

Automatic Test Data Generation for TTCN-3 using CTE

Zhen Ru Dai, Peter H. Deussen,

Maik Busch, Laurette Pianta Lacmene, Titus Ngwangwen

*FraunhoferInstitute for Open Communication Systems (FOKUS)
Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany*

Jens Herrmann, Michael Schmidt

*DaimlerChrysler AG, Research and Technology
Alt-Moabit 96A, 10559 Berlin, Germany*

Abstract: With the growing system complexity the need for solid testing increases. The *Test and Testing Control Notation, version 3* (TTCN-3) is a standardized testing language to ease the specification of test suites. Test data play an important role during test execution in order to explore all aspects of the system under test. Although TTCN-3 supports good means for test data specification such as an extensive type system, automated test data generation is not in the scope of the TTCN-3 language and requires complementary means. A promising approach is the *classification tree method* (CTM), which allows for test data generation by categorizing data domains in equivalence classes. This paper introduces an *Eclipse plugin* which enables the integration between TTCN-3 and *Classification Tree Editor* (CTE).

Keywords: *Testing*, TTCN-3, test data generation, CTM, CTE, Eclipse

1. INTRODUCTION

Black-box testing in a nutshell might be defined as the evaluation of the outputs of a *System under Test* (SUT) by a test system in response to stimuli sent by this test system. The term “test data” refers to a selection of those stimuli and responses. There are various data-oriented tools for generation of test data. In this paper, we consider how the *Classification Tree Method* (CTM) [2], which bases on a partitioning of the data domain by means of data classifications. CTM is used in industrial application such as in the automotive domain. An implementation of the CTM in form of a *Classification Tree Editor* (CTE) [1] has been developed by DaimlerCrysler. The approach introduced in this paper is to use CTE to generate input data for abstract test cases expressed in TTCN-3.

TTCN-3 provides means to specify more complex tests than the classification tree can do, since the classification tree only describes test data which are used for the test, but neither relevant aspects like test configuration or test behaviour. However, being a language, TTCN-3 does not provide a systematic approach to define test data.

Classification trees can be viewed as a systematic method to divide a set of data items into certain equivalence classes according to various attributes (so-called classifications). If related to testing, the main idea is that the *system under test* (SUT) behaves in the same way for each member of a specific equivalence class. Thus it is not necessary to use all members of this class for testing, it suffices to test with just one representative. Good data classifications thus may lead to a significant reduction of the number of tests that have to be performed.

In this paper, we describe three different approaches to make the classification tree method available for data classifications in TTCN-3. The *type-oriented approach* departs from a TTCN-3 type definition (e.g. a record structure). CTE is used to derive a partitioning of the data set define by this type. The resulting equivalence classes are represented as TTCN-3 data templates. The *test case oriented approach* concentrates not on a single data type but on test behaviour, i.e. a sequence of data sent to and received from the SUT. For technical reasons, this behaviour is formalized as a TTCN-3 test case or a test function. A combination of the type-oriented and the test case oriented approach is the *template-oriented approach*, which uses TTCN-3 templates directly as data classes. The goal is to produce several versions of the same test behaviour with different combinations of test data. To prove the applicability of the approach, an implementation has been done on the basis of the TWorkbench plugin set for TTCN-3 [6] available for the Eclipse Integrated Development Environment [3].

The paper is organized as follows. Section 2 summarized the Classification Tree Method and introduces briefly the Classification Tree Editor. Section 3 addresses TTCN-3. The three different integration approaches are discusses in Section 4. Section 5 deals with our implementation. Section 6 summarizes the paper and discusses briefly open issues.

2. THE CLASSIFICATION TREE METHOD CTM AND ITS EDITOR

The classification-tree method CTM is a special approach for data generation for black-box oriented functional testing. By means of the CTM, the input and output domains of a potential SUT are regarded under various aspects assessed as relevant for the test. For each aspect, disjoint and complete classifications are formed, which yield a partitioning of the respective domains into equivalence classes. The underlying idea is known as the *test hypothesis* which says that the SUT behaves in the same way for all equivalent stimuli, or, more precisely:

*For each sequence of stimuli that belongs to a certain input equivalence class, the SUT will produce a sequence of responses that belongs to a single output class and moreover, the verdict (**pass** or **fail**) assigned by the test system is the same for each response sequence of the SUT that belongs to that output class.*

Note that the test hypothesis makes an assumption about the way the SUT perceives its input data. As a consequence, the stimuli / response sequences that belong to the same combination of input / output classes may be considered as *equivalent* in the sense that it is not necessary to use the complete input data domain for testing but it suffices to test for a single representative of the input class.

This partitioning of the input and output data domains by means of classifications is represented graphically in the form of a tree, allowing for recursive refinement of classifications. Subsequently, test cases are formed by combining classes of different classifications.

In practice, this is done by using the tree as the head of a combination table in which the test cases are marked. When using the CTM, the most important source of information for the tester is the functional specification of a potential SUT. A major advantage of the CTM is that it turns test case design into a process comprising several structured and systematized parts - making it easy to handle, understandable and also documentable. Figure 1 shows an example of a resulting classification tree.

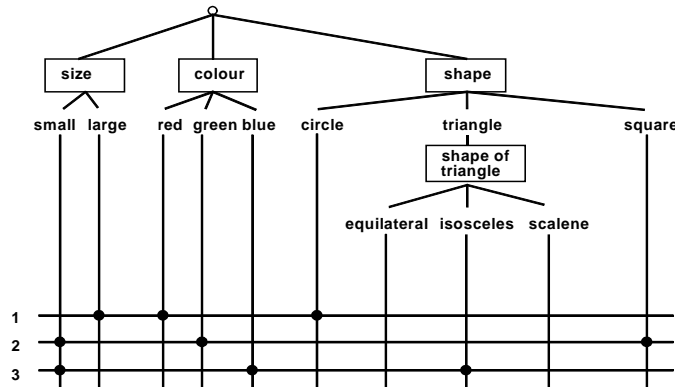


Figure 1: A Classification Tree

The classification-tree editor CTE supports the CTM. The two main phases of the CTM - design of a classification tree and definition of test cases in the table - are both supported by the tool. For each phase a suitable working area is provided. Additionally, CTE enables automatic data selection by using Boolean operators: if S_1 and S_2 are classes (like *size* and *colour* in the above example), then the expression $S_1 * S_2$ produces all combinations of representatives of these classes (i.e. the whole set $\{small, large\} \times \{red, green, blue\}$), while $S_1 + S_2$ yields a (randomly selected) representative of this combination. Figure 2 shows a screenshot of the classification tree editor. The CTE offers features which allow large-scale classification trees to be structured in order to support the test case design for large testing problems efficiently. CTE has been developed by DaimlerChrysler Research Berlin [1].

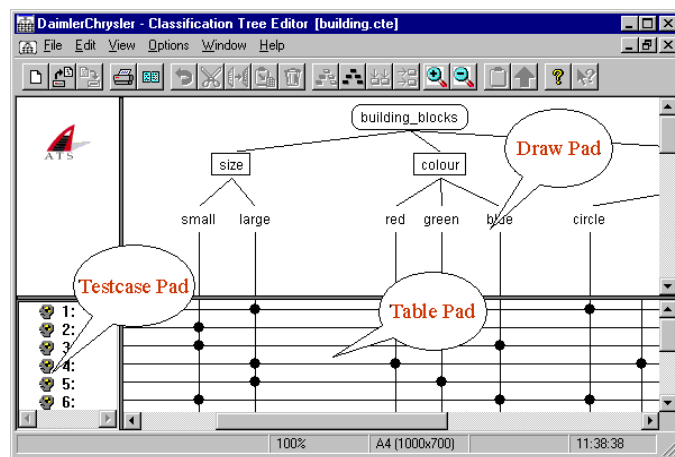


Figure 2: The Classification Tree Editor

3. TESTING AND TEST CONTROL NOTATION

TTCN-3 [4] has been developed at the European Telecommunication Institute (ETSI) as a standardized notation for test description and test system implementation. TTCN-3 is a testing language enriched with various concepts for black-box testing of reactive systems: Synchronous as well as asynchronous communication

primitives, dynamic distributed test configurations by means of parallel test components, verdict handling, modules and test groups, timers, and – for particular interest of the topic addressed in this paper – extensive structured type system built up from basic types like strings, integers, floats, enumerations, sub-ranges, etc. by means of records, arrays, sets, etc. Additionally, data templates of various data types can be defined and used for pattern matching to select alternative continuations of a test run.

4. CTE INTEGRATION APPROACHES

The relationship between the data representation in TTCN-3 and the data representation in CTE is shown in Figure 3. Starting from left, TTCN-3 data types for messages or parameters and TTCN-3 templates can be converted to equivalence classes within a classification tree. In order to convert back to TTCN-3, the equivalence classes in the classification tree can be converted to TTCN-3 parameters or templates. By means of bidirectional conversion, predefined data can be reused and data refinement can be performed both for the TTCN-3 code and the classification tree. The steps of conversions, re-usage and refinement will be explained below by use cases of the application of CTE for test data specification within TTCN-3. Herein, three use cases considering test data generations are taken into account: type-based, testcase-based and template-based test data generations.

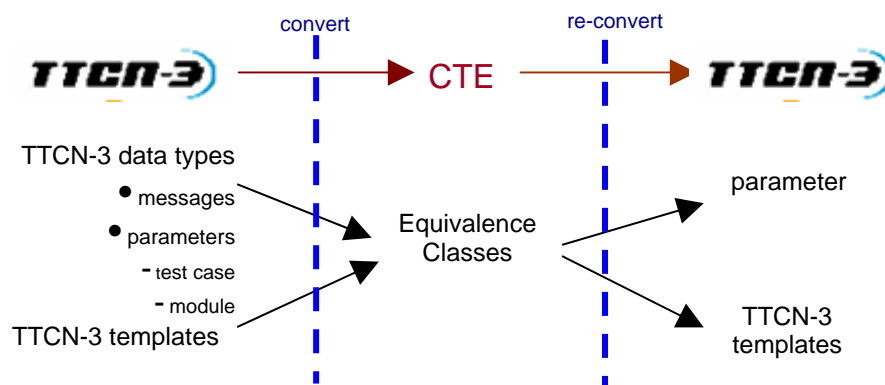


Figure 3: Data integration of TTCN-3 and CTE

4.1. Type-Based Test Data Generation

In the type-based test data generation use case, the starting point is a particular TTCN-3 data type selected by the test developer. This data type is used to generate a classification tree in CTE according to the structure of the type. The test developer then has to add appropriate criteria and combinations for the equivalence classes and a selection of interesting combinations of classes (if not using CTE’s build-in selection mechanisms). Each of these selected test cases in the CTE can be converted to a TTCN-3 data template. Please note that the classification tree method uses the term “test case” in a different way than TTCN-3. A CTE test case only defines test data. A TTCN-3 test case also specifies test behaviour.

An example for the type based test data generation is presented in Figure 4. For a better identification of the names in the classification tree towards the TTCN-3 code, the following naming convention is proposed: the name of the root of the classification tree is the name of a test module (the top level structure for test suites in TTCN-3). The composition name in the tree is the name of the TTCN-3 data type. Classifications are present if the type in question is a record type; in this case, the classifications are named after the fields of the record. The classes are the equivalence classes of the possible values of the scalar type or record fields, respectively. For the example shown in Figure 4, a TTCN-3 data type called *RadioType* in a test suite called *RadioNavTest* is defined:

The classification tree root is named to *RadioNavTest*, according to the TTCN-3 test suite name. The composition in the tree is named to *RadioType*, like the given data type, which is a record type with three fields of types *charstring*, *Boolean*, and *integer*, respectively. Those three field types build the classifications in the tree. The classes in the example tree are chosen by the test developer. If not immediately available from the classes chosen in the tree as such the value for *Market* = “USA” in Figure 4, representatives of classes are defined by the test developer, such as the representative *SignalStrength* = -1 for the class “< 0”. In many cases, there are reasonable default values (e.g. *true* and *false* for *Boolean*), or test critical values (e.g. 0 for *integer*). The classification tree in Figure 4 embodies three CTE test cases, representing three different TTCN-3 templates. For instance, the test case marked in Figure 4 represents the TTCN-3 data template { “USA”, *true*, -1 }.

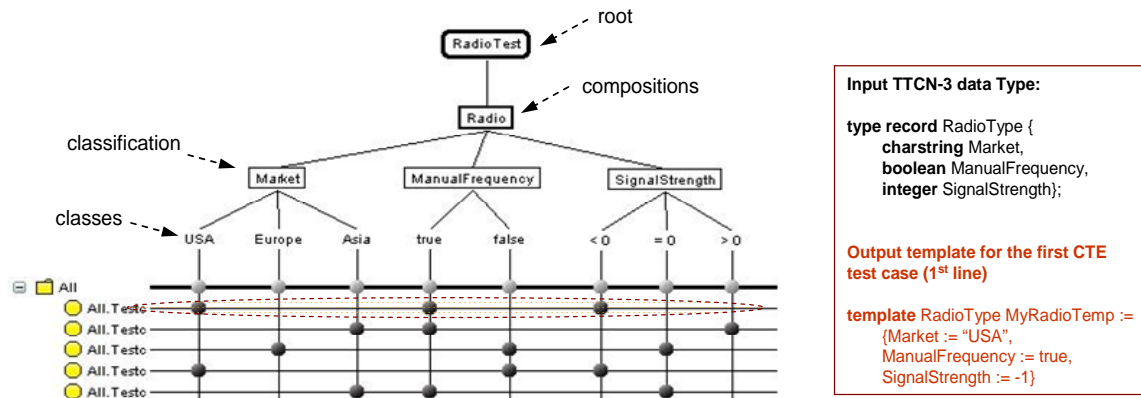


Figure 4: Example on type-based test data generation

4.2. Testcase-based Test Data Generation

Beside test data specification, TTCN-3 also specifies test behaviour. This second test data generation use case is based on a given TTCN-3 test behaviour (i.e. a test case or a test function) and data occurring in this behaviour (i.e. data types and templates). The difference between type-based and testcase-based test generation is that in the former case test behaviour does not take part at all.

The generation method is divided into two steps. First, it converts TTCN-3 data types into a classification tree, containing compositions and classifications, but no classes yet which as in the type-based approach need to be defined by the test developer. Second, it converts the CTE test cases back to TTCN-3 templates. Because of its re-conversion, this approach is also suitable for data refinement where data should be concretized step-by-step.

Step 1: In the first step, a TTCN-3 test behaviour fragment is given which describes the test behaviour, e.g. a send-receive cascade where only the test data types and no concrete test data or templates are given.¹ Using these TTCN-3 type information as data input, a classification tree is generated in CTE. The choice of classes is taken by the test developer. The classification tree comprises of sub-trees for each data type used in the given code fragment. By means of the classification tree, all combinations of test templates can be generated automatically or a subset of test templates can be selected either by hand or by means of the operators * and +.

In the example below, two data types, *RadioType* and *NavType* and two variables of these types are defined. Additionally, a short code fragment of test behaviour is specified, where in the beginning, test data of the type *RadioType* is sent via *port1*. After that, some responses are received from *port2*. In the end, a second test data of data type *NavType* is sent.

```

// type definitions:
type record RadioType { charstring Market, boolean ManualFrequency, integer SignalStrength };
type record NavType { charstring Map, integer SignalStrength };
var RadioType RadioPDU;
var NavType NavPDU;

// Behavior definition in TTCN-3-like notation:
testcase myTestcase() runs on TC system sut {
  port1.send(RadioType); // Here, only types are given, not templates.
  port2.receive(*);
  port1.send(NavType);
  setverdict(pass);
}

```

¹ Strictly, the code shown in the example is not valid in TTCN-3 since in this language it only allowed to send concrete data, i.e. values or completely specified templates. In practice, we elevate the implementation problem that the TTCN-3 compiler embedded in the Eclipse platform rejects code like the one in the example by misusing the parameterization mechanism of TTCN-3 to specify which data types have to be classified by means of CTE. We however omit a detailed discussion of this implementation detail for the sake of clearness.

Figure 5 shows the classification tree with the appropriate test templates. Each data type incarnates one classification sub tree. In this example, since we only concentrate on the stimuli in the test behaviour, only two sub trees for *RadioType* and *NavType* are needed in the classification tree. If all combinations of test templates should be generated automatically, $3*2*3*3*3 = 162$ templates at maximum (product of the number of classes of the bottom-most classifications) can be generated.

Step 2: In the second step, CTE test cases are re-converted into TTCN-3 test behaviour code fragments. For each CTE test case, the code fragment for test behaviour is duplicated and turned into a separate test case while replacing the abstract type information within the send and receive commands with the generated templates.

Continuing with our example, the sub-trees marked as *r1* and *n1* as shown in Figure 5 are selected for template generation and mapped into TTCN-3 as shown as follows. At first, the body of the TTCN-3 test case defining the test behaviour in question is “factored out” and transformed into a separate function with parameters corresponding to the data templates to be generated:

```
function myFunc(r1,n1) runs on TC {
  port1.send(r1);
  port2.receive(*);
  port1.send(n1);
  setverdict(pass);
}
```

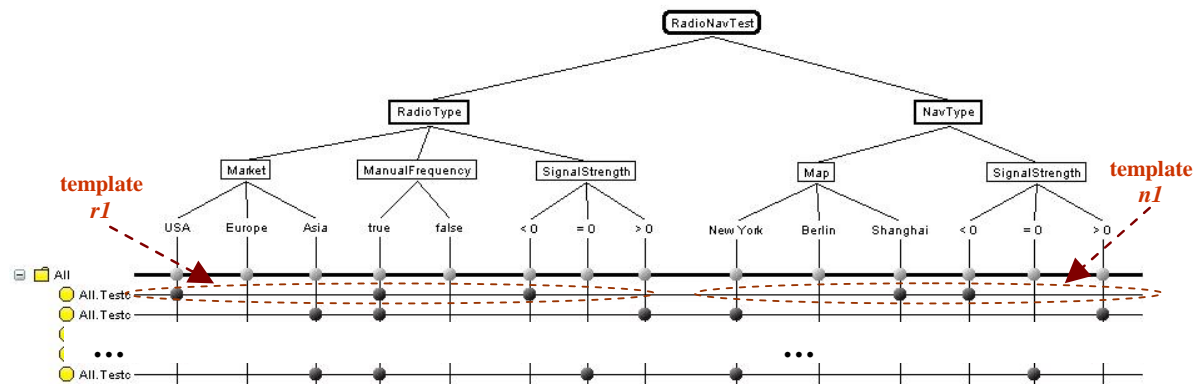


Figure 5: Template derivation of the classification tree

For each selected CTE test case, a new TTCN-3 test case is introduced which just calls the generated body function with template parameters derived from the classification tree.

From the first line in the CTE tree in Figure 5, the templates *RadioPDU*:{“USA”, true, -3} and *NavPDU*:{“Shanghai”, -1} are generated. These templates are then used as parameters for the call of *myFunc()* in the first generated test case. This scheme is repeated for each selected CTE test case. The generated test cases are then called in the control part of the test suite.

```
testcase myTestCase1 runs on TC system sut {
  myFunc(RadioPDU:{"USA",true,-3}, NavPDU:{"Shanghai",-1})
  // the templates derive from the first line of the CTE tree.
}
testcase myTestCase2 runs on TC system sut {
  myFunc(RadioPDU:{"Asia",true,3}, NavPDU:{"NewYork",1})
  // the templates derive from the second line of the CTE tree.
}
testcase myTestCase3 runs on TC system sut {
  myFunc(RadioPDU:{"Asia",true,0}, NavPDU:{"NewYork",0})
  // the templates derive from the last line of the CTE tree.
}
control {
  execute(myTestCase1);
  execute(myTestCase2);
  execute(myTestCase3);
}
```

4.3. Template-based Test Data Generation

A set of pre-defined data templates is taken as input for template-based test data generation as well as descriptions of test behaviour. Since these data templates can be constructed by e.g. using the type-oriented test data generation approach, the template based approach constitutes a bridge between the approaches discussed in the previous sections. The leaf nodes of the classification tree now comprise (names of) TTCN-3 templates.

A combination of data templates can be selected in the CTE, which are used in the test behaviour.

As an example, four templates for the data types *RadioType* and *NavType* introduced in the example of Section 4.2 are defined. The templates *r1* and *r2* are templates of data type *RadioType*. The templates *n1* and *n2* are templates of data type *NavType*:

```

template RadioType r1 := { Market := "Europe", ManualFrequency := true, SignalStrength := 0};
template RadioType r2 := { Market := "USA", ManualFrequency := false, SignalStrength := -2};
template NavType n1 := { Map := "Berlin", SignalStrength := 3};
template NavType n2 := { Map := "New York", SignalStrength := 0};
  
```

By means of the given templates, a classification tree is constructed (Figure 6). CTE test cases now define parameterizations of test cases with send and receive templates.

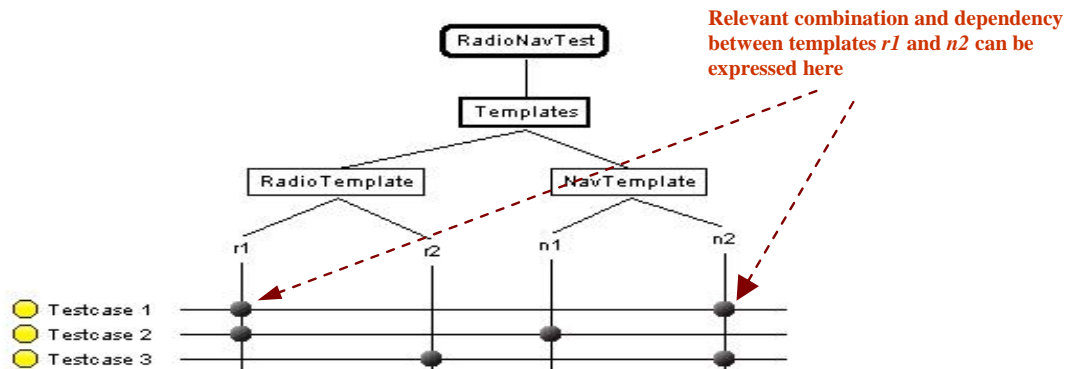


Figure 6: Example on template-based test data generation

After the re-conversion, the TTCN-3 code becomes like the following (using again the test case `myTestCase()` and the body function `myFunc()` already discussed in Section 4.2):

```

testcase myTestCase1 runs on TC system sut {
  myFunc(r1, n2) // the templates derive from the first line of the CTE tree.
}
testcase myTestCase2 runs on TC system sut {
  myFunc(r1, n1) // the templates derive from the second line of the CTE tree.
}
testcase myTestCase3 runs on TC system sut {
  myFunc(r2, n2) // the templates derive from the last line of the CTE tree.
}
control {
  execute(myTestCase1);
  execute(myTestCase2);
  execute(myTestCase3);
}
  
```

5. THE CTE INTEGRATION REALIZATION

The three CTE integration approaches discussed above has been realized by integration into the TTCN-3 plugin set TTworkbench for the Eclipse development environment. TTworkbench provides an open interface to access the internal representation of a TTCN-3 program. Thus Eclipse plugins can be developed that retrieve information on the syntactic structure of a TTCN-3 program and are able to perform syntactic modifications. The structure of a TTCN-3 is described by a so-called Metamodel for TTCN-3. We will not go into details about approaches for Model Driven Architecture (MDA), which are behind the model centric implementation of the

TTworkbench plugin set; the interested reader is referred e.g. to [7]. For the current purposes it is sufficient to consider a Metamodel as a machine processible version of the abstract grammar of a TTCN-3 program.

Figure 7 shows the implementation architecture of the CTE Plugin approach. From the instance of the TTCN-3 Metamodel, all relevant syntactic structures (i.e. selected data types, templates, and test case and function definitions) within the TTCN-3 code are obtained to be processed by the CTE plugin. The CTE plugin generates a temporary file readable by CTE and calls the editor for further user definition of the test data. When the user has finished with the specification of the test data within the classification tree, the CTE file is saved and exported back to the CTE Plugin again by means of an external file. The newly generated data are used to perform the modifications of the internal representation of the TTCN-3 code as described in the previous sections. The modified code is instantly shown in its textual (and graphical²) version in the TTworkbench editor windows.

The plugin implementation makes use of a Java Application Programmer Interface (API) and an associated classification tree library which is provided by CTE.

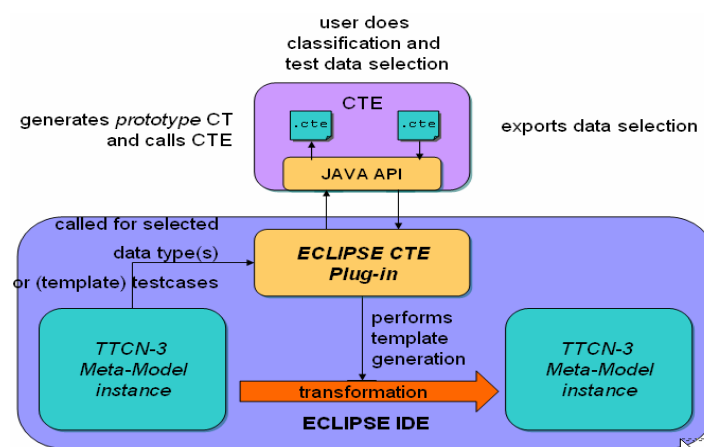


Figure 7: Implementation Architecture of CTE Integration Plugin

6. SUMMARY AND OPEN ISSUES

In this paper, an approach to integrate CTE with TTCN-3 to enable automatic test data generation has been described. Departing from the general idea to use classification trees for test data specification, three use cases has been investigated leading to three integration types. In order to proof the concept, this approach has been implemented as an Eclipse plugin based on the TTworkbench plugin set for TTCN-3. The following problems have been encountered:

- TTCN-3 does not allow for the specification of incomplete code, in particular, one has to use concrete values in send directives. Specification and refinement of test stimuli (and responses) however is the main goal of the classification tree method. Thus, proper syntactical means (distinguished from the function and parameter passing concepts of TTCN-3) are needed to define behavioural templates as a basis for the test case oriented and the template-oriented integration approach.
- In our current implementation, Eclipse with the TTworkbench (which drives the CTE plugin) and CTE are separate programs working on different internal data structures. This has the consequences that once a data classification has been performed and the control is delegated back to Eclipse, the tools “forget” about the classification step, i.e. the CTE cannot be reopened to work again on that classification to refine or to modify it. A closer integration based on a proper model based tool approach would have to use transformation of internal data to couple the tools (instead of communication via external files), and also provide extensive bookkeeping and change tracing mechanisms.

² Besides its textual representation format, TTCN-3 provides several other formats covering specific aspects of test definition. In particular, behavioral aspects can be specified using a graphical format [5].

REFERENCES

- [1] Classification Tree Editor. Available at <http://www.systematic-testing.com/>.
- [2] M. Grochtmann and K. Grimm: Classification trees for partition testing. *Software Testing, Verification and Reliability*, 3(2), 1993.
- [3] Eclipse Integrated Development Environment. Available at <http://www.eclipse.org/>.
- [4] ETSI European Standard (ES) 201 873-1 V2.2.1 (2003-02 Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language
- [5] ETSI European Standard (ES) 201 873-1 V2.2.1 (2003-02 Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 3: TTCN-3 Graphical presentation Format
- [6] TTworkbench plugin set for TTCN-3 and Eclipse. Information available at http://www.testingtech.de/products/ttwb_intro.php
- [7] OMG: Model-Driven Architecture (MDA) available at <http://www.omg.org/mda/>.