MaTeLo – Statistical Usage Testing by Annotated Sequence Diagrams, Markov Chains and TTCN-3

Winfried Dulz, Fenhua Zhen
Martensstrasse 3, D-91058 Erlangen, Germany
Institute for Computer Science, University of Erlangen-Nuremberg
Tel.: +49 9131 852 7929   Fax: +49 9131 85 27409
{dulz, zhen}@informatik.uni-erlangen.de

ABSTRACT

In this paper, we present a general framework for testing time-critical systems and software, as it is proposed in the European IST project MaTeLo. The main focus is on automatically generating a MCUM (Markov Chain Usage Model) starting from an FDT (Formal Description Technique) description in order to derive TTCN-3 (ETSI Testing and Test Control Notation version 3) compatible test case definitions.

Our approach is a combination of statistical usage testing based on a given MCUM and specification-based testing that is using FDT inputs. Within the MaTeLo project special attention is given to international standardized FDT notations, specifically ITU-T MSC (Message Sequence Chart) and OMG UML Sequence Diagrams. In addition, we make use of annotations in order to specify selected non-functional requirements to support the automated software testing of time-critical systems. We also defined an XML-based representation format called MCML (Markov Chain Markup Language) of the MCUM to build a common interface between various parts of the MaTeLo tool set.

Topics
Automated software testing, specification-based testing, testing strategies tools and standards.

Keywords
Automated software testing, Markov chain usage model, formal description technique, specification-based testing, XML, TTCN-3.
1. INTRODUCTION

Testing has a predominant role when developing software. While an error prone software is inconceivable, as development remains a human activity, although code is now partially generated, more and more efforts are spent on testing:

- to detect programming faults
- to evaluate the code reliability or performance
- to ensure that a critical function of a system meets given requirements

As depicted in [1], the ratio of time spent on testing should not be neglected:

- the cost of testing can go from more than 33% during the development phase to 67% during maintenance
- a bug becomes more and more expensive following the development progress. An error found during the integration process will cost ten times those from a unitary test. This rate can reach more than 100 times when the software has been released.

Testing can be classified in two broad non-isolated categories: structural (white box) and functional (black box) testing. **Structural testing** takes into account the structure of the software and is based on the code implementation. During the development cycle the first focus is on **functional testing** that deals with the software specifications validation. It considers only the expected functionality of the software from an external point of view - and has to be independent of the generated code.

In many application domains, e.g. mobile communication, avionic or automotive it is necessary to also specify **non-functional requirements**, in particular

- **timing** requirements, e.g. action deadlines or signal delays,
- **resource** restrictions, e.g. communication bandwidth,
- **deployment** directives and **probability** assignments to support performance and reliability evaluation,
- **tool directives** to enable performance and real-time related monitoring techniques [10].

Based on non-functional specifications it is possible to model, analyze, optimize, implement and test a specific system. **Non-functional testing** is done in parallel by using additional information that has to be imported from special tools that support the development of these kinds of systems.

The European IST project **MaTeLo** (Markov Test Logic) is based on recent researches in the field of **statistical usage testing** ([1], [3], [4] and [5]). Test cases are generated from a MCUM that describes the expected usage of the software - and gives a strictly external view of the software. It also gives the most efficient use of testing resources with respect to user perceived reliability. Furthermore, the ratio of errors found when going through the selected test cases is an accurate indicator of the system reliability and can be used to predict the software release time.

In MaTeLo special attention is given to the compliance with FDTs and existing standards, in particular MSC-96, UML 1.2 and TTCN-3 [6] that can be used for many types of reactive testing. Typical areas of application are protocol testing, module testing, testing of CORBA based platforms, testing of APIs etc.
Sequence diagrams (ITU MSC and UML/SD) are mainly used to describe dynamic system properties by means of messages and corresponding responses of interacting system components but they can also be applied for specifying test cases [7]. However, the standardized FDTs did not support the description of non-functional system properties in the past. Therefore a number of extensions have been proposed, e.g. [8], [9] either by adding new keywords and semantics to the languages or using annotational concepts. In MaTeLo the Corsair approach is used as explained in [10] and which is based on SDL*/PMSC [11].

2. STATISTICAL USAGE TESTING

When a population is too large for exhaustive study, as is the case for all possible uses of a software system, a statistically correct sample must be drawn as a basis for inferences about the population. In statistical testing of software, testing is treated as an engineering problem to be solved by statistical methods.

![Figure 1. Example for Markov Chain Usage Model](image)

A usage model as shown in Fig. 1 is a characterization of all possible scenarios of software use at a given level of abstraction. Usage models can be constructed before code is written and are finite representations of the infinite usage scenarios of a given system. A test case is any traversal of the usage model. A random test case is any traversal of the usage model based on state transitions that are randomly selected from a usage probability distribution. Certification means attaining reliability and confidence goals for an environment of use and to define stopping criteria.

The basic principle behind statistical usage testing is that the tests are based on the anticipated operational usage of the system. Test cases are generated from the user external view, and it thus guarantees independence with respect to the implementation. Predictions and analysis can thus rely on
realistic circumstances. The general approach to test a given SUT (system under test) by means of statistical usage testing consists of a number of consecutive steps [1]:

1. Get a system specification of the SUT by means of any formal or informal description technique.
2. Derive the structure of a MCUM (Markov chain usage model) of the SUT, i.e. a graph consisting of state nodes and arcs between states (this step is discussed in more detail in section 4).
3. Assign transition probabilities to the MCUM by means of requirement definitions, monitored field data of a running similar or predecessor system or simulation data. Because the system may work within different environments there may exist several sets of probabilities for a single MCUM structure.
4. Analyze and verify the MCUM by means of standard Markov techniques, such as
   • long-run occupancy, i.e. the percentage of time spent in each usage state
   • occurrence probability, i.e. the probability of state occurrences in a random run of the system
   • occurrence frequency, i.e. the expected number of state occurrences in a random run of the system
   • first occurrence, i.e. the expected number of other states in the system before a given state occurs the first time
   • expected sequence length, i.e. the expected number of state transitions in a random run of the system; this number is equal to the average length of a use case or a test case that are derived from the system specification.

The analysis of the MCUM will also show which parts of the system will be involved in heavy use and have to be tested more carefully. The outcome of the analysis may also lead to restructuring the MCUM or to reassign another set of transition probabilities to the arcs. In any case steps 2 to 4 have to be repeated until the MCUM is stable and will represent the SUT in its typical behavior.

5. Create special, nonrandom test cases to remove uncertainty about how the system will behave in various well known. Possible types of nonrandom tests are
   • model coverage tests, i.e. in using only the structure of the MCUM, a graph-theoretic algorithm generates the minimal sequence of test events to cover all arcs - and therefore all states,
   • mandatory tests, i.e. any specific set of test sequences that is known to fulfill mandatory requirement specifications,
   • regression tests are an effective way to discover the redundancy in the test suite and assess its omissions,
   • critical but unlikely use, i.e. critical states, transitions, and sub paths that would have low likelihood of arising in a random sample and have to be tested,
   • importance tests, i.e. importance sampling that can be implemented by producing transition probabilities that will emphasize high test value scores for the SUT in the sampling process and
   • partition testing, i.e. the MCUM can be used to identify and define partitions to gain a better sampling efficiency.

6. Planning of performing statistical tests means
   i. Estimation and generation of the number of random test cases that have to be run by using the expected test case length derived during model analysis.
ii. Definition of the best-case scenario, i.e. under the assumption that no failures will occur in random testing determine the values of product quality and process sufficiency that can be achieved by running the number of test cases generated in Step 1

iii. Definition of the worst-case scenario, i.e. assume some profile of failures and construct a failure log based on the profile.

iv. Analysis of the coverage of all model states, arcs, and paths that will occur during the test.

v. Analysis might show that testing as planned cannot fulfill the assumptions for model coverage or the required reliability. In this case one has either to revise the goals or the test plans.

7. Perform Random test cases that may be automatically generated from the MCUM. Each test case is a random walk through the MCUM, from its invocation state up to the termination state. In the case of TTCN-3 test cases consist of executable components, which produce input sequences for automated testing of the SUT.

8. Perform the test certification process, which calculates the merits of additional ongoing testing. The decision to stop testing can be derived from

- the confidence in a reliability estimate
- the degree to which testing experience has converged to the expected usage of the system
• model coverage criteria based on a degree of state, arc, or path coverage during random testing.

Very often, only subjective attributes are used to decide to stop test campaigns. Using a statistical approach allows however to assess the software reliability and thus to define stopping criteria that rely on a mathematical approach.

3. SPECIFICATION-DRIVEN TESTING

The basic idea behind specification-driven testing is derived from the method of **specification-driven performance monitoring** [2] which is illustrated in Fig. 2. Here, an object system that is fully specified by means of an FDT approach, e.g. SDL is instrumented for performance monitoring (Fig. 2b) using a so-called abstraction specification (Fig. 2a) that describes the system behavior with respect to different abstract views to the object system.

In this approach **PMSC** (Performance MSC) [11] is used for the abstraction specification, which is an annotated version of the MSC standard. Annotations that specify non-functional requirements, such as timing requirements, resource restrictions, deployment directives, probability and tool directives enable performance evaluation and real-time related monitoring techniques.

After instrumentation and system synthesis (Fig. 2c) the combined testing and monitoring (Fig. 2f) process is carried out by automatically generated executable test cases [7], which are derived from the selected abstraction specification (Fig. 2a) in the steps (Fig. 2d) and (Fig. 2e). Additional tool directives in the PMSC annotations will configure the performance evaluation tools and (Fig. 2g) in order to achieve the performance evaluation process (Fig. 2h) and to present the testing and measurement results in a user-friendly representation (Fig. 2i). Depending on the result interpretation (Fig. 2j) it is possible to
select another level of abstraction and to perform a hierarchical refinement of the system evaluation in any case of unexpected functional or non-functional behavior.

Note, that performance monitoring can be seen as a white box test, which is executed for a given implementation and nicely extends black box performance testing as described in [9].

**Specification-driven test case generation** within the MaTeLo project also starts from annotated sequence diagrams as shown in Fig. 3. The main difference is the generation of a MCUM, which will be explained in section 4.

The use of annotations permit the assignment of probabilities within an MSC scenario in order to select a given usage behavior for the test generation. In addition, TTCN-3 test verdicts PASS, FAILED and INCONCLUSIVE [6] can easily be extended by means of annotations for testing non-functional system properties, e.g. correct real-time behavior or user-defined system utilization.

Studies within the MaTeLo project have shown that there is a real need to extend the use of standard FDTs to new sectors, and to consequently develop associated testing methodologies and tools. Therefore, MaTeLo aims at facilitating a technological transfer between different application domains such as telecommunications, aerospace and automotive. It tries to confront the different opinions regarding FDT usage in heterogeneous environments.

At present, existing tools for test case generation produce test cases directly from the given specification language. That is why test campaigns are not optimized and can be greatly improved because

- test coverage is too large, or not relevant enough to guarantee an appropriate level of reliability
- test campaigns mean to be exhaustive, although this is an unreachable goal.

Within the MaTeLo approach a MCUM is build first as a dedicated test model. This building process can be automated and allows validating the specification by highlighting problems during the development process. ALITEC, which is one of the MaTeLo partners, has experimented that 20% of specification reworking after the specification stage can be avoided by building the Markov Model.

In the context of the MaTeLo project TTCN-3 [6], the new version of the standardized test specification language that has been published by the European Telecommunication Standardization Institute ETSI will play a major role.

DANET, which is another MaTeLo partner, has updated its current product so that TTCN-3 test cases can be processed. Due to the advantages that TTCN-3 offers, especially more flexibility and general support for testing outside the telecommunication conformance test philosophy, the test cases produced by the MaTeLo tool chain will be described in TTCN-3 notation.

In addition to the support of TTCN-3 the new TRI (TTCN-3 Runtime Interface) functionality will be considered. This standardized interface offers an easier and more stable basis for the implementation of protocol- and system-dependent interface libraries for the communication between the test system and the implementation under test.
4. GENERATING THE STRUCTURE OF THE MARKOV CHAIN USAGE MODEL

Fig. 3 illustrates that the first step inside the MaTeLo tool chain consists of deriving a Markov chain description from a certain MSC specification, which contains all scenarios of the future software usage given at a definite level of abstraction.

Therefore, an algorithm is required to convert a set of MSCs into a valid MCUM representation. In particular

- **conditions** within the MSC descriptions are mapped to **states** of the MCUM and
- **events** inside the MSCs are mapped to **transitions** of the MCUM.

In addition, non-functional requirements of the PMSCs [11] are reflected within special annotations of the MCUM.

The algorithm for transforming MSC scenarios into a MCUM representation consists of six successive steps:

2. Adding of so-called **intermediate conditions** into the user-defined MSCs (see Fig. 4) enables a straightforward mapping of MSC conditions to MCUM states. The strategy is the following: intermediate conditions (green hexagons) have to be added to all MSC instances if there is a direct sequence of MSC events (message, action, timer, etc) without a separating user-defined condition (blue hexagons) in between.

![Figure 4. Normalized MSC with intermediate conditions](image-url)
3. Selection of the granularity for the MCUM, i.e. at what level shall the system be tested:
   - Complete system behavior, i.e. all MSC instances
   - Partial system behavior of a set of MSC instances
   - Individual behavior of exactly one MSC instance

4. Construction of the MCUM that depends on the granularity the user has chosen. The main mapping algorithms is very straightforward:
   i. Start at the top of the instance and take the first condition, which is either user-defined or intermediate. Mark this as the first state after activation of the MCUM.
   ii. Take the next event (message, action, timer, etc) and put it into the transition to the next state of the MCUM.
   iii. Take the next condition (either user-defined or intermediate) and mark it as MCUM state that is reachable from the previous MCUM state.
   iv. Repeat steps ii and iii until the end of the instance is reached and mark the last condition as final state of the MCUM.

5. Optimization of the MCUM by removing superfluous intermediate conditions and merging redundant transitions. An intermediate conditions is superfluous if it is directly followed by a user-defined condition. The resulting MCUM for the Initiator instance is shown in Fig. 5.

![Figure 5. MCUM for the Initiator instance](image-url)
To understand the structure of the MCUM one has to know that the MCUM is constructed from a set of five MSC scenarios. Within the different MSCs the user has defined different usage behaviors starting from the same user-defined condition. For example, from the (blue) Wait state in Fig. 5 three transitions are possible due to three different MSCs, all containing the same user-defined condition named Wait.

6. Representation of the MCUM by means of an XML-based MCML (Markov Chain Markup Language) description. This allows to pass the MCUM conveniently between different parts of the MaTeLo tool set, i.e. converter, editor, TTCN-3 test case generator. On the other hand it is possible to use existing XML tools as shown in the next section.

5. IMPLEMENTATION OF THE MSC2MCML CONVERTER

In order to generate TTCN-3 test cases for the MSC-specified system, we developed an MSC-96 converter tool called msc2mcml (MSC to MCML transformation), which implements the algorithm described in the previous section. We decided to use standard, public-domain compiler development tools based on Java and XML. Fig. 6 gives an overview of the main parts developed within the msc2mcml converter.

In the front-end of the converter MSC files in instance-oriented PR (Phrase Representation) notation of the ITU Z.120 '96 standard edition are read by msc2xml and an intermediate XML file format is produced as output. The msc2xml converter was build by means of the public-domain compiler generator tools JavaCC [12] and JJTree [13].

![Figure 6. Main components of the msc2mcml converter](image)
In the back-end a set of Java classes is used to analyze and manipulate the intermediate XML files. These Java classes use JDOM [14] as an API and SAX [15] as the XML parser to read the XML descriptions. Via the MCML DTD (Document Type Definition), the final MCML format for the MCUM is generated as explained in section 4 above.

The advantage of this dual-stage converting technique is the flexibility with respect to changing input/output requirements. This may happen in the front-end for reading different input languages, e.g. XMI descriptions of UML sequence diagrams and in the back-end for deriving different versions of the MCML, e.g. additional annotations concerning the testing of non-functional user requirements.

6. SUMMARY

Statistical usage testing is a current research field that has barely been put into practice in the industrial world. The basic features of the MaTeLo approach, i.e. test automation starting from established specification techniques and optimization of the test process with the use of statistics based on a MCUM offer unique technical and economical prospects in the field of testing.

There is a strong need to improve and to extend FDT simulation, that would foster their adoption in the industry. Investigations on standard FDTs and their possible extensions will be further made within MaTeLo. For example it is not known weather specification techniques with a strong semantic model like SDL or MSC can easily be extended to validate and test non-functional requirement specifications as well, without using annotation concepts outside the scope of these languages.

By using a dedicated MCUM test model, it is possible to optimize the generated test campaigns, as well as to validate the specification at certain level of confidence. In addition, Markov modeling techniques allow to include innovative features like time constraints modeling or performance evaluation techniques more easily, because there exist a rich theory in using and evaluating these kind of system representation.

The new TTCN-3 notation and related technologies will make the test execution phase more efficient. Because TTCN-3 is no longer fixed to the conformance test of communication systems but offers modern component-oriented inter-communication facilities it can be applied to the description and generation of executable test cases for many application domains.

There is a real need for metrics related to software and system reliability, performance and quality of service. Important impacts on the development processes can therefore be expected: accurate predictions of software release time, better control of schedule and cost, increased quality, consumer satisfaction and last but not least safety.

7. REFERENCES


