko, N. Yevtushenko: IWTCS volume 11, Tomsk (Russia), August 1998.

[6] ITU-T: B-ISDN ATM Adaptation Layer specification: Type 1 AAL, Recommendation I.363.1, 1996.

[7] ISO: Conformance Testing Methodology Framework, ISO 9646, 1992.

[8] I. Schieferdecker, B. Stepien, A. Rennoch: PerfTTCN, a language extension for performance testing. In: M. Kim, S. Kang, K. Hong: IWTCS volume 10, Cheju Island (Korea), September 1997.