Test Data Variance as a Test Quality Measure: Exemplified for TTCN-3

Diana Vega[1], Ina Schieferdecker[1,2], George Din[2]

[1] Technical University Berlin, Franklinstr. 28/29, D-10623 Berlin,
{vega,ina}@cs.tu-berlin.de**
[2] Fraunhofer FOKUS, Kaiserin-Augusta-Allee 31, D-10589 Berlin,
{schieferdecker,din}@fokus.fraunhofer.de

Abstract Test effectiveness is a central quality aspect of a test specification which reflects its ability to demonstrate system quality levels and to discover system faults. A well-known approach for its estimatation is to determine coverage metrics for the system code or system model. However, often these are not available as such but the system interface only, which basically define structural aspects of the stimuli and responses to the system.
Therefore, this paper focuses on the idea of using test data variance analysis as another analytical approach to determine test quality. It presents a method for the quantitative evaluation of structural and semantical variance of test data. Test variance is defined as the test data distribution over the system interface data domain. It is expected that the more the test data varies, the better the system is tested by a given test suite. The paper instantiates this method for black-box test specifications written in TTCN-3 and the structural analysis of send templates. Distance metrics and similarity relations are used to determine the data variance.

1 Introduction

Today's test specifications used in industry and for standardised test suites are usually complex (several hundred test cases, several thousand lines of test code, etc.). As they are hard to evaluate and assess, test quality aspects are constantly subject of discussions. Various test metrics have been developed already measuring selected aspects [1,2,3]. Therefore [4] provided a framework for the different quality aspects of test specifications: it proposed a quality model for test specifications based on the ISO/IEC 9126 [5] quality model. The concrete quality analysis for *Testing and Test Control Notation* (TTCN-3) test suites however concentrated on internal quality aspects only — to analyse potentials of test suite reuse and maintenance.

However, TTCN-3 [6] being standardized by *European Telecommunications Standards Institute* (ETSI) allows not only to specify tests abstractly, but also to make them executable by compilation and execution together with additional

** This work has been supported by the *Krupp von Bohlen und Halbach - Stiftung.*

run-time components (such as an *SUT adapter*). By that not only the internal, but also the external quality is of interest.

Test effectiveness is the external quality aspect of a test specification which reflects its ability to demonstrate system quality levels and to discover system faults — in other words, its ability to fulfill a given set of test purposes. According to [4], test effectiveness is divided into

- the *suitability* aspect which is characterised by *test coverage*. Coverage constitutes a measure for test completeness and can be measured on different levels, e.g. the degree to which the test specification covers system requirements, system model, system code and alike,
- the *test correctness* aspect which reflects the correctness of a test specification with respect to the system specification or the set of test purposes, and
- finally the *fault-revealing capability* on the capability of a test specification to actually reveal faults.

In practice, both system model and system code are not always available to the testers, for example when testing third-party components, integrated with off-the-shelf components or tested on system and acceptance level. Hence, the test correctness and fault-revealing capabilities are hard and if not impossible to determine. In contrast, system interfaces as such are available (often also provided in terms of interface specifications and/or documentations) test coverage for the system interfaces could be analysed — despite the fact, that a more thorough analysis would be possible if more information in terms of system model and system code would be available. In the latter case, for white-box (structural) testing code coverage metrics and for black-box (functional) testing system model coverage metrics are in use. Traditionally, code metrics have been used only. With the advances of model-based system development, system model coverage metrics have been adapted from code coverage metrics by using state coverage (the counterpart for statement coverage), transition coverage (the counterpart for branch coverage) and alike.

In this paper we investigate the typical, but less comfortable situation where only the system interface is given. The system interface defines the input data to the system (the stimuli) and the output data to the system (the reactions). These are either provided in form of data structures (e.g. accessing the system information directly), of messages (e.g. accessing the system via asynchronous communication means) or by means of operations (e.g. accessing the system via synchronous operation invocations).

This paper describes an approach to analyse system interface coverage by means of test data variance. It is expected that the more the test data variates, the better the system is tested by the given test suite. We concentrate on asynchronous test data only by analysing type and send value templates only[3].

[3] Without loss of generality, we do not consider signatures and signature templates. The approach however can be extended to handle the case of synchronous communication.

Please note that although we consider external test quality, we are using analysis methods that do not execute the test suite itself. We use such an approach of analysing the abstract test suite itself as every test execution involves also the SUT. Hence because of the SUT capabilities and quality, a test execution may reveal selected errors only, may allow to execute a small subset of test cases only and alike - although the test suite might principally be able to reveal more errors. Therefore, we are aiming at determining the *error revealing potential* of a test suite instead.

The paper is structured as follows: after reviewing related work in Section 2, the principal approach is explained in Section 3 and the data variance computation method is presented in Section 4. Distance metrics for TTCN-3 types are discussed in Section 5 and further aspects of test data variance are discussed in Section 6. An example is given in Section 7 and details of our implementation are highlighted in Section 8. Concluding remarks in Section 9 complete the paper.

2 Related work

Independent of a test specification language, the *test data adequacy criterion* remains among the most important factors for the effectiveness of a test. According to [7], a test data adequacy criterion is a formalism to decide if a software has been tested enough by test data. The same author introduces in [8] a theoretical model for the notion of adequacy in the context of test effectiveness. Many other test adequacy criteria are defined in the literature [9] such as the well-known control-flow criteria, but also more complicated ones such as the Modified Condition/Decision Coverage (MC/DC) or Reinforced Condition/Decision Coverage (RC/DC) criteria introduced by Kapoor [10].

Approaches to study how good the test data is selected include the notion of *distance* between programs, where programs are the tested systems. A test data set is considered to be *adequate* if it distinguishes the tested program from other programs which are *sufficiently far* from it, i.e. produce different input-output behavior. Close programs producing same results are considered the same [8]. A similar concept of *Adaptive Random Testing* is provided in [11] where random test inputs rely on the notion of *distance* between the test values. The authors define the object distance in the context of object-oriented programs. In addition, they propose a method to compute the distance and use it to generalize their approach.

Another approach of test data coverage called *statistical coverage* is presented in [12]. The concept of statistical coverage derives from statistical testing and requires continues testing until it is unlikely that new coverage items will appear. The proposed statistical coverage method uses a binomial distribution to compute the probability of finding new coverage item and an associated confidence interval, with the assumption that software test runs are independent of each other.

All these approaches intend to study *test data variance* as a measure of test data values spread over the input domain. Given the very large number of

possible inputs (e.g. almost all types have a theoretically unlimited number of values) that could be feed to a program to be tested, the goal is to minimize the number of test cases (in software testing, test data applied to the same system behaviour) in a test suite while keeping test effectiveness as high as possible.

3 The principal approach

The basic idea for test data variance of black-box tests is to analyse the coverage of test inputs with respect to the system interface and its structure as depicted in Figure 1.

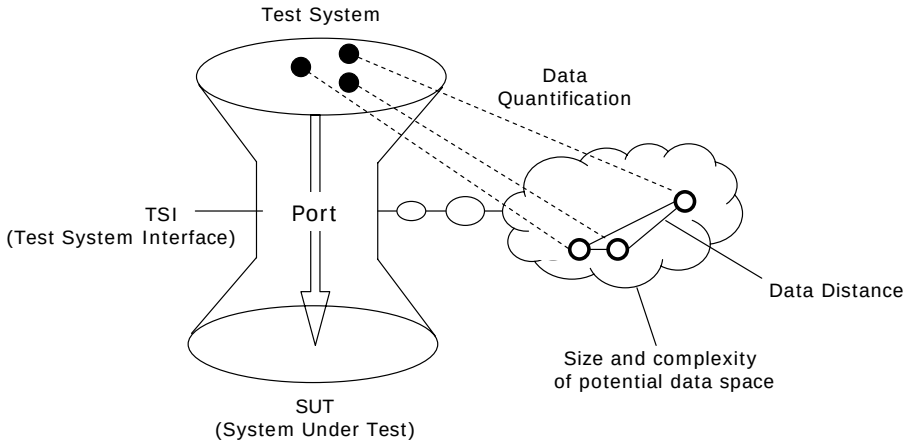

Figure 1. The principle test data variance approach

In order to get a conceptual framework for presenting our approach, we use TTCN-3 terminology [6]. The *system under test (SUT)* is represented with its interfaces and in relation to the test system only: the *test system interface (TSI)* consists of a number of *ports*. Every port can be of different *port type*. A port type defines the kind of a port to be *message-based* or *signature-based* and to be *unidirectional* or *bidirectional*. For every *test case*, the TSI is defined (explicitly or implicitly) within the *system* clause. By that, one test suite represented by a (set of) TTCN-3 modules can test different TSIs (even of potentially different SUTs, although that is not recommended).

A high test coverage with respect to a given TSI requires that the test data has to fulfill the following criteria:

- Every port and every type transportable to the SUT via that port (incl. every type element in case of structured types) have to be "touched" by the test data.

– The test data has to be representative with respect to a given type. Representative data can be identified either semantically or structurally, i.e. data can vary with respect to qualitative or quantitative similarity.

While quantitative similarity is by use of distance measures easier to derive, qualitative similarity is assumed to provide better results. In particular, the partitioning method can be used to provide a qualitative characterization of an input space. In this paper, we provide a computation method open to both the qualitative and quantitative similarity of test data (see Section 4), but concentrate later on quantitative similarity for TTCN-3 (see Section 5) only as this can be derived purely from the TSI. For the sake of simplicity, we assume in the following

– that all test cases have the *same TSI*,
– that the TSI consists of *one port* only,
– that this port is a *message-based* port, and
– which can transport *data to the SUT*[4].

4 The data variance computation method

Let us assume a set of test data of a given type being sent over a given port to the SUT, from which we select two (i.e. two TTCN-3 templates resolving to concrete values). Their distance is calculated by use of type specific *distance* metrics. By use of a type specific *distance threshold*, the similarity or dissimilarity of the test data is being determined. The type coverage is finally determined by the number of subsets of similar test data, meaning that a set of dissimilar test data covers a type better than one with similar data.

For that, we consider basically *types* T. As in structured types however elements/fields can be optional, a type extended with omit are being considered in those cases:

$$T' = T \cup \{omit\}$$

Without loss of generality, we consider subsequently T' for types. We partition types into subtypes by considering a value v of T' and the set of values being logically or numerically nearby v:

$$partition_{T'} : T' \rightarrow \Pi_{T'}$$
$$\text{for } v \neq omit : partition_{T'}(v) = \{v' \in T : similar(v, v')\}$$
$$partition_{T'}(omit) = \{omit\}$$

The *similarity* of values is a Boolean relation which can be used to determine qualitative or quantitative similarity. For the moment, we restrict ourselves to

[4] This is in fact not a limitation, but a precondition that test data can be sent to the SUT via that port

quantifiable similarity. For that, we use the *distance* between values and a *distance threshold* so that any two values are considered similar whenever their distance is smaller than the distance threshold[5]:

$similar_{T'} : T' \times T' \rightarrow \mathcal{B}$
for $v_1, v_2 \neq omit : similar_{T'}(v_1, v_2) = \{ {}^{true \text{ for } distance_{T'}(v_1,v_2) < threshold_{T'}}_{false \text{ otherwise}}$
$similar_{T'}(v, omit) = \{ {}^{true \text{ for } v=omit}_{false \text{ otherwise}}$

The distance of a type is defined as a float value in between 0 and 1:

$distance_{T'} : T' \times T' \rightarrow [0..1]$
for $v_1, v_2 \neq omit : distance_{T'}(v_1, v_2) = \{ {}^{>0 \text{ as defined in Section 5 for } v_1 \neq v_2}_{0 \text{ otherwise}}$
$distance_{T'}(v, omit) = \{ {}^{0 \text{ for } v=omit}_{1 \text{ otherwise}}$

Finally, we determine the *coverage* of a type *T'* for a given set of values of that type:

$coverage_{T'} : \Pi_{T'} \rightarrow [0..1]$
$coverage_{T'}(V) = \#partitions_{T'}(V) \star threshold_{T'}$

where the *number of partitions* determines the number of varying data of a value set for a given type:

$\#partitions_{T'} : \Pi_{T'} \rightarrow \mathcal{N}$
$\#partitions_{T'}(\emptyset) = 0$
for $V \neq \emptyset : \#partitions_{T'}(V) = 1 + \#partitions_{T'}(V')$
with $V' = V \setminus partition_{T'}(v)$ for a selected $v \in V$

That completes the data variance computation method, which allows us to determine type coverage based on qualitative or quantitative notion of test data variance.

5 Distance metrics for TTCN-3 values

This section defines the distance measures to derive the quantitative similarity of TTCN-3 values being sent to the SUT. The TTCN-3 type system consists of *basic* and *structured* types. Their distance definitions are given in Table 1 and Table 2[6]. Please remember that the distance for *omit* has been already defined in Section 4: it is maximum, i.e. 1, between *omit* and any other concrete value and minimum, i.e. 0, between *omit* and *omit*.

[5] Please note that this gives us a dynamic classification of values into sets of similar values — depending on the chosen values, the set of values considered similar will differ. This is different to the qualitative approach of equivalence classes [13], where equivalence classes (also representing data partitions) are statically defined.

[6] We left out the objid type as it is often used together with ASN.1 specifications only.

Table 1. Distance Metrics for Values of Basic TTCN-3 Types

Basic Type	Distance based on	Definition of distance d for values x and y
Integer	One-Dimensional Euclidian Distance	$d(x,y) = \frac{\lvert x-y \rvert}{sizeof(Integer)}$
Float	One-Dimensional Euclidian Distance	$d(x,y) = \frac{\lvert x-y \rvert}{sizeof(Float)}$
Boolean	Inequality	$d(x,y) = \{ {}^{0 \text{ for } x=y}_{1 \text{ otherwise}}$
Bitstring	Hamming Distance	number of positions for which the bits are different (the shorter bitstring is extended into the longer bitstring by filling it with leading '0'B) divided by the longer length: $d(x,y) = \frac{\mathfrak{d}(x,y)}{maxlength(x,y)}$ with $\mathfrak{d}(x,y)$ = number of i where $x_i \neq y_i$
Hexstring	Hamming Distance	same but with leading '0'H
Octetstring	Hamming Distance	same but with leading '0'O
Charstring	Hamming Distance	same but with leading " " (spaces)
Universal Charstring	Hamming Distance	same but with leading " " (spaces)

Table 2. Distance Metrics for Values of Structured TTCN-3 Types

Structured Type	Distance based on	Definition of distance d for values x and y
Record	N-Dimensional Euclidian Distance	$d(x,y) = \frac{\sqrt{\sum_{i=1}^{n}(d(x_i,y_i))^2}}{n}$
Record of	Hamming Distance	$d(x,y) = \frac{\sum_{i=1}^{n}\mathfrak{d}(x,y)}{maxlength(x,y)}$ with $\mathfrak{d}(x,y)$ = number of i where $d(x_i,y_i) > \frac{1}{3}$ and where the record sequence is extended into the longer record sequence by filling it with leading *omit*
Set	N-Dimensional Euclidian Distance	same as for record
Set of	Hamming Distance	same as for record of
Enumerated	Inequality	$d(x,y) = \frac{\lvert \mathfrak{n}(x)-\mathfrak{n}(y) \rvert}{n}$ where $\mathfrak{n}$ is the sequentially numbered index of the enumeration
Union	Distance defined above	$d(x,y) = d(\mathfrak{v}(x),\mathfrak{v}(y)) = \{ {}^{1 \text{ for } \mathfrak{v}(x)=\mathfrak{v}(y)}_{0 \text{ otherwise}}$

As defined in Section 4, every type T has an associated $threshold_T$, which is the basis to determine data similarity out of the data distance. In spite of our pure quantitative data analysis, our analysis of selected test suites (i.e. for *Session Initiation Protocol* (SIP), *IP Multimedia Subsystem* (IMS), *SS7 MTP3-User Adaptation Layer* (M3UA) and *Internet Protocol version 6* (IPv6)) indicated that a uniform threshold of $\frac{1}{3}$ is a good basis for representing the data variance requirements: $\frac{1}{3}$ means that there should be three representative values such as from the "beginning", "middle" and "end" of a type. Only for the case of *Boolean* and two-value enumerations this threshold should even be reduced to $\frac{1}{2}$. However note that, if we consider an optional field of these types (and hence T' instead) we take $threshold_{T'} = \frac{1}{3}$ as in this case *omit* constitutes an own similarity class of the data being sent to the SUT.

6 Distance metrics for TTCN-3 templates

In general, test data to be analysed with respect to their type coverage are not just concrete values, but templates that are sent over the same port to the SUT. These templates constitute a *template subset* to be considered, where the following aspects complicate the analysis:

- *global and local templates*: Templates can be defined in global or local scope. For the latter case, a call flow analysis would be adequate in order to derive the template subset precisely. For the moment, we analyse all templates independently of their scope.
- *template parameters*: Template fields may use parameters directly or parameters within more complicated expressions. In both cases, a symbolic analysis is needed to derive the limitations for the template fields. For the moment, we use for parameterized fields the maximum distance as they have the potential to spread the field type completely.
- *modified templates*: Template modifications are used to change the setting of template fields of global or local templates. The updated field can be defined in terms of concrete values or more complex expressions, which may reference parameters, functions and alike. Currently, we use distances for concrete values only and a maximum distance in all other cases.
- *inline templates*: In this case, the send template is formed directly in the send statement where the values may take any form of expression such as function calls, variables reference and alike. A precise analysis of inline templates requires a combination of call flow analysis with symbolic computation. As for modified templates, we use distances for concrete values only and a maximum distance in all other cases.

These template aspects make the analysis of data variance tricky and demonstrate why the quality of a real TTCN-3 test suite is hard to assess. Our current solution overestimates the coverage of a test suite. However, as tool development is progressing the provided measures will become more and more precise.

7 An Example

In this section, we show how to apply the introduced concepts to small TTCN-3 examples. In the listing below we define a simple TTCN-3 record type R that contains two fields: one of type integer of range (1..100) and an optional boolean field. Based on this type definition, several templates are defined $r1, r2 \ldots, r4$.

Listing 1.1. TTCN-3 Example

```
                                                                    1
type record R {
        integer i (1..100),
        boolean b optional
}
                                                                    6
template R r1:= {1, true}
template R r2:= {10,true}
template R r3:= {90,omit}
template R r4:= {35,false}
```

Assuming that all these templates are used as SUT stimuli over a port that carries R, they form a template subset of interest. The next step is to determine each distance $d(r_i, r_j), i \neq j$ — recursively for the fields as given in Table 3 and Table 4 according to the formulas given in Table 1 and then for the complete record (see Table 5) according to the formulas in Table 2.

Table 3. Distances for boolean record field in the example

	true	false	omit
true	0	0,5	1
false	0,5	0	1
omit	1	1	0

Table 4. Distances for integer record field in the example

	1	10	90	35
1	0	0,09	0,89	0,34
10	0,09	0	0,8	0,25
90	0,89	0,8	0	0,45
35	0,34	0,25	0,45	0

We see that the fields themselves are well covered (and, indeed, we have seen this immediately because of the simplicity of the example). Looking however at the records in Table 5, it shows that R is not well covered.

Table 5. Distances for records in the example

	r1	r2	r3	r4
r1	0	0,05	0,67	0,3
r2	0,05	0	0,64	0,28
r3	0,67	0,64	0	0,57
r4	0,3	0,28	0,57	0

The records $r1$, $r2$ and $r4$ are similar (and are hence considered stimulating the same system behavior — they are considered belonging to the same similarity class). The separation of the templates into similarity classes is made by comparing the distances between them and a threshold value. The threshold of $\frac{1}{3}$ separates R into three similarity classes: r1,r2 and r4 form one similarity class, r3 a second, but the third is missing. It is not so obvious that although the field types are covered, the record type R is not. The situation can be resolved by adding e.g. $r5$ which represents the third similarity class. The distances between $r5$ and the other templates are computed in Table 6.

Table 6. Distances for added record

	r1	r2	r3	r4
r5	0,49	0,44	0,5	0,41

Listing 1.2. Extended TTCN-3 Example

```
:
template R r1:= {1, true}
template R r2:= {10,true}
template R r3:= {90,omit}                                    4
template R r4:= {35,false}
template R r5:= {99,true} // added to cover R completely
```

Whenever representatives for similarity classes are missing, approaches for test refactoring [14] and/or pattern-driven test generation [15] can help here to improve the quality of a test suite. Once the analysis has shown that selected interface aspects are not covered, additional templates (to be sent by additional test cases) could be proposed for inclusion into the test suite.

8 Implementation

In order to compute the test data variance and system interface coverage, there is a clear requirement for a TTCN-3 tool to automatically compute the variance measures.

Our implementation is based on the TTworkbench [16] product, an Eclipse-based IDE that offers an environment for specifying and executing TTCN-3 tests. The main reason for selecting this tool, is that it provides a metamodel for the TTCN-3 language which is technically realized by using the *Eclipse Modelling Framework* (EMF) provided by Eclipse. EMF is a Java framework and code generation facility which helps turning models rapidly into efficient, correct, and easily customizable Java code.

The generated Java classes provide an interface useful to traverse every TTCN-3 test suite loaded and access every element of it by creating an associated metamodel instance. The plug-in based structure of the tool allows adding new features by plugging them into the core platform. The incorporated TTCN-3 metamodel is a central test repository that can be used to present the test suite in various formats: in the core language format (the CLEditor is used to edit TTCN-3 code in its textual format) or in the graphical format (for which a GFT Editor is provided).

Our work on the automated template distance collector follows up an earlier work [2] where TTCN-3 test quality indicators are derivedfrom a statical analysis of a TTCN-3 test suite. It is designed as a plug-in whose invocation triggers a) the access to the metamodel instance and b) the traversal of elements of interest as shown in Figure 2. The most significant steps are:

- visit test cases
- identify templates used in send operations
- recursively traverse pairs of templates in order to measure their distance

In the tool terminology, a TTCN-3 project contains all TTCN-3 modules composing an overall test suite. Given a project identifier, the metamodel loader engine is able to load all modules and build a tree-like structure having as root the main module. The code snippet in Figure 3 shows how to extract the *runs on* component name from each test case declaration using the provided EMF API. This one will be used furthermore to find its definition and extract the list of ports used in send direction.

Every TTCN-3 element has a corresponding EMF element that could be accessed and modified handling only EMF generated Java classes. While searching for simple EMF element definitions translates into accessing directly the tree nodes and getting the needed information, obtaining in parallel the values of two templates whose distance is to be measured, introduces a much more increased degree of complexity. For example, for structured type based templates, it is required to design a visitor that traverses recursively and in parallel every child from each template until values in leaves are reached. Then, the distance formula for templates of basic types is applied to the leaves belonging to the same level in the tree hierarchy and returned to the upper level in a recursive process.

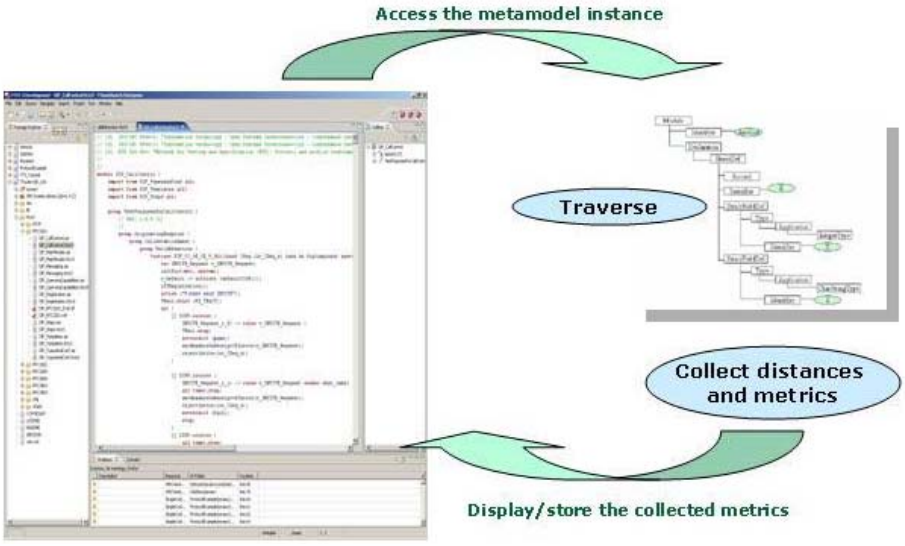

Figure 2. The principle of the implementation

```
Iterator it = testcaseDeclarationListPerModule.iterator();
while(it.hasNext()) {
    mev = (MuTTCNEMFModuleElementView) it.next();

    ValueInterpreter valueInterpreter = ValueInterpreter.create(this.repository.getId(),
                                                                (Element)mev.getObject());
    if (valueInterpreter.getInput() instanceof ConstantDeclarationImpl)
    {
        ConstantDeclarationImpl currentTestcase =
                                (ConstantDeclarationImpl)valueInterpreter.getInput();
        if (currentTestcase.getTheType() instanceof FunctionTypeImpl ){
            FunctionTypeImpl currentTestcaseTheType =
                            (FunctionTypeImpl)currentTestcase.getTheType();
            if ( currentTestcaseTheType.getTheToType() instanceof TestcaseBehaviorTypeImpl){
                TestcaseBehaviorTypeImpl testcaseBehaviorTypeImpl =
                                (TestcaseBehaviorTypeImpl)currentTestcaseTheType.getTheToType();
                String runsOnComponentName =
                       testcaseBehaviorTypeImpl.getTheFromType().getTheName().getTheName();

                if(runsOnComponentName.equals(component))
                    testcasesRunsOnComponentList.add(valueInterpreter);
            }
        }
    }
}
```

Figure 3. Metamodel traversal - code snippet

9 Conclusions and outlook

In this paper we investigate test data variance as a way to assess the test coverage for the system interface quantitatively. We and others consider test data variance as an import factor of test effectiveness. This paper defines a *principal method for deriving test data variance* based on notions of qualitative or quantitative data similarity. This method is then exemplified for TTCN-3 and for *quantitative data similarity*.

We define distance metrics for basic and structured types. A threshold based weighting process of distance evaluation leads to an empirical assessment of data similarity: it is false when the values are different "enough". Different values are counted into separate *similarity classes* representing partitions of a given type where similar values belong to the same partition. The number of present similarity classes of a type in a test suite defines finally the coverage for that type. With the aggregation of the coverage of all stimuli types, we obtain the overall test suite coverage.

Although the approach is in its beginning, it demonstrated already the value of coverage analysis for system interfaces. In future work, the empirical analysis will become more precise with the addition of a dedicated call flow analysis and symbolic execution. In addition, the simplifying assumptions given in Section 3 for defining the data variance computation method are easy to leverage and are being leveraged already in our tool

- by slicing a test suite into sets of test cases with the same test system interface (TSI) and considering the TSIs individually,
- by considering every port of a TSI individually, and
- by expanding the notion of distance, similarity and coverage to signatures and their parameters,

Finally we plan to detail the system interface analysis by extending toward semantical aspects like condition, assertions and behaviours of the interface usage. By all that, we foresee an application of the test data variance analysis in test generation approaches as a control and stopping criteria.

References

1. Sneed, H.M.: Measuring the Effectiveness of Software Testing. In Beydeda, S., Gruhn, V., Mayer, J., Reussner, R., Schweiggert, F., eds.: Proceedings of SOQUA 2004 and TECOS 2004. Volume 58 of Lecture Notes in Informatics (LNI)., Gesellschaft für Informatik (2004)
2. Vega, D.E., Schieferdecker, I.: Towards quality of TTCN-3 tests. In: Proceedings of SAM'06 – Fifth Workshop on System Analysis and Modelling (formerly SDL and MSC Workshop), May 31st-June 2nd 2006, University of Kaiserslautern, Kaiserslautern, Germany, 2006. (2006)
3. Zeiss, B., Neukirchen, H., Grabowski, J., Evans, D., Baker, P.: Refactoring and Metrics for TTCN-3 Test Suites. In Gotzhein, R., Reed, R., eds.: System Analysis and Modeling: Language Profiles. Volume 4320 of Lecture Notes in Computer Science., Springer (2006)

4. Zeiß, B., Vega, D., Schieferdecker, I., Neukirchen, H., Grabowski, J.: Applying the ISO 9126 Quality Model to Test Specifications Exemplified for TTCN-3 Test Specifications. In: Software Engineering 2007 (SE 2007). Lecture Notes in Informatics (LNI). Copyright Gesellschaft für Informatik, Köllen Verlag, Bonn (2007)
5. ISO/IEC: ISO/IEC Standard No. 9126: Software engineering – Product quality; Parts 1–4. International Organization for Standardization (ISO) / International Electrotechnical Commission (IEC), Geneva, Switzerland (2001-2004)
6. ETSI: ETSI Standard ES 201 873-1 V3.2.1 (2007-03): The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis, France (2007)
7. Weyuker, E.J.: The evaluation of program-based software test data adequacy criteria. Commun. ACM **31** (1988) 668–675
8. Davis, M., Weyuker, E.: Metric space-based test-base adequacy criteria. Comput. J. **31** (1988) 17–24
9. Weiss, S.N.: Comparing test data adequacy criteria. SIGSOFT Softw. Eng. Notes **14** (1989) 42–49
10. Vilkomir, S.A., Bowen, J.P.: Reinforced condition/decision coverage (RC/DC): A new criterion for software testing. In: ZB. (2002) 291–308
11. Ciupa, I., Leitner, A., Oriol, M., Meyer, B.: Object distance and its application to adaptive random testing of object-oriented programs. In: RT '06: Proceedings of the 1st international workshop on Random testing, New York, NY, USA, ACM Press (2006) 55–63
12. Howden, W.E.: Systems testing and statistical test data coverage. In: COMPSAC '97: Proceedings of the 21st International Computer Software and Applications Conference, Washington, DC, USA, IEEE Computer Society (1997) 500–504
13. Grochtmann, M., Grimm, K.: Classification trees for partition testing. Software Testing, Verification and Reliability **3** (1993) 63–82
14. Zeiss, B., Neukirchen, H., Grabowski, J., Evans, D., Baker, P.: Refactoring for TTCN-3 Test Suites. In: Proceedings of SAM'06: Fifth Workshop on System Analysis and Modelling, May 31–June 2, 2006, University of Kaiserslautern, Germany. (2006)
15. Vouffo-Feudjio, A., Schieferdecker, I.: Test patterns with TTCN-3. In Grabowski, J., Nielsen, B., eds.: FATES. Volume 3395 of Lecture Notes in Computer Science., Springer (2004) 170–179
16. TestingTechnologies: TTworkbench: an Eclipse based TTCN-3 IDE. www.testingtech.de/products/ttwb_intro.php (2007)