

AGEDIS CONSORTIUM

AGEDIS



THE UML TESTING PROFILE

An Overview

www.agedis.de

Alessandra Cavarra

alessandra.cavarra@comlab.ox.ac.uk

The UML Testing Profile

In this article we introduce the UML testing profile draft and compare it to the AGEDIS Modeling Language (AML), emphasizing, when possible, commonalities and divergences.

Motivations and Objectives

So far, UML has focused primarily on the definition of system structure and behavior. However, with the approach towards system engineering according to model-driven architectures, the need for solid conformance testing is increased. The UML testing profile document [3] has been released in April 2002 in response to the OMG request for proposal ad/2001-06-08 [2]. The specific objective of this RFP is to define a UML profile that allows capturing all needed information for black-box test approaches to evaluate the correctness of system implementations. The UML testing profile is intended to support an effective, efficient and as far as possible automated testing of system implementations according to their computational UML models [2].

As solicited by the OMG RFP, the UML testing profile intends to fulfill three main objectives:

- is based upon the UML meta-model,
- enables the specification of tests for structural (static) and behavioral (dynamic) aspects of computational UML models, and
- is capable of inter-operation with existing test technologies for black box testing.

This profile is based upon UML 2.0. It is divided in three sub-packages: *test behavior*, which addresses the observations and activities during a test; *test architecture*, containing the elements and their relationships involved in a test; *test data*, i.e. the structures and meaning of values to be processed in a test.

In the UML testing profile the system under test is *not* specified as part of the test model; instead, the test architecture package *imports* the complete design (UML) model of the SUT in order to get the right to access to the elements to be tested. In such a way, the SUT can be exercised via its public interface operations and signals by the test components. However, no further information can be obtained from the SUT as it is a black-box.

In accordance to the UML, this testing profile is only a language and not a method (i.e. language plus process) and therefore it only provides a notation but no guide on how to use it. However, the testing profile is specific to a particular testing technology, namely *functional black box testing*, where the term functional refers to the correct functional behavior of system under test, i.e. its correct input/output behavior, and black box testing means that the internal structure of the SUT remains hidden.

The AGEDIS modeling language fulfills the first two of the three above-mentioned objectives addressed by the UML testing profile, i.e. is based upon the UML (1.4) meta-model, and enables the specification of tests for structural (static) and behavioral (dynamic) aspects of

computational UML models. AML is divided into only two packages: the *system model* depicting relevant elements of the SUT, and the *test directives* which define the collection of information that, when combined with the system model, defines the test suite that will be generated.

AML comes as part of the AGEDIS methodology [1] and has been designed with two main goals in mind: create a test adequate abstraction of the SUT that will be analyzed by the AGEDIS tools and set meaningful test directives for the testing process. An AML specification of the SUT is usually a submodel of the system design model and therefore parts of the design model can be reused for the test model, modulo possible adaptations. This means that interesting (i.e. relevant to the tests to be conducted) structural and behavioral aspects of the SUT are explicitly modeled, together with directives on what to test. Test models usually comprise class, object, and state diagrams. Test generation directives include: specific test purposes (currently modeled by extended state diagrams), generic coverage criteria, restrictions or constraints on the test cases and the test suite to be generated.

Whereas a test model written in the UML profile requires the specification of test objectives and the corresponding test cases, an AML model only defines directives on how to explore the system according to the behavioral model provided for it, whilst abstract test suites including sequences of stimuli and the expected behavior of the system in response to those stimuli are automatically generated from the behavioral model. No special syntax has been defined in AML to model test cases yet.

Terminology

In this section we give an overview of the terms and concepts of the UML testing profile packages together with their notation, when defined, and compare them with the corresponding AML concepts.

Test Behavior

The Test Behavior package describes the concepts required to specify test cases and their associated behaviors. Test behaviors might be described in different forms such as *Sequence Diagrams*, *Sequence Activity Graph*, or *State Machines*.

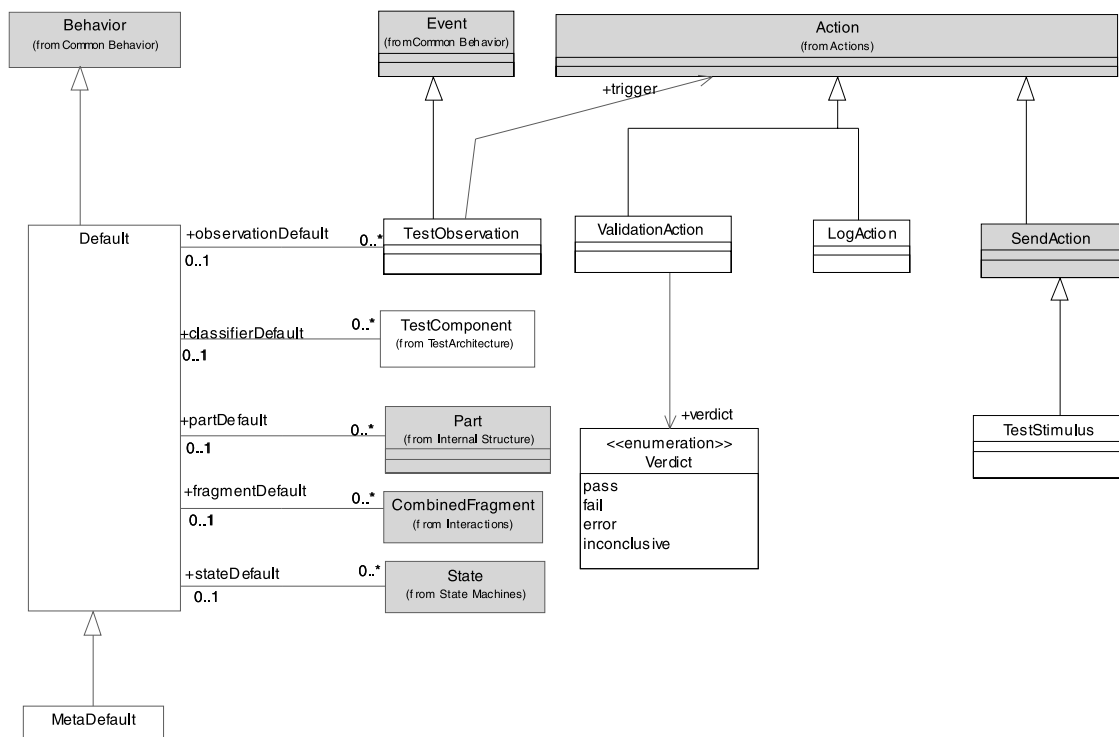


Figure 1: Test Behavior Package

Test Objective A test objective is a named element describing what should be tested. The notation to model test objectives *has not been defined yet* (this information refers to the document [3]). In the ATM example provided in the profile document, natural language has been used.

In AML, test objectives correspond to test directives.

Test Case A test case is a specification of one case to test the system, including what to test with which input, result, and under which conditions. It is a complete technical specification of how the SUT should be tested. A test case is defined in terms of sequences, alternatives, loops and defaults of stimuli to and observations from the SUT. It implements a test objective. A test case may invoke other test cases. A test case is a property of a test context.

A test case is executed to verify the adherence of the SUT to a test objective. A test case execution results in a trace being associated to the test case. Each trace has an associated verdict representing the adherence of the SUT to the test objective. A test case defines a behavior, which is realized by the set of test component objects (being defined in the internal structure of the test context of that test case). The semantics specific to a test behavior is provided by the following behavioral elements: Test stimulus (see below), Test observation (see below), Default (see below), Validation action (see below), Log action (see below). *There is no special syntax to define a test case.*

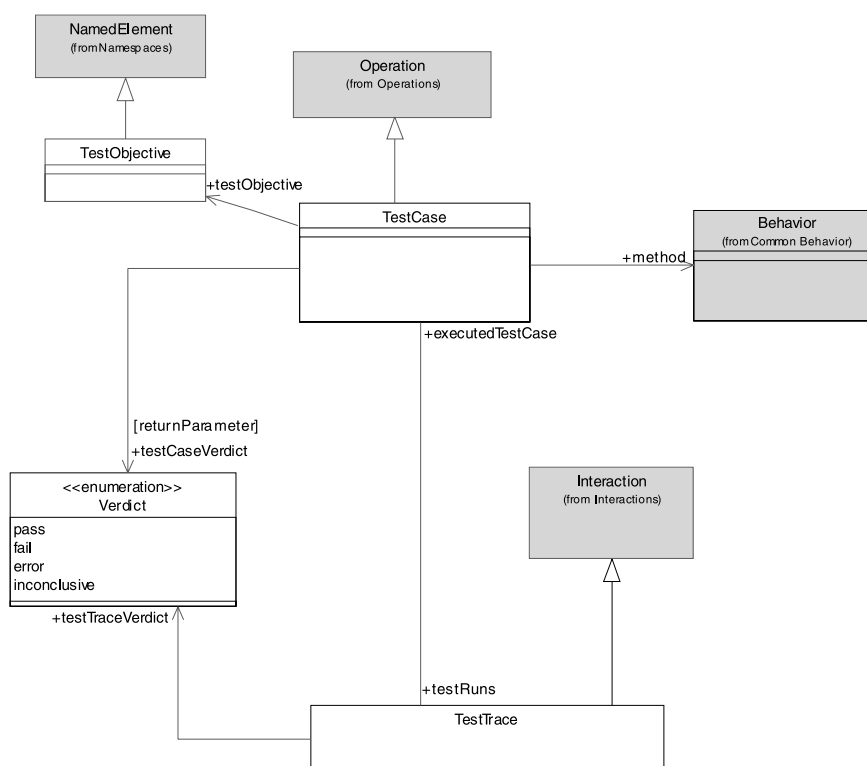


Figure 2: Test Case

Stimulus Test data sent to the SUT in order to control it and to make assessments about the SUT when receiving the SUT reactions to these stimuli.

Observation Test data reflecting the reactions from the SUT and used to assess the SUT reactions which are typically the result of a stimulus sent to the SUT.

Default A default is a behavior triggered by a test observation that is not handled by the behavior of the test case per se.

A default is triggered whenever a test observation is not handled by the behavior of a test case. There is a hierarchy of defaults:

- within the behavior of a test component object associated to a state (i.e. in a state machine) or to a combined fragment (in an interaction diagram);
- for the complete behavior of a test component object associated to a part (typed with a test component) in the internal structure of a test context (i.e. in a test configuration);
- for all test component objects of a test component associated to a test component in a test architecture.

In AML there is need to model defaults as the SUT is not viewed as a black box, and therefore we can handle these cases in the specification either at the system or test directives level.

Verdict Verdict is the assessment of the correctness of the SUT. Test cases yield verdicts. Verdicts can also be used to report failures in the test system. Predefined verdict values are *pass*, *fail*, *inconclusive* and *error*. *Pass* indicates that the test behavior gives evidence for correctness of the SUT for that specific test case. *Fail* describes that the purpose of the test case has been violated. *Inconclusive* is used for cases where neither a Pass nor a Fail can be given. An *Error* verdict shall be used to indicate errors (exceptions) within the test system itself. Verdicts can only be used in behaviors of test cases and are set by test components.

Setting a verdict is denoted by the **setverdict** keyword.

Validation Action A validation action is an action performed by a test component to assess a test observations and/or additional characteristics/parameters. A notation to model validation actions is still to be defined.

Test Trace A trace is an interaction resulting from the execution of a test case. It represents the different messages exchanged between the test components and the SUT. A trace is associated with a verdict representing the adherence of the SUT to the test objective of the associated test case. No special syntax is recommended to model test traces.

Log Action A log action is an action performed by a test component to log test observations and/or additional characteristics/parameters for further analysis.

Test Architecture

The test architecture package contains test components and/or related classes from which test configurations may be specified. The test architecture provides a basis for defining the types that will participate as parts¹ to realize the behavior of a test case. The test architecture

¹Parts have been introduced in UML 2.0. A part represents a set of instances that are owned by a containing classifier instance. Parts may be joined by attached connectors and specify configurations of linked instances to be created within an instance of the containing classifier.

concepts provides *reuse* of elements from other test architectures by using the importing capabilities provided with packages.

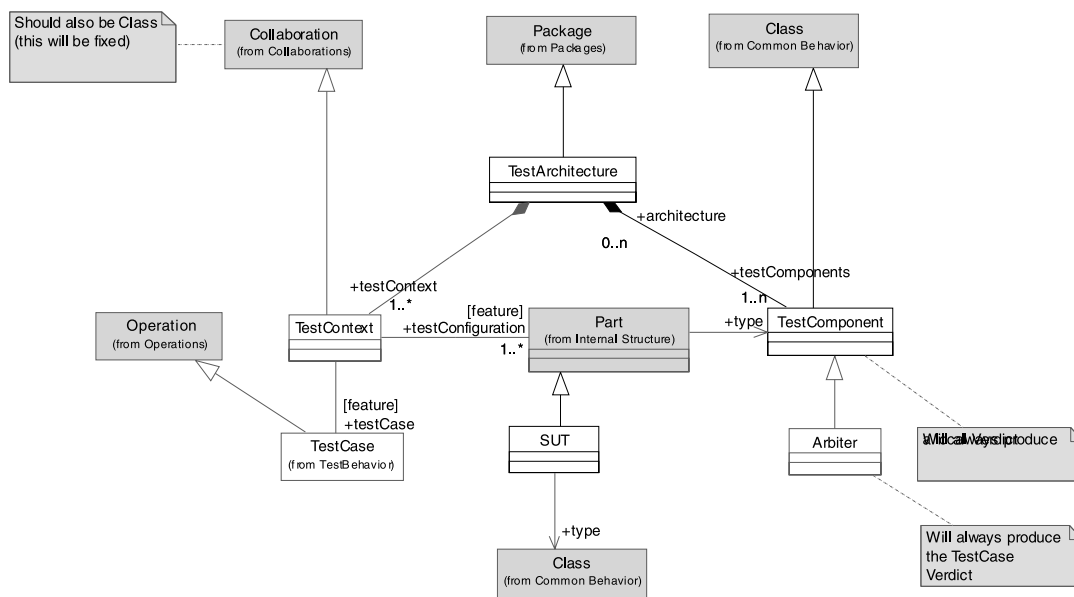


Figure 3: Test Architecture Package

SUT (System Under Test) The specification of a system under test (SUT), which is characterized by the set of interfaces via which a real SUT can be controlled and observed during testing by means of communication. A SUT can be on different abstraction levels: a complete system, a subsystem thereof, a set of or a single component, object or class. The SUT is exercised via its public interface operations and signals by the test components. No further information can be obtained from the SUT as it is a black-box.

Test Component A test component is a class of a test system. Test component objects realize the behavior of a test case. A test component has a set of interfaces via which it may communicate via connections with other test components or with the SUT.

A test component object executes a sequence of behaviors against the SUT in the form of test stimuli and test observations. It can also perform validation actions, and can log information into the test trace. Whenever a test component performs a validation action it updates its local verdict.

Arbiter The arbiter is a specialized test component which is responsible for maintaining the verdict for the test case as a whole. Every test component has a connection to the arbiter in order to keep it informed of changes in its local verdicts.

The arbiter realizes the semantics of verdict setting. Each test component in the test case has a connection to the arbiter, even if it is not explicitly shown. The arbiter is notified of the test component's verdict and updates the test case verdict according to the verdict setting rules. The arbiter is a passive component, except for reporting the test case verdict at the

conclusion of a test case.

This concept is not present in AML.

Test configuration The collection of test component objects and of connections between the test component objects and to the SUT. The test configuration defines both (1) test component objects and connections when a test case is started (the initial test configuration) and (2) the maximal number of test component objects and connections during the test execution.

Test Context A test context is a collaboration that specifies a test configuration and a set of test cases being executed on this test configuration. The test context shows how instances of the elements in the test architecture are combined to realize various test objectives.

A test context contains as internal structure a test configuration, which consists of a set of test component objects and SUTs. The SUT is a set of black-box parts being tested. The test configuration may contain additional parts being neither test component objects nor SUT parts. These additional parts can be used miscellaneous for test cases. The test component objects and SUT parts can be connected via connectors. The connectors allows operations specified by the interfaces of the SUT/test components to be invoked by other parts connected to the SUT/components.

Test Data

In general, test data refers to the specification of types and values that are received from or sent to the SUT. Data can be specified as being static or dynamic. Where, static data refers to the definition of types, and values given as arguments or read-only attributes. And dynamic data refers to the manipulation of values during the execution of a test behavior. UML does not have a concrete syntax for values and expressions, but does allow the use of OCL within constraints.

Static Data

UML allows the definition of complex types and values through the definition of classifiers. However, the use of concepts, such as pattern matching, logical partitions and TTCN-3 templates, tend towards more concise test specifications making them much easier to validate. Pattern matching and logical partitions allow the definition of equivalence value sets. For example, the specification of any value, ranges, or abstract value sets etc. Whereas, TTCN-3 templates are used to allow the partial value definition for a type.

Test Parameter The concrete technical specification which data is sent to the SUT (stimulus) and should be received from the SUT (observations). Test parameters might consist of several arguments and/or data partitions.

Argument A concrete physical value for a parameter used in a stimulus or in an observation.

Data Partition A logical value for a parameter used in a stimulus or in an observation. It typically defines an equivalence class of physical values (e.g. valid user names.).

Templates Templates to represent sets of values for test data can be specified using sub-typing.

Dynamic Data

The definition of dynamic data has not been provided yet.

Bibliography

- [1] AGEDIS - Automated Generation and Execution of Test Suites for Distributed Component-based Software. <http://www.agedis.de>.
- [2] Object Management Group. RFP for a UML Testing Profile - ad/2001-06-08. <http://neptune.irit.fr/Biblio/01-06-08.doc>
- [3] Object Management Group. UML Testing Profile (revised initial submission). http://www.fokus.gmd.de/research/cc/tp/projects/u2tp/Related_Documents/UMLTestingProfile_InitialRevised1.2.pdf