FTF Report

of the

UML 2.0 Testing Profile Finalization Task Force

to the

Platform Technical Committee

of the

Object Management Group

Version: 19th April 2004

Document Number: ptc-2004-04-01

Accompanied by: ptc-2004-04-02 (revised spec)
ptc-2004-04-03 (revised spec with changebars)

Table of Contents

Summary of UML 2.0 Testing Profile FTF Activities

Formation

- Chartered By: Platform Technology Committee
- On: June 6, 2003 in Paris, France
- Comments Due Date: September 8, 2003
- Report Due Date: April 30, 2004

Revision / Finalization Task Force Membership

Member	Organization	Status
DESFRAY, Philippe	Softeam	Charter, Veto
GERY, Eran	ILogix	Charter
SAMUELSSON, Eric	Telelogic AB	Charter, Veto
MANSUROV, Nikolai	KLOCwork Inc.	Charter
HAUGEN, Oystein	Ericsson	Charter (Resigned Dec. 2003)
LUCIO, Serge	International Business Machines	Charter, Co-chair, Veto
BAKER, Paul	Motorola	Charter
SCHIEFERDECKER, Ina	Fraunhofer FOKUS	Charter Co-chair

Initial Issues from Architecture Board Review:

AB Issue No: 6290

Title: Details of the Standalone Model

Source:

Pete Rivett, Adaptive Inc, pete.rivett@adaptive.com

Summary:

The standalone metamodel is not detailed enough to enable the implementation of testing profile compliant tools. In particular, many of the classes are having no attributes. Hence, a behavioural and further semantic foundation should be added to the standalone metamodel.

Resolution:

We will include the Hyades standalone metamodel for the UML Testing Profile, which is a real implementation and has to solve all the open issues of the current standalone metamodel.
Basically, we add a name and a definition to all the concepts in the standalone model.

Revised Text:

AB Issue No: 6291

Title: XMI Schema (Profile and Standalone Model)

Source:

Pete Rivett, Adaptive Inc, pete.rivett@adaptive.com

Summary:

The UML Testing Profile lacks currently a definition of XMI schema for the profile and the standalone model. These are needed to allow tool interchange of test specifications. Here, the reference to an XML DTD in the compliance point definition should be changed to a reference to the schema definitions.

Resolution:

.

Revised Text:

Issue Disposition:

Disposition	Number of Occurrences	Meaning of Disposition
Resolved	8	The RTF/FTF agreed that there is a problem that needs fixing, and has proposed a resolution (which may or may not agree with any resolution the issue submitter proposed)
Unresolved	14	The RTF/FTF agrees that there is a problem that needs fixing, but could not agree on a resolution.
Deferred	0	The RTF/FTF agrees that there is a problem that needs fixing, but decided to defer its resolution to a future RTF working on this specification (perhaps because of a lack of time or urgency).
Transferred	0	The RTF/FTF decided that the issue report relates to another specification, and recommends that it be transferred to the relevant RTF.
Closed, no change	2	The RTF/FTF decided that the issue report does not, in fact, identify a problem with this (or any other) OMG specification.

Duplicate or merged	0	This issue is eitehr an exact duplicate of another issue, or very closely related to another issue: see that issue for disposition.

{enter the number of occurrences for each disposition. For example, if 10 total issues were reported to the RTF / FTF and 6 were Resolved, 1 was Unresolved, 1 was Closed with no change, and 2 were Duplicate, then you would enter the respective number in each of the rows above.}

Voting Record:

Poll No.	Closing date	Issues included
1	October 20, 2003	**OMG ISSUE NO: 6293** **OMG ISSUE NO: 6297** **OMG ISSUE NO: 6305**
2	March 15, 2004	**AB ISSUE NO: 6290** **OMG ISSUE NO: 6292** **OMG ISSUE NO: 6294** **OMG ISSUE NO: 6295** **OMG ISSUE NO: 6300** **OMG ISSUE NO: 6951** **OMG ISSUE NO: 6952** **OMG ISSUE NO: 6953**
3	April 5, 2004	**AB ISSUE NO: 6291** **OMG ISSUE NO: 6296** **OMG ISSUE NO: 6298**

		OMG ISSUE NO: 6299 **OMG ISSUE NO: 6301** **OMG ISSUE NO: 6956** **OMG ISSUE NO: 6955** **OMG ISSUE NO: 6302** **OMG ISSUE NO: 6306** **OMG ISSUE NO: 6307** **OMG ISSUE NO: 7193** **OMG ISSUE NO: 7194** **OMG ISSUE NO: 6303** **OMG ISSUE NO: 6304** **OMG ISSUE NO: 6954** **OMG ISSUE NO: 7104** **OMG ISSUE NO: 7218**
4	{ Spell out date poll closed, e.g. 25 January 2003}	{Issue numbers of all issues whose resolution was included in this poll}
5	{ Spell out date poll closed, e.g. 25 January 2003}	{Issue numbers of all issues whose resolution was included in this poll}

Voter	Vote in poll 1	Vote in poll 2	Vote in poll 3
DESFRAY, Philippe	Yes	Did not vote	Did not vote
GERY, Eran	Did not vote	Did not vote	Did not vote
SAMUELSSON, Eric	Yes	Abstain: 6290,	Yes

		6300 Yes: all others	
MANSUROV, Nikolai	Did not vote	Did not vote	Did not vote
HAUGEN, Oystein	Yes	(resigned Dec. 2003)	
LUCIO, Serge	Yes	Yes	Abstain: 6307 Yes: all others
BAKER, Paul	Yes	Yes	Yes
SCHIEFERDECKER, Ina	Yes	Yes	Yes

Summary of Changes Made

The UML 2.0 Testing Profile FTF made changes that:

- Corrected features that impeded implementation or did not serve the original intent of the specification
- Provided additional convenience for implementers
- Increased the clarity of the specification

The following is a table that categorizes the issues as to the degree of changes that were made in resolving them.

Extent of Change	Number of Issues	OMG Issue Numbers
Significant - Fixed problems with normative parts of the specification that raised concern about implementability	**6**	**6290, 6293, 6295, 6300, 6951, 6291,**
Minor - Fixed minor problems with normative parts of the specification	**11**	**6292, 6297, 6305, 6952, 6296, 6298, 6299, 6301, 6307, 6303, 6954**
Support Text -Changes to descriptive, explanatory, or supporting material.	**6**	**6302, 6306, 7193, 6304, 7104, 7218**

Disposition: Resolved

OMG Issue No: 6293

Title: Verdict in Test Case (Standalone Model)

Source:

IBM, Serge Lucio, slucio@us.ibm.com

Summary:

Verdicts are part of a test case specification; however, a test case specification may contain several different verdicts – as different test sequences (all part of that test case) may lead to different verdicts. Hence, there is not a single test verdict and the attribute verdict of a test case should be deleted.

Resolution:

Delete the attribute from the test case stereotype

Revised Text:

A revised metamodel figure and a revised text:

Test Case
Semantics
A test case is a set of behavior performed against the SUT and owned by a test suite. Test cases have access to all elements in a test suite, including the SUT elements and test components. A test case produces a trace containing all log actions and the verdict.
Associations
• behavior:Behavior[1] The dynamic behavior of the test case.
• executions:Trace[0..*] Trace elements representing the logged information for each execution of a test case.
• testObjective:TestObjective[1..*] A test objective is a description of the capability being validated by the test case.
• testSuite:TestSuite[1] The test suite to which the test case belongs.

OMG Issue No: 6297

Title: Traces vs. Log/Journal/… (Profile and Standalone Model)

Source:

IBM, Serge Lucio, slucio@us.ibm.com

Summary:

The notion trace could be confusing as it is used e.g. in UML 2.0 interactions to define the interaction semantics. Hence, a renaming is proposed.

Resolution:

Rename trace to TestLog.

Revised Text:

All occurrences of trace in figures and texts to be renamed to log.

OMG Issue No: 6305

Title: Editorial for Fig. 4

Source:

U2TP Consortium, u2tp@fokus.fraunhofer.de

Summary:

There is a “hanging” e top left of Fig. 4.

Resolution:

Simply delete it.

Revised Text:

The revised figure.

AB Issue No: 6290

Title: Details of the Standalone Model

Source:

Pete Rivett, Adaptive Inc, pete.rivett@adaptive.com

Summary:

The standalone metamodel is not detailed enough to enable the implementation of testing profile compliant tools. In particular, many of the classes are having no attributes. Hence, a behavioural and further semantic foundation should be added to the standalone metamodel.

Resolution:

The Eclipse/Hyades project has built a working implementation of the standalone metamodel. The standalone model has been updated to reflect the improvements and modifications done by the Hyades working groups, specifically in the areas of attributes.
Primarily, every element becomes a “named element”, and for those requiring a specification or definition, not captured in the standalone metamodel, they include a reference to this specification, which might be proprietary to the tool.
The scope of the metamodel has been added to the text of the submission and limits the interoperability of the standalone metamodel to architectural elements.
Behaviors are out of the scope of the MOF metamodel. The LogAction, ValidationAction, Default, DefaultApplication are removed from the standalone metamodel.

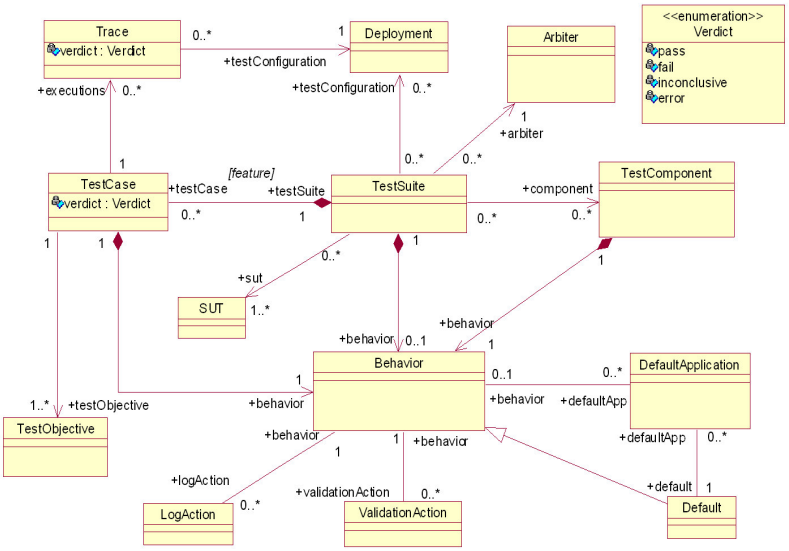
Revised Text:

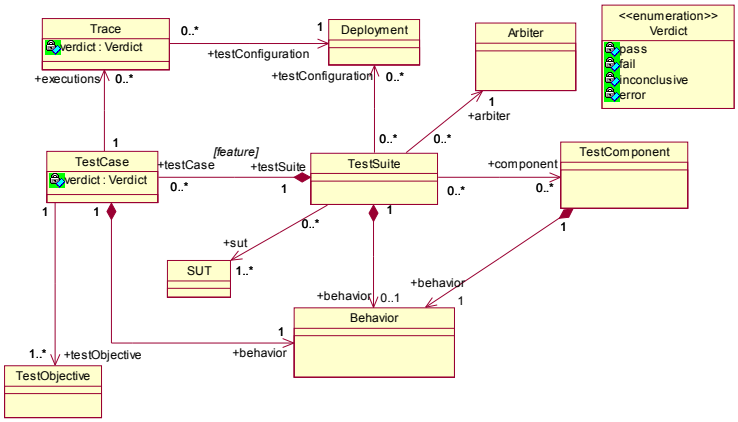
The introduction is changed to reflect the scope of the MOF metamodel:

> This section provides a standalone metamodel for the UML Testing Profile. This metamodel is an instance of the MOF metamodel, providing the ability for MOF based tools to comply with the Testing Profile standard. The compliance level that We present the MOF metamodel basic diagrams and provide more detailed information as it pertains to the metamodel. Primarily the structural elements of the Testing Profile are present in the metamodel, enabling data interchange between MOF based testing tools. Specifically, the behavioral aspects of the testing profile are not present in the metamodel as they would require a significant portion of A majority of the concepts from the Testing Profile are also present in the metamodel.

This section provides a standalone metamodel for the UML Testing Profile. This metamodel is an instance of the MOF metamodel, providing the ability for MOF based tools to comply with the Testing Profile standard. The compliance provided by the MOF-based metamodel is limited to the architecture elements of the Testing Profile enabling traceability and management of tests assets across tools. Specifically, the behavioral aspects of the testing profile are left out of the current metamodel as they would not provide any susbstantial improvement over this goal, while requiring a significant portion of the UML 2.0 metamodel to be included in the standalone metamodel.

All behavioral aspects such as LogAction, ValidationAction, Default, DefaultApplication are removed. The architecture diagram is updated:





The following text is removed:

Log Action

Semantics

A log action is an element in a behavior that specifies that an entity should be logged to the execution trace for further analysis. The target of a log action is a logging mechanism in the run-time system. The representation of this system is not specified.

Associations

behavior:Behavior[1]	The behavior in which the log action is specified.

Validation Action

Semantics

When a validation action is specified, it indicates that at run-time, the expression within the action will be evaluated. If the expression evaluates to true, the verdict "pass" is sent to the arbiter using the setVerdict operation. If the expression is false, the verdict "fail" is sent to the arbiter. Instead of an expression, valid verdicts for the arbiter implementation may also be used.

Associations

behavior:Behavior[1]	The behavior in which the validation action is specified.

Verdict

Verdict is defined exactly as in profile section 2.3

Default

Semantics

A default is a behavior that is activated when an unexpected event occurs on a test component.

Associations

defaultApp [0..*]	The set of applications of the default to specific test components.

Default Application

Semantics

A default application is the a reference which associates a particular default with a particular behavior. This allows the default to be activated should an unexpected event occur on a component.

Associations

behavior:Behavior[0..1]	The behavior to which the default application is attached.
default:Default[1]	The default being applied to the behavior.

The Behavior associations are updated:

Behavior

Semantics

Behavior represents the dynamic behavior of a test suite, test case, or test component in the testing system. In the MOF metamodel, it is a high level concept that allows the programmed behavior of the aforementioned elements to be explicitly referenced It is based explicitly on the behavior concept from the U2 partners UML 2.0 submission.

Associations

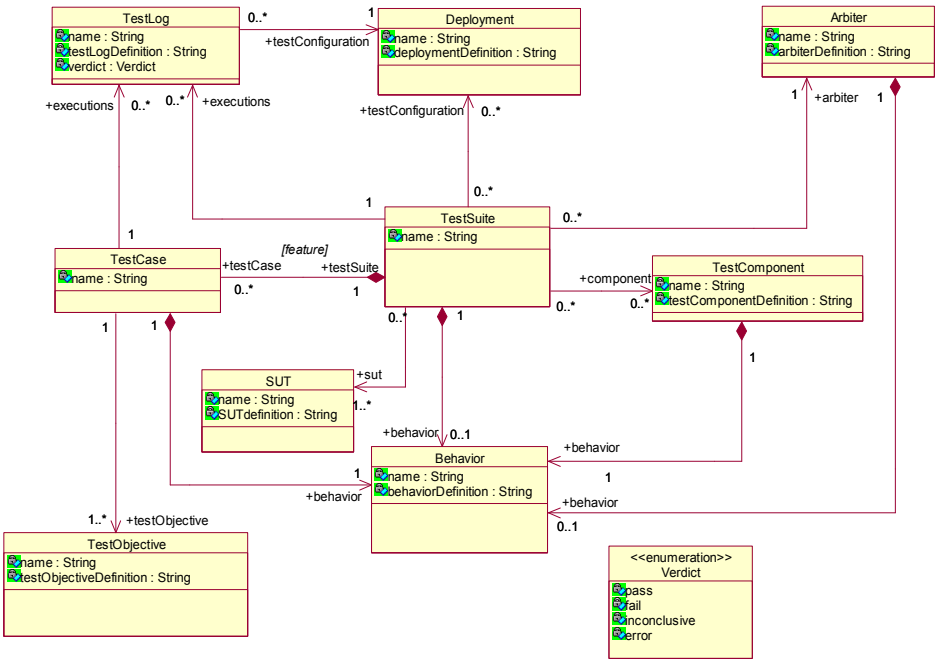
defaultApp:DefaultApplication[0..*]	The set of default applications which have been associated with the behavior.
logAction:LogAction[0..*]	The set of log actions that are performed when the behavior is executed.
validationAction:ValidationAction[0..*]	The set of validation actions that are performed when the behavior is executed.

Behavior

Semantics

Behavior represents the dynamic behavior of a test suite, test case, or test component in the testing system. In the MOF metamodel, it is a high level concept that allows the programmed behavior of the aforementioned elements to be explicitly referenced It is based explicitly on the behavior concept from the U2 partners UML 2.0 submission.

New attributes are added to the meta-classes: all meta-classes get a name and a definition attribute.



The attribute description section is updated to reflect that.

Also change the figure in the annex.

OMG Issue No: 6292

Title: Test Suite / Test Case

Source:

IBM, Serge Lucio, slucio@us.ibm.com

Summary:

Users of the testing profile are confused with the containment relation of test cases to test suites. Test cases are considered to be independent of test suites (they are designed in the test planning phase), while test suites just denote one possible execution of test cases.

Resolution:

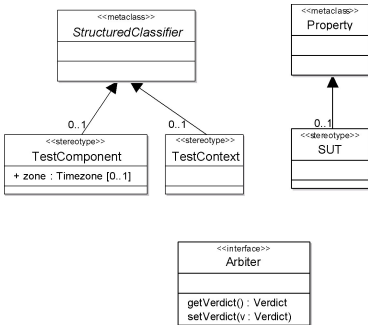
Rename as follows:

Test suite to become **test context**.

Revised Text:

For the **profile** model:

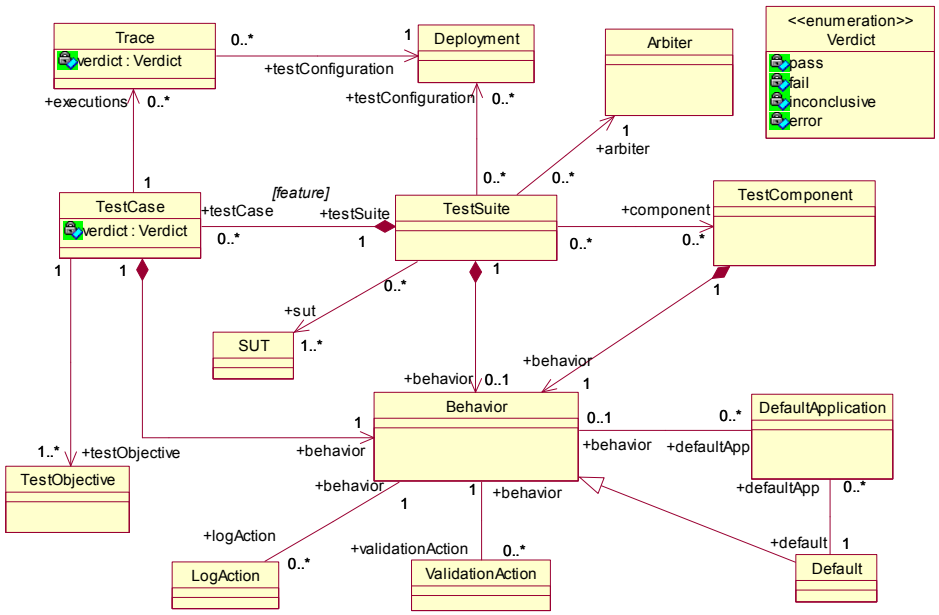
The architecture diagram is updated:

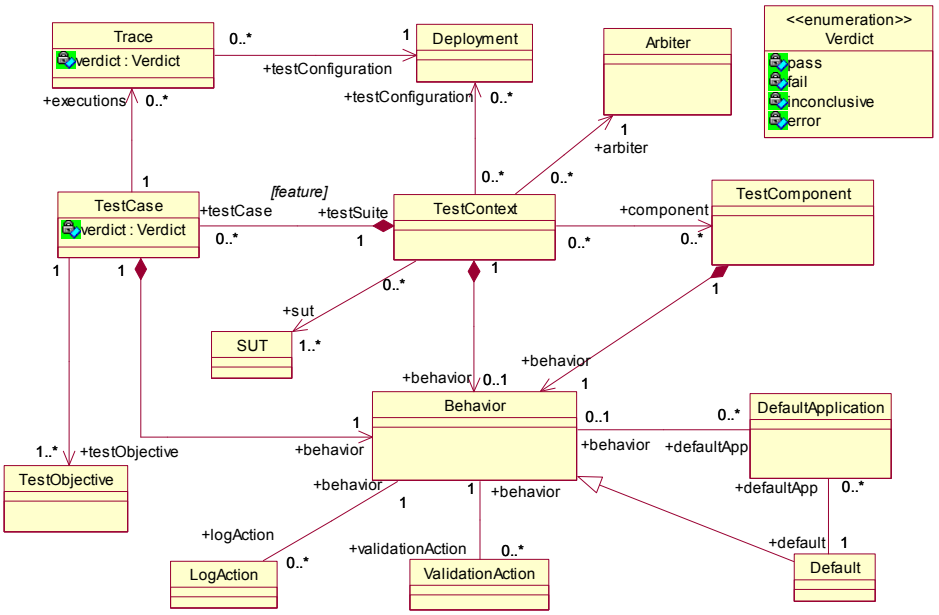


All the text is updated

For the **standalone** model:

The architecture diagram is updated:





The TestCase associations are updated:

Associations

	behavior:Behavior[1]	The dynamic behavior of the test case.

executions:Trace[0..*]	Trace elements representing the logged information for each execution of a test case.
testObjective:TestObjective[1..*]	A test objective is a description of the capability being validated by the test case.
testSuite:TestSuite[1]	The test suite to which the test case belongs.

Associations

behavior:Behavior[1]	The dynamic behavior of the test case.
executions:Trace[0..*]	Trace elements representing the logged information for each execution of a test case.
testObjective:TestObjective[1..*]	A test objective is a description of the capability being validated by the test case.
testContext:TestContext[1]	The test context to which the test case belongs.

The remaining descriptive text is updated to reflect the renaming from test suite to test context.

For the examples: all the figures and explanations are updated. Also, all further text such as in the glossary is updated.

Also change the figures in the annex.

OMG Issue No: 6295

Title: Traces (Standalone Model)

Source:

IBM, Serge Lucio, slucio@us.ibm.com

Summary:

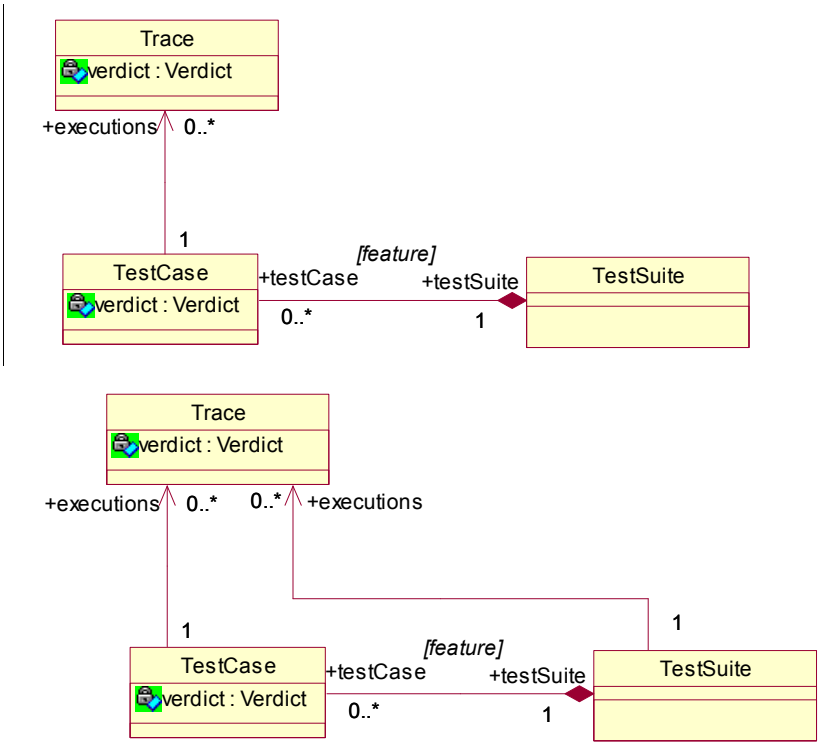
Traces are currently related to test cases only, but a trace should be possible for a test suite as well.

Resolution:

An association between test suite and trace should be added.

Revised Text:

The architecture diagram is updated:



s and text are

The TestSuite associations are updated with a new association:

executions:Trace[0..*]	Traced elements representing the logged information for each execution of a test suite.

Also change the figure in the annex.

OMG Issue No: 6300

Title: Simplification of the Standalone Model

Source:

IBM, Serge Lucio, slucio@us.ibm.com

Summary:

There are certain concepts in the standalone model which are of no use to the user: the default application, validation action and log action belong to behavior, which is not described in the standalone model. Hence, a tool vendor cannot really handle these concepts, e.g. a user could not really define a validation action outside of a context of a concrete behavioral specification.
Another view on this: we need for a tool vendor a compliance level which is an intermediate step from their existing tools to the standalone model. This compliance level should exclude the behavior definition, i.e. it should be open to any kind of test behaviors.

Resolution:

Remove behavior related concepts from the standalone model:

- logAction
- Default
- DefaultApplication
- ValidationAction

Revised Text:

See Issue No 6290.

OMG Issue No: 6951

Title: Relationship between Arbiter and Behavior (Standalone Model)

Source:

U2TP Consortium, u2tp@fokus.fraunhofer.de

Summary:

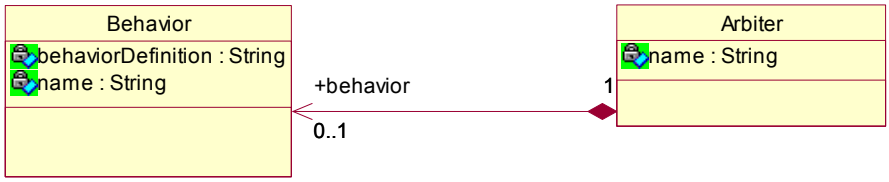
The current standalone model does not allow to define a behavior for an arbiter.

Resolution:

Add an aggregation from arbiter to behavior.

Revised Text:

An aggregation is added.



A new association is added to the Arbiter text:

Associations

behavior:Behavior[1]	The behavior of the arbiter.

OMG Issue No: 6952

Title: Default refining from Behavior (Standalone Model)

Source:

U2TP Consortium, u2tp@fokus.fraunhofer.de

Summary:

In the standalone model, default inherits from behavior, which would allow to define a test case or a test suite as a default.

Resolution:

The resolution relates to the resolution of 6300: remove defaults from the standalone model as anyhow behavioral concepts are not detailed in this model.

Revised Text:

See Issue 6290.

AB Issue No: 6291

Title: XMI Schema (Profile and Standalone Model)

Source:

Pete Rivett, Adaptive Inc, pete.rivett@adaptive.com

Summary:

The UML Testing Profile lacks currently a definition of XMI schema for the profile and the standalone model. These are needed to allow tool interchange of test specifications. Here, the reference to an XML DTD in the compliance point definition should be changed to a reference to the schema definitions.

Resolution:

The XMI schema for the profile will follow the XMI schema definition of UML2.0 which includes also the schema definition for UML 2.0 profiles.
The XMI schema for the standalone model will be included.
The text for the compliance point definition will be updated accordingly.

Revised Text:

Add an annex to the document with the following content:

XMI Schema

The Profile

The XMI schema definition for the exchange of U2TP profile specifications follows the XMI schema definition of UML 2.0 for UML 2.0 profiles. Please refer to the schema definition of UML 2.0

The MOF-based Metamodel

The XMI schema definition for the MOF-based metamodel is given below.

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://www.omg.org/U2TPSA"
    xmlns:u2tp="http://www.omg.org/U2TPSA"
    xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import namespace="http://www.omg.org/XMI"
schemaLocation="XMI.xsd"/>
    <xsd:simpleType name="Verdict">
        <xsd:restriction base="xsd:NCName">
            <xsd:enumeration value="pass"/>
```

```
            <xsd:enumeration value="fail"/>
            <xsd:enumeration value="inconclusive"/>
            <xsd:enumeration value="error"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="Arbiter">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="arbiterDefinition" type="xsd:string"/>
            <xsd:element name="behavior" type="u2tp:Behavior"/>
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="arbiterDefinition" type="xsd:string"/>
    </xsd:complexType>
    <xsd:element name="Arbiter" type="u2tp:Arbiter"/>
    <xsd:complexType name="Scheduler">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="schedulerDefinition" type="xsd:string"/>
            <xsd:element name="behavior" type="u2tp:Behavior"/>
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="schedulerDefinition" type="xsd:string"/>
    </xsd:complexType>
    <xsd:element name="Scheduler" type="u2tp:Scheduler"/>
    <xsd:complexType name="Deployment">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="deploymentDefinition" type="xsd:string"/>
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="deploymentDefinition" type="xsd:string"/>
    </xsd:complexType>
    <xsd:element name="Deployment" type="u2tp:Deployment"/>
    <xsd:complexType name="SUT">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="name" type="xsd:string"/>
```

```
            <xsd:element name="SUTdefinition" type="xsd:string"/>
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="SUTdefinition" type="xsd:string"/>
    </xsd:complexType>
    <xsd:element name="SUT" type="u2tp:SUT"/>
    <xsd:complexType name="TestComponent">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="testComponentDefinition"
type="xsd:string"/>
            <xsd:element name="zone" type="xsd:string"/>
            <xsd:element name="behavior" type="u2tp:Behavior"/>
            <xsd:element name="dataPool" type="u2tp:DataPool"/>
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="testComponentDefinition"
type="xsd:string"/>
        <xsd:attribute name="zone" type="xsd:string"/>
        <xsd:attribute name="dataPool" type="xsd:string"/>
    </xsd:complexType>
    <xsd:element name="TestComponent" type="u2tp:TestComponent"/>
    <xsd:complexType name="TestContext">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="testContextDefinition"
type="xsd:string"/>
            <xsd:element name="sut" type="u2tp:SUT"/>
            <xsd:element name="component" type="u2tp:TestComponent"/>
            <xsd:element name="arbiter" type="u2tp:Arbiter"/>
            <xsd:element name="scheduler" type="u2tp:Scheduler"/>
            <xsd:element name="behavior" type="u2tp:Behavior"/>
            <xsd:element name="testConfiguration"
type="u2tp:Deployment"/>
            <xsd:element name="testCase" type="u2tp:TestCase"/>
            <xsd:element name="executions" type="u2tp:TestLog"/>
            <xsd:element name="dataPool" type="u2tp:DataPool"/>
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
```

```
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="testContextDefinition" type="xsd:string"/>
        <xsd:attribute name="sut" type="xsd:string"/>
        <xsd:attribute name="component" type="xsd:string"/>
        <xsd:attribute name="arbiter" type="xsd:string"/>
        <xsd:element name="scheduler" type="u2tp:Scheduler"/>
        <xsd:attribute name="testConfiguration" type="xsd:string"/>
        <xsd:attribute name="executions" type="xsd:string"/>
        <xsd:attribute name="dataPool" type="xsd:string"/>
    </xsd:complexType>
    <xsd:element name="TestContext" type="u2tp:TestContext"/>
    <xsd:complexType name="TestLog">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="testLogDefinition" type="xsd:string"/>
            <xsd:element name="verdict" type="u2tp:Verdict"/>
            <xsd:element name="testConfiguration"
type="u2tp:Deployment"/>
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="testLogDefinition" type="xsd:string"/>
        <xsd:attribute name="verdict" type="u2tp:Verdict"/>
        <xsd:attribute name="testConfiguration" type="xsd:string"/>
    </xsd:complexType>
    <xsd:element name="TestLog" type="u2tp:TestLog"/>
    <xsd:complexType name="BaseDefault">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
    </xsd:complexType>
    <xsd:element name="BaseDefault" type="u2tp:BaseDefault"/>
    <xsd:complexType name="Behavior">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="behaviorDefinition" type="xsd:string"/>
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
        <xsd:attribute name="name" type="xsd:string"/>
```

```
        <xsd:attribute name="behaviorDefinition" type="xsd:string"/>
    </xsd:complexType>
    <xsd:element name="Behavior" type="u2tp:Behavior"/>
    <xsd:complexType name="TestCase">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="testCaseDefinition" type="xsd:string"/>
            <xsd:element name="behavior" type="u2tp:Behavior"/>
            <xsd:element name="executions" type="u2tp:TestLog"/>
            <xsd:element name="testObjective"
type="u2tp:TestObjective"/>
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="testCaseDefinition" type="xsd:string"/>
        <xsd:attribute name="executions" type="xsd:string"/>
        <xsd:attribute name="testObjective" type="xsd:string"/>
    </xsd:complexType>
    <xsd:element name="TestCase" type="u2tp:TestCase"/>
    <xsd:complexType name="TestObjective">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="testObjectiveDefinition"
type="xsd:string"/>
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="testObjectiveDefinition"
type="xsd:string"/>
    </xsd:complexType>
    <xsd:element name="TestObjective" type="u2tp:TestObjective"/>
    <xsd:complexType name="CodingRule">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="coding" type="xsd:string"/>
            <xsd:element name="value" type="u2tp:InstanceValue"/>
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
        <xsd:attribute name="coding" type="xsd:string"/>
        <xsd:attribute name="value" type="xsd:string"/>
    </xsd:complexType>
```

```
    <xsd:element name="CodingRule" type="u2tp:CodingRule"/>
    <xsd:complexType name="InstanceValue">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="literalAny" type="u2tp:LiteralAny"/>
            <xsd:element name="literalAnyOrNull"
type="u2tp:LiteralAnyorNull"/>
            <xsd:element name="literalNull" type="u2tp:LiteralNull"/>
            <xsd:element name="coding" type="u2tp:CodingRule"/>
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
        <xsd:attribute name="literalAny" type="xsd:string"/>
        <xsd:attribute name="literalAnyOrNull" type="xsd:string"/>
        <xsd:attribute name="literalNull" type="xsd:string"/>
        <xsd:attribute name="coding" type="xsd:string"/>
    </xsd:complexType>
    <xsd:element name="InstanceValue" type="u2tp:InstanceValue"/>
    <xsd:complexType name="LiteralAny">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="value" type="u2tp:InstanceValue"/>
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
        <xsd:attribute name="value" type="xsd:string"/>
    </xsd:complexType>
    <xsd:element name="LiteralAny" type="u2tp:LiteralAny"/>
    <xsd:complexType name="LiteralAnyorNull">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="value" type="u2tp:InstanceValue"/>
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
        <xsd:attribute name="value" type="xsd:string"/>
    </xsd:complexType>
    <xsd:element name="LiteralAnyorNull" type="u2tp:LiteralAnyorNull"/>
    <xsd:complexType name="LiteralNull">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="value" type="u2tp:InstanceValue"/>
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
        <xsd:attribute name="value" type="xsd:string"/>
```

```
</xsd:complexType>
<xsd:element name="LiteralNull" type="u2tp:LiteralNull"/>
<xsd:complexType name="ITimer">
    <xsd:choice maxOccurs="unbounded" minOccurs="0">
        <xsd:element name="isRunning" type="xsd:boolean"/>
        <xsd:element ref="xmi:Extension"/>
    </xsd:choice>
    <xsd:attribute ref="xmi:id"/>
    <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
    <xsd:attribute name="isRunning" type="xsd:boolean"/>
</xsd:complexType>
<xsd:element name="ITimer" type="u2tp:ITimer"/>
<xsd:complexType name="Timer">
    <xsd:choice maxOccurs="unbounded" minOccurs="0">
        <xsd:element ref="xmi:Extension"/>
    </xsd:choice>
    <xsd:attribute ref="xmi:id"/>
    <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
</xsd:complexType>
<xsd:element name="Timer" type="u2tp:Timer"/>
<xsd:complexType name="IArbiter">
    <xsd:choice maxOccurs="unbounded" minOccurs="0">
        <xsd:element ref="xmi:Extension"/>
    </xsd:choice>
    <xsd:attribute ref="xmi:id"/>
    <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
</xsd:complexType>
<xsd:element name="IArbiter" type="u2tp:IArbiter"/>
<xsd:complexType name="DataPool">
    <xsd:choice maxOccurs="unbounded" minOccurs="0">
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="dataPoolDefinition" type="xsd:string"/>
        <xsd:element name="selector" type="u2tp:DataSelector"/>
        <xsd:element ref="xmi:Extension"/>
    </xsd:choice>
    <xsd:attribute ref="xmi:id"/>
    <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="dataPoolDefinition" type="xsd:string"/>
    <xsd:attribute name="selector" type="xsd:string"/>
</xsd:complexType>
<xsd:element name="DataPool" type="u2tp:DataPool"/>
<xsd:complexType name="DataSelector">
    <xsd:choice maxOccurs="unbounded" minOccurs="0">
        <xsd:element name="name" type="xsd:string"/>
```

```
            <xsd:element name="dataSelectorDefinition"
type="xsd:string"/>
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="dataSelectorDefinition" type="xsd:string"/>
    </xsd:complexType>
    <xsd:element name="DataSelector" type="u2tp:DataSelector"/>
    <xsd:complexType name="DataPartition">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="dataPartitionDefinition"
type="xsd:string"/>
            <xsd:element name="selector" type="u2tp:DataSelector"/>
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="dataPartitionDefinition"
type="xsd:string"/>
        <xsd:attribute name="selector" type="xsd:string"/>
    </xsd:complexType>
    <xsd:element name="DataPartition" type="u2tp:DataPartition"/>
    <xsd:complexType name="IScheduler">
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
            <xsd:element ref="xmi:Extension"/>
        </xsd:choice>
        <xsd:attribute ref="xmi:id"/>
        <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
    </xsd:complexType>
    <xsd:element name="IScheduler" type="u2tp:IScheduler"/>
</xsd:schema>
```

Change the statement of compliance from

Proposed compliance points

The compliance points are as follows:

1. UML Profile for Testing: a compliant implementation supports the UML profiling mechanism, the UML entities extended by the Testing Profile, and the stereotyped entities of the UML Testing Profile.

2. MOF-based Metamodel for Testing: the compliant implementation supports all of the entities in the MOF-based metamodel.

3. Notation: If graphical notation is used, the compliant implementation recognizably supports the notation defined by the Testing Profile specification.

4. XMI/DTD: An XMI compliant implementation of the Testing Profile and/or MOF metamodel provides the UML XMI exchange mechanism.

5. Static Requirements: The compliant implementation checks the specified constraints automatically.

To

Proposed compliance points

The compliance points are as follows:

1. UML Profile for Testing: a compliant implementation supports the UML profiling mechanism, the UML entities extended by the Testing Profile, and the stereotyped entities of the UML Testing Profile.

2. MOF-based Metamodel for Testing: the compliant implementation supports all of the entities in the MOF-based metamodel.

3. Notation: If graphical notation is used, the compliant implementation recognizably supports the notation defined by the Testing Profile specification.

4. XMI: An XMI compliant implementation of the Testing Profile and/or MOF metamodel provides the XMI exchange mechanism using the Testing Profile XMI schema definitions.

Gelöscht: /DTD

Gelöscht: UML

5. Static Requirements: The compliant implementation checks the specified constraints automatically.

OMG Issue No: 6296

Title: Traces for test cases and test suite (Profile)

Source:

FOKUS, Ina Schieferdecker, schieferdecker@fokus.fraunhofer.de

Summary:

It should be possible to have traces for test suites and test cases. These traces should be defined comparable to the standalone metamodel traces.

Resolution:

Add test logs (formerly traces) to the profile definition by defining a test log to be a behavior and attaching it to a test context (formerly test suite) or a test case.

Revised Text:

Add the following text to the behavior part of U2TP:

At the beginning:

TestLog

Both a test context and a test case may trace their executions. These traces are behaviors in general. They can be recorded as test logs and become part of the test specification. A test log has to be attaced to a test context or a test case such that it is specified from which test context or test case that test log has been taken.

Add the figure to the profile metamodel figures:

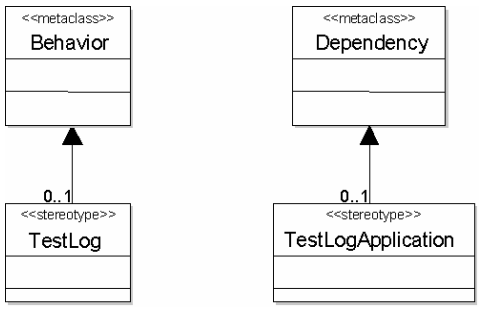


Figure xxx: Test Logs

Within the metaclass definitions:

TestLog

Description

Extends behavior. A test log represents the behavior resulting from the execution of a test case or a test context. Also, it helps to understand potential actions and validations performed by the test context behavior which might impact the test case verdict. A test case or a test context may have any number of test logs.

Constraints

No constraints for TestLog defined.

Notation

A test log is a behavior and has no special notation.

TestLogApplication

Description

A dependency to a test case or a test context.

Constraints

The client of a test log application must be a named element with a test case or test context stereotype applied.

The supplier of a test log must be a named element with a test log stereotype applied.

Notation

The notation for a test log is identical to a comment, i.e. a rectangle with a bent corner, with the keyword testlog.

Examples

See example in Figure 2-8. ATMSuite_log is a test log of the test context ATMSuite. invalidPIN_log is a test log of the test case invalidPIN.

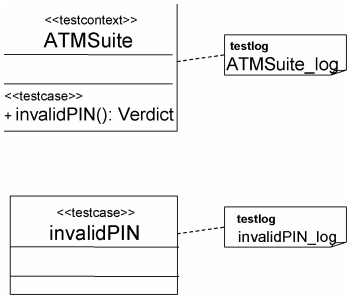


Figure xxx: Example on TestLog

OMG Issue No: 6298

Title: Data Pools and Data Partitions (Profile and Standalone Model)

Source:

IBM, Serge Lucio, slucio@us.ibm.com

Summary:

The notions of data pool and data partition are used throughout the document but explained only in the terminology section. In the profile, it is argued that data pools and data partitions are modeled by use of standard UML 2.0 concepts, but it is not explained how. In the standalone model, both are missing completely. Hence, data pool and data partition should be added to the standalone model. In addition, a standard way of representing data pools and data partitions should be defined for the profile (by defining stereotypes for data pools and data partitions).

Resolution:

Data pools and data partitions are to be added to the profile and the standalone model.

Revised Text:

Modify the glossary section from

Data Pool

A data pool is a collection of values. It is used by test components as a source of values for the execution of test cases. Data pools can be represented by utility parts or be logically described by constraints.

Data Partition

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

To

Data Pool

A data pool is a collection of data partitions or explicit values that are used by a test context, or test components, during the evaluation of test contexts and test cases. In doing so, a data pool provides a means for providing values or data partitions for repeated tests.

Data Partition

A logical value for a parameter used in a stimulus or in an observation. It typically defines an equivalence class for a set of values, e.g. valid user names etc,.

Data Selector

An operation that defines how data values or equivalence classes are selected from a data pool or data partition.

Modify the Test Data section from

The Test Data section contains concepts additional to UML data concepts needed to describe test data. It covers wildcards for a flexible specification of test data and coding rules for the specification of test data transmission.

Coding Rules

Coding rules are shown as strings referencing coding rules defined outside the Testing Profile such as by ASN.1, CORBA or XML. Coding rules are basically applied to value specification to denote the concrete encoding and decoding for these values during test execution. They can also be applied to properties and namespaces in order to cover all involved values of the property and/or namespace at once.

Wildcards

Wildcards are literals and denote an omitted value, any value or any value or omit. These literals can be used wherever value specifications can be used. They are typically used for a loose specification of data to be expected from the SUT or provided to the SUT.

UML 2.0 provides LiteralNull, which is used by the Testing Profile for the representation of omitted values. Wildcards for any value (LiteralAny) and any value or omit (LiteralAnyOrNull) are extensions to UML 2.0.

To

The Test Data section contains concepts additional to UML data concepts needed to describe test data. It covers: wildcards for a flexible specification of test data, data pools, data partitions, data selection, and coding rules for the specification of test data transmission.

Wildcards

Wildcards are literals and denote an omitted value, any value or any value or omit. These literals can be used wherever value specifications can be used. They are typically used for a loose specification of data to be expected from the SUT or provided to the SUT.

UML 2.0 provides LiteralNull, which is used by the Testing Profile for the representation of omitted values. Wildcards for any value (LiteralAny) and any value or omit (LiteralAnyOrNull) are extensions to UML 2.0.

Data Pool

Test cases are often executed repeatly with different data values to stimulate the SUT in various ways. Also when observing data, abstract equivalence classes are used to defined sets of allowable values. Typically these values are taken from data partitions, or lists of explicit values. For this purpose a data pool provides a means for associating data sets with test contexts and test cases. A data pool is a classifier containing either data partitions (equivalence classes), or explict values; and can only be associated with either a test context or test components.

Data Partition

A data partition is used to defined an equivalence class for a given type e.g. “ValidUserNames” etc. By denoting the partitioning of data explicitly we provide a more visible differentiation of data. A data partition is a stereotyped classifier that must be associated with a data pool.

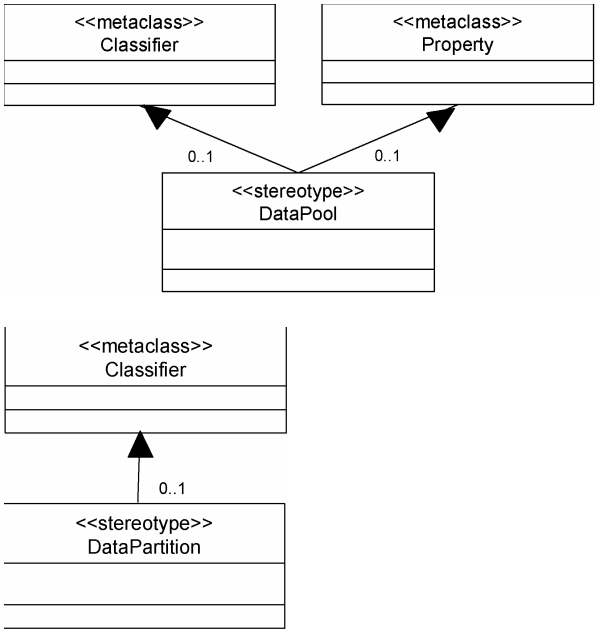
Data Selector

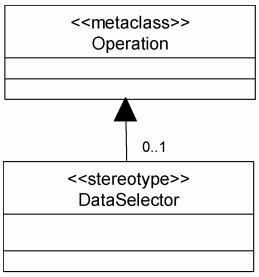
To facilitate the different data selection strategies and data checking one or more data selectors can be associated with either a data pool or data partition. Typically, these data selectors are operations that operate over the contained values or value sets.

Coding Rules

Coding rules are shown as strings referencing coding rules defined outside the Testing Profile such as by ASN.1, CORBA or XML. Coding rules are basically applied to value specification to denote the concrete encoding and decoding for these values during test execution. They can also be applied to properties and namespaces in order to cover all involved values of the property and/or namespace at once.

Extend the Test Data figure with the new metaclasses





Add the stereotype definitions

DataPool

Description

The data pool stereotype extends a classifier or property to specify a container for explicit values or data partitions that are used by test contexts or test cases. A data pool provides an explicit means for associating data values for repeated tests (e.g. values from a database etc.), and equivalence classes that can be used to defined abstract data sets for test evaluation.

Constraints

A data pool stereotyped classifier can only be referenced by a test context or test component.

A data pool stereotyped property can only be applied to a property associated connected with a test component within the context of a test context stereotyped classifier.

A datapool stereotyped classifier cannot be associated with both a test context and a test component.

Semantics

The semantics of a data pool are given by the classifier, and contained data partitions, that realises it.

Notation

The notation for data pool is a classifier with stereotype <<DataPool>>

Examples

A data pool example is given in XXX.

DataPartition

Description

The data partition stereotype extends a classifier to specify a container for a set of values. These data sets are used as abstract equivalence classes during test context and test case evaluation.

Constraints

A data partition stereotyped classifier can only be associated with a data pool or another data partition.

Semantics

The semantics of a data partition are given by the classifier that realises it.

Notation

The notation for data partition is a classifier with stereotype <<DataPartition>>

Examples

A data partition example is given in XXX

DataSelector

Description

The data selector stereotype extends an operation to allow the implementation of different data selection strategies.

Constraints

If a data selector stereotype is applied to an operation, the featuring classifier must have either a data pool or data partition stereotype applied.

Semantics

The semantics of a data selector are given by the behaviour that realises it.

Notation

The notation for data selector is an operation with stereotype <<DataSelector>>

Examples

A data selector example is given in XXX

Add an example to the example section.

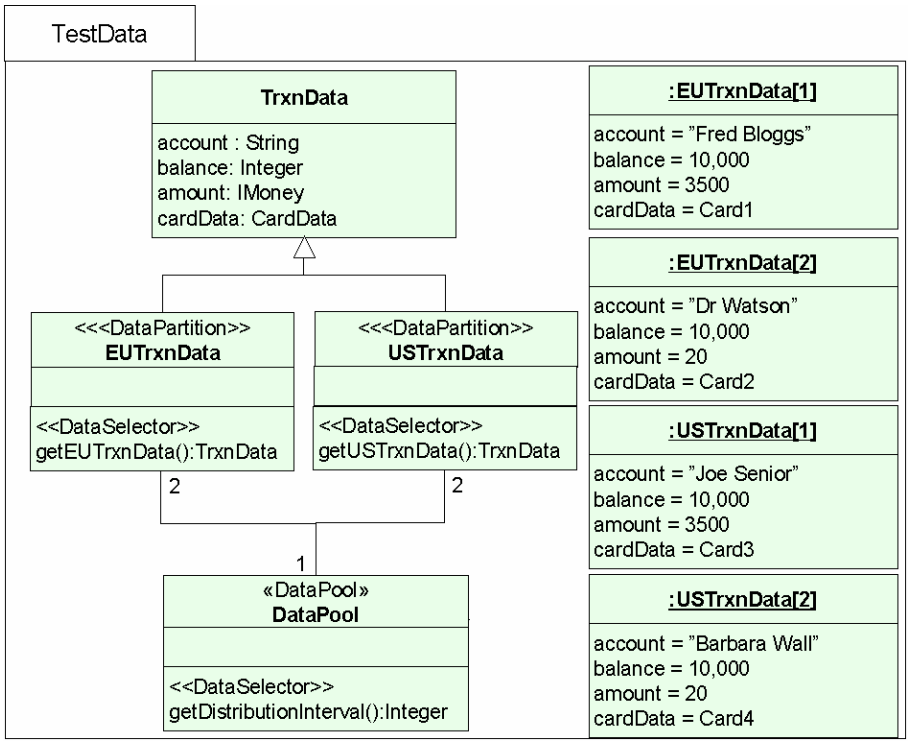
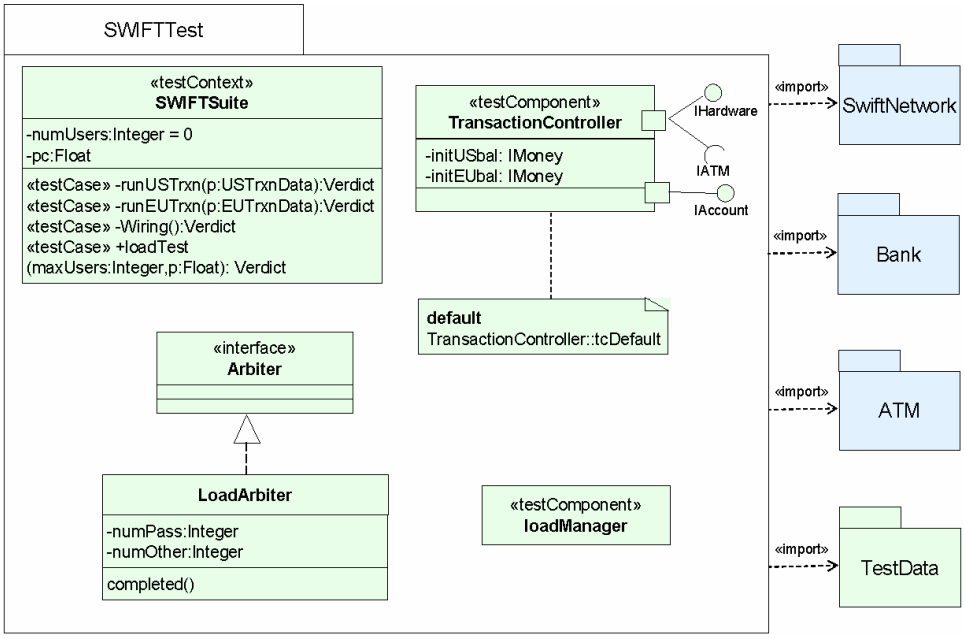
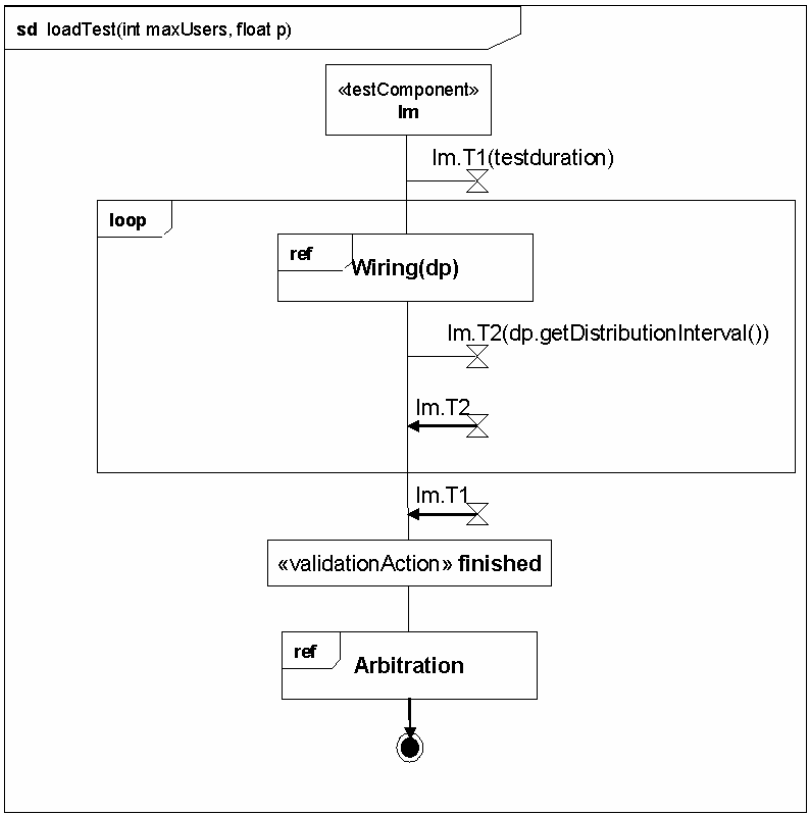


Figure xxx is a package illustrating the data pool, data partition and data selector concepts. The TestData package defines for TrxnData the data pool DataPool and the data partitions EUTrxnData and USTrxnData. The data partitions have two data samples defined each. Data selectors getEUTrxnData, getUSTrxnData and getDistributionInterval are used for the access to the data pool and the data partitions.

Modify the load test package (Figure 36)



Modify the Main Test Behavior (Figure 39) to



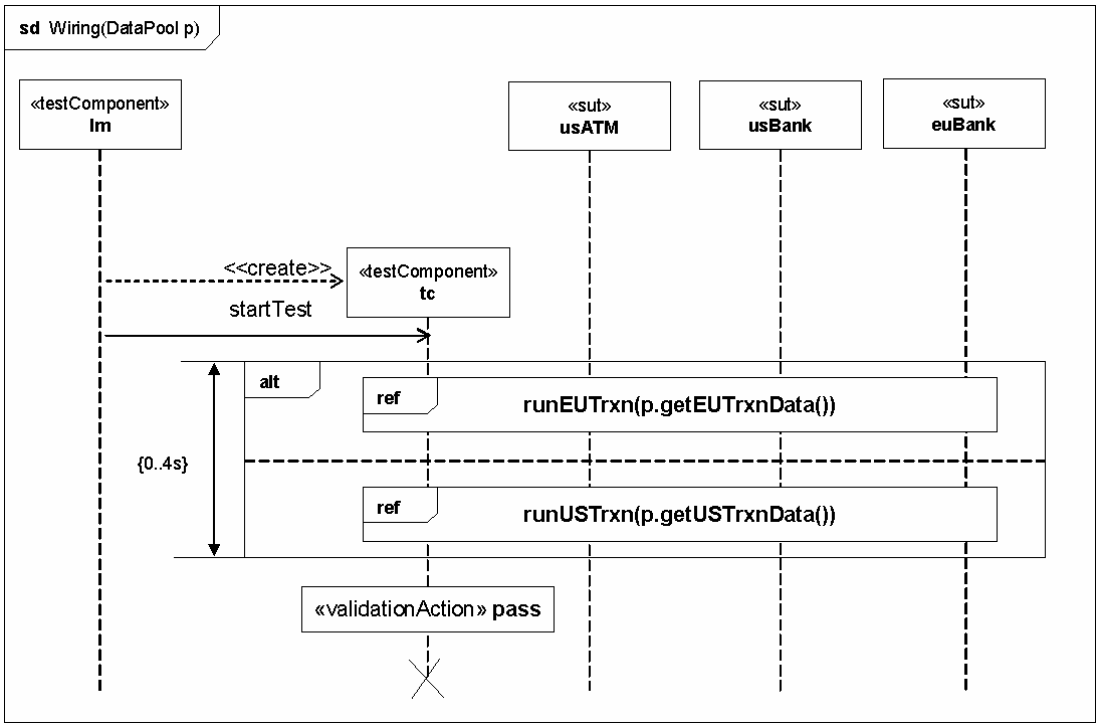
Change the explanatory text from

Figure 39 and Figure 41 illustrate the behavior of the loadTest method of the TestContext element. The test is invoked with three parameters: the maximum number of virtual users that are simulated at any one time, the percentage of wiring transactions that should be successful, and the data pool reference. Given these parameters, the system launches the appropriate number and types of test cases and monitors their outcomes. When the load test has been running for the defined duration the load arbiter is informed (via the arbitration reference) that the test is completed and that the final test case verdict should be calculated

To

Figure 39 and Figure 40 illustrate the behavior of the loadTest method of the TestContext element. The test is invoked with two parameters: the maximum number of virtual users that are simulated at any one time and the data pool reference. Given these parameters, the system launches the appropriate number and types of test cases and monitors their outcomes. The load manager lm is used as a gneration. At first, the total test duration is set by a timer. Then, the first Wiring is started. Following the start of the Wiring test scenario, another timer is started which is used to set the duration between generations when the second timer expires. This will loop as long as the test duration expires. Then, the loop is left. The load manager applies a special validation action value and the arbitration is entered to calculate the final test case verdict (see Figure 41).

Modify the US initiated wiring transaction figure (figure 40):



Change the explanatory text from

Figure 40 illustrates how an individual US wiring transaction is executed. Firstly the test case variable numUsers is incremented then an instance of the transaction controller is created and started. The timing constraint stipulates that the execution of the transaction should take less than 4 seconds, otherwise the default handler will inform the Load Arbiter that the transaction has failed. If the transaction is successful the numUsers variable is decremented, the Load arbiter is informed that the transaction was successful, and the transaction controller is terminated.

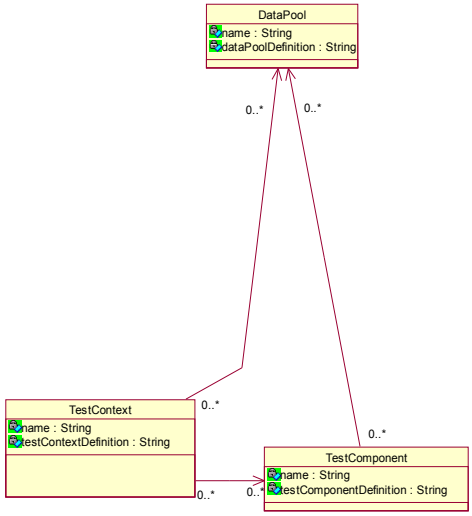
To

Figure 40 illustrates how an individual US wiring transaction is executed. The behavior is parameterized with the data pool dp. Firstly, an instance of the transaction controller is created and started. The the test is started and a transation is started using either data taken from the getEUTrxnData data partition or from the getUSTrxnData partition. The timing constraint stipulates that the execution of the transaction should take less than 4 seconds, otherwise the default handler will inform the Load Arbiter that the transaction has failed.

Modify the Transaction detail (figure 42):

Modify the standalone model.

Add the data pool to the architecture:



Add the dataPool association to the TestContext and TestComponent stereotypes:

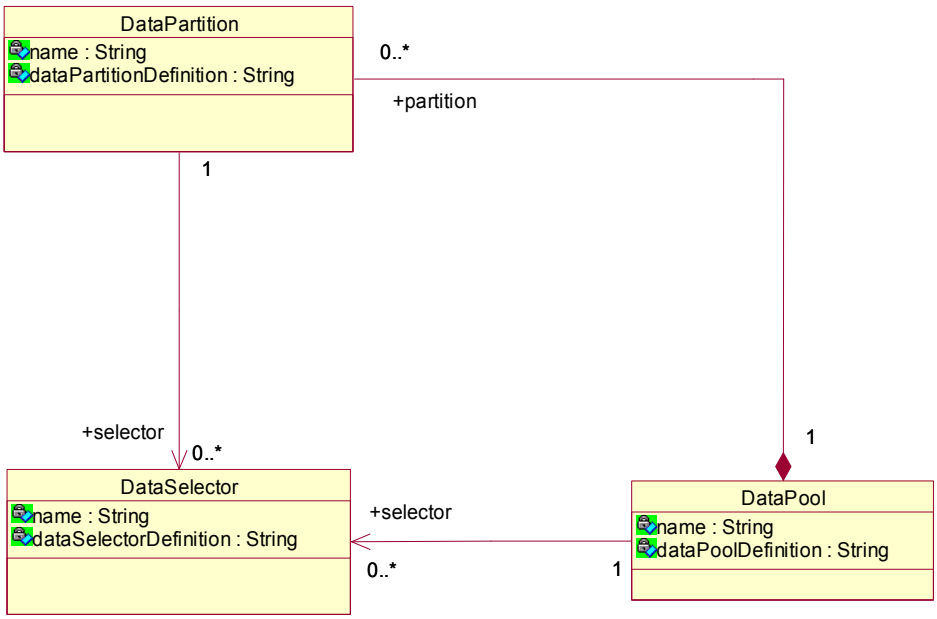
Associations in TestContext

dataPool:DataPool[0..*]	Data pools associated to the test context.

Associations in TestComponent

dataPool:DataPool[0..*]	Data pools associated to the test component.

Add the data pool, data partition and data selector to the data part:



Add the data pool, data partition and data selector stereotype definitions

DataPool

Semantics

Zero or more data pools can be associated to test contexts or test components. Data pools specify a container for explicit values or data partitions. They provide an explicit means for associating data values for repeated tests (e.g. values from a database etc.), and equivalence classes that can be used to defined abstract data sets for test evaluation

Associations

partition:DataPartition[0..*]	A set of data partitions defined for this data pool.
selector:DataSelector[0..*]	A set of data selectors defined for this data pool.

Attributes

name: String [1]	The name of the data pool.
dataPoolDefinition: String [1]	The definition of the data pool.

DataPartition

Semantics

Zero or more data partition can be defined for a data pool. A data partition is a container for a set of values. These data sets are used as abstract equivalence classes within test context and test case behaviors.

Associations

selector:DataSelector[0..*]	A set of data selectors defined for this data partition.

Attributes

name: String [1]	The name of the data partition.
dataPartitionDefinition: String [1]	The definition of the data partition.

DataSelector

Semantics

Zero or more data selectors can be defined for data pools or data partitions. Data selectors allow the definition of different data selection strategies.

Attributes

name: String [1]	The name of the data selector.
dataSelectorDefinition: String [1]	The definition of the data selector.

Add to the JUnit Mapping

Data pool	A class together with operations to get access to the data pool.
Data partition	A class (inheriting from a data pool) together with operations to get access to the data partition.
Data selector	An operation of a data pool or a data partition.

Add to the TTCN-3 Mapping

Data pool	An external constant (referring to the data in the data pool) or external functions to get access to the data pool.
Data partition	TTCN-3 matching mechanisms can be used to handle data partitions for observations. For stimuli however, user defined functions are needed to realize the test case execution with different data to be sent to the SUT.
Data selector	An external function to get access to the data of a data pool or data partition.

OMG Issue No: 6299

Title: Data Picker/Data Selection (Profile and Standalone Model)

Source:

IBM, Serge Lucio, slucio@us.ibm.com

Summary:

In testing, typically test cases are executed with different test data. For that, testdata is selected along different selection strategies. However, currently test data selection is not a separate concept in the UML testing profile.
The central point is the separation of test data specification and test selection for test execution. An additional interface to plug in different test selection strategies could be a solution.

Resolution:

Add data selectors to the U2TP profile and standalone model.

Revised Text:

See 6298.

OMG Issue No: 6301

Title: Relation of Test Suite and Arbiter (Profile)

Source:

IBM, Serge Lucio, slucio@us.ibm.com

Summary:

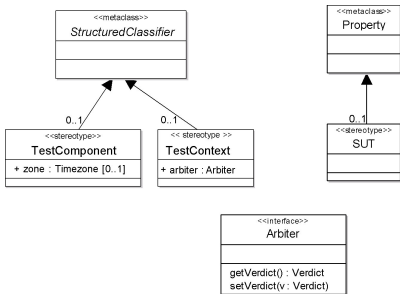
The definition for a test suite in the profile has a condition: “A test suite must contain exactly one property realizing the Arbiter interface.” This gives a constraint on the metamodel. The property is not named.

Resolution:

Add an arbiter attribute to test context.

Revised Text:

Revise Figure 1 to



Add the definition for the arbiter attribute to the TestContext stereotype.

Attributes

arbiter : Arbiter [1] Realizes the arbiter interface.

OMG Issue No: 6302

Title: Load Tests (Profile)

Source:

Motorola, Paul Baker, paul.baker@motorola.com

Summary:

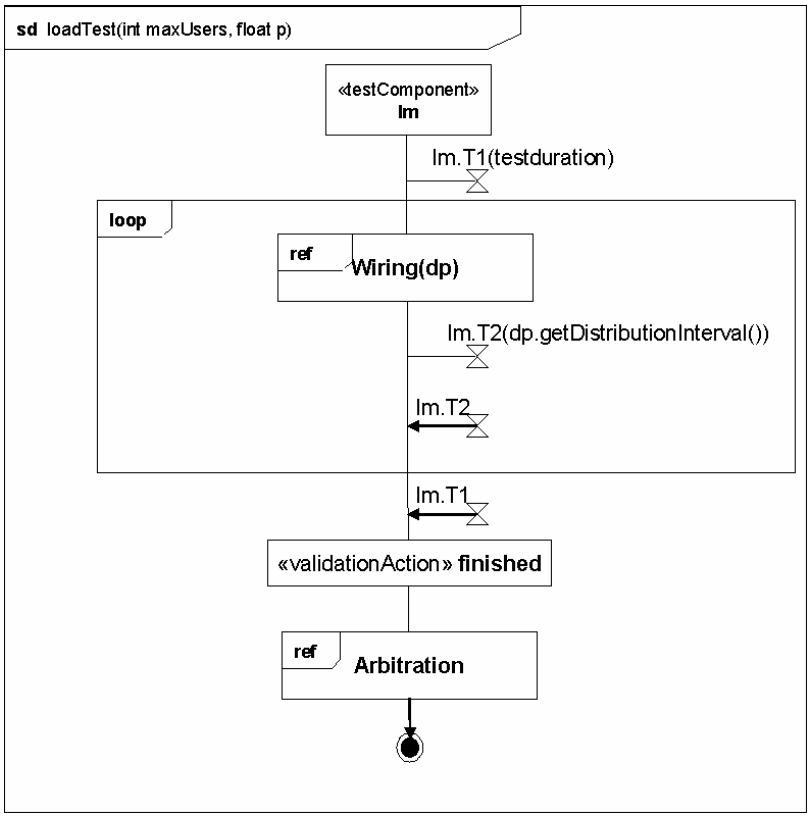
The definition of load tests is cumbersome. There are no high-level operators to enable the parallel and quasi-simultaneous execution of test components – following a certain distribution function and realizing different test behaviors. Currently, one has to define e.g. a separate interaction to spawn the test components like in the example given in the U2TP specification.

Resolution:

An explicit scheduling component is added – this component coordinates the creation/termination of test components. It is a mean to integrate also generation aspects into a U2TP specification.

Revised Text:

Modify the Main Test Behavior (Figure 39) to



Change the explanatory text from

Figure 39 and Figure 41 illustrate the behavior of the loadTest method of the TestContext element. The test is invoked with three parameters: the maximum number of virtual users that are simulated at any one time, the percentage of wiring transactions that should be successful, and the data pool reference. Given these parameters, the system launches the appropriate number and types of test cases and monitors their outcomes. When the load test has been running for the defined duration the load arbiter is informed (via the arbitration reference) that the test is completed and that the final test case verdict should be calculated

To

Figure 39 and Figure 40 illustrate the behavior of the loadTest method of the TestContext element. The test is invoked with two parameters: the maximum number of virtual users that are simulated at any one time and the data pool reference. Given these parameters, the system launches the appropriate number and types of test cases and monitors their outcomes. The load manager lm is used as a gneration. At first, the total test duration is set by a timer. Then, the first Wiring is started. Following the start of the Wiring test scenario, another timer is started which is used to set the duration between generations when the second timer expires. This will loop as long as the test duration expires. Then, the loop is left. The load manager applies a special validation action value and the arbitration is entered to calculate the final test case verdict (see Figure 41).

See also issue 6298.

OMG Issue No: 6306

Title: Arbiter Semantics

Source:

U2TP Consortium, u2tp@fokus.fraunhofer.de

Summary:

The current specification says nothing about the semantics of an arbiter for example how the arbiter gets to know that a test case finished and that the arbiter has to calculate the final verdict.

Resolution:

Explicit arbiter protocols will be added. They show how the arbiter works in combination with the scheduler.

Revised Text:

See 6302.

OMG Issue No: 6307

Title: Synchronization/Coordination of Test Components

Source:

U2TP Consortium, u2tp@fokus.fraunhofer.de

Summary:

The current specification does not offer high-level concepts for functional synchronization and coordination between test components like rendezvous, joint start and joint termination.

Resolution:

A scheduling component is to be added. It allows to control the coordination for test component creation and termination. Other high-level concepts for functional synchronization will not be provided.

Revised Text:

Add to the terminology:

> **Scheduler:** A property of a test context used to control the execution of the different test components. The scheduler will keep information about which test components exist at any point in time, and it will collaborate with the arbiter to inform it when it is time to issue the final verdict. It keeps control over the creation and destruction of test components and it knows which test components take part in each test case.

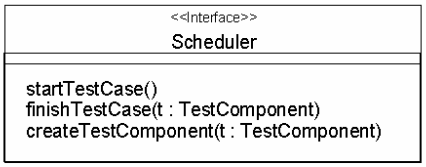
Add to the architectural section of the profile:

Scheduler

Scheduler is a predefined interface provided with the Testing Profile. The purpose of a scheduler implementation is to control the execution of the different test components. The scheduler will keep information about which test components exist at any point in time, and it will collaborate with the arbiter to inform it when it is time to issue the final verdict. It keeps control over the creation and destruction of test components and it knows which test components take part in each test case.

Every test context must have an implementation of a scheduler. Any tool must provide such an implementation which will be used if there are no explicit realization defined in the test context.

Add to the architecture figure (Figure 1):



Scheduler (a predefined interface)

Description

Scheduler is a predefined interface defining operations used for controlling the tests and the test components. None of the operations of the Scheduler is available to the UML specifier.

Operations

Scheduler() : The constructor of Scheduler. It will start the SUT and the Arbiter.

startTestCase() : The scheduler will start the test case by notifying all involved test components.

finishTestCase(t:TestComponent) : Records that the test component t has finished its execution of this test case. When all test components involved in the current test case have finished, the arbiter will be notified.

createTestComponent(t:TestComponent) : Records that the test component t has been created by some other test component.

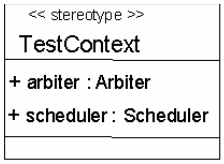
Semantics

The implementation of the predefined interface will determine the detailed semantics. The implementation must make sure that the scheduler has enough information to keep track of the existence and participation of the test compoenents in every test case. The test context itself will ensure the creation of a scheduler.

Examples

Example protocols for how the scheduler could work is found in annex XXX.

Add a scheduler attribute to the test context:



Attributes

arbiter : Arbiter [1] Realizes the arbiter interface.

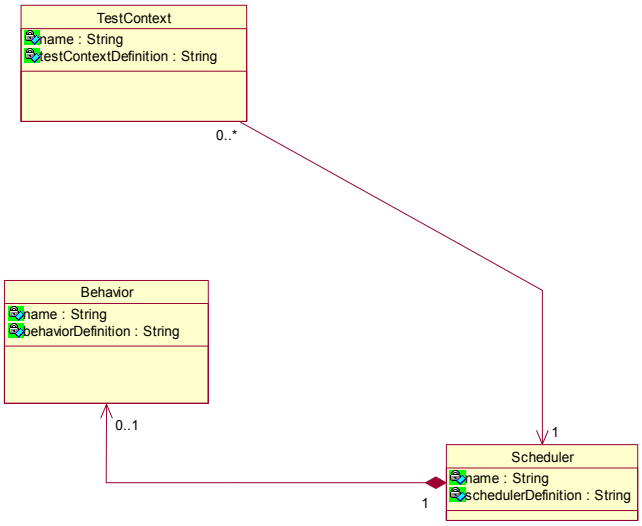
scheduler : Scheduler [1] Realizes the scheduler interface.

Constraints

A test context must contain exactly one property realizing the Arbiter interface.

A test context must contain exactly one property realizing the Scheduler interface.

Add to the standalone model



Scheduler

Semantics

A scheduler is an entity within a test context which controls the running of the test cases. It will keep track of the creation and destruction of test components and give instructions to the existing test components when to start executing a given test case. It will communicate with the arbiter when the time is right to produce the verdict for a test case.

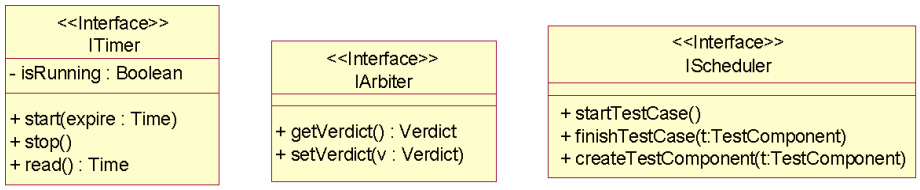
Associations

behavior:Behavior[1]	The behavior of the scheduler.

Attributes

name: String [1]	The name of the scheduler.
schedulerDefinition: String [1]	The definition of the scheduler (in addition to the behavior definition of the scheduler).

Add the scheduler interface to the standalone model (figure 16):

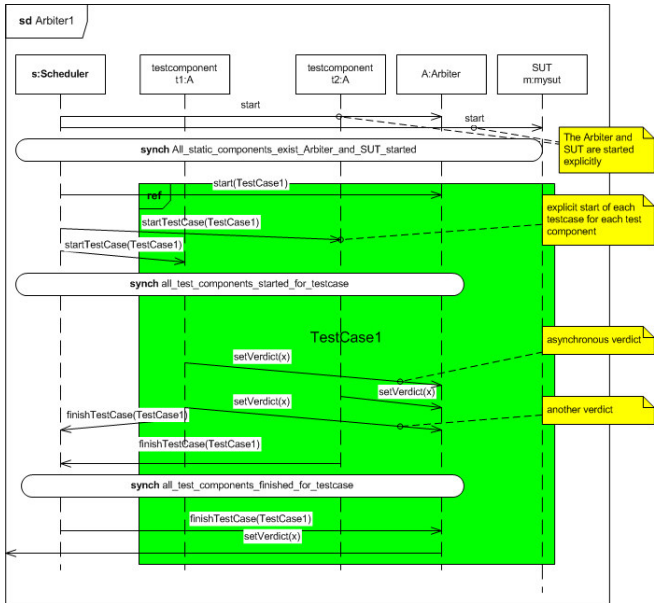


Add an annex with the scheduler and arbiter protocols.

Arbiter and Scheduler protocols

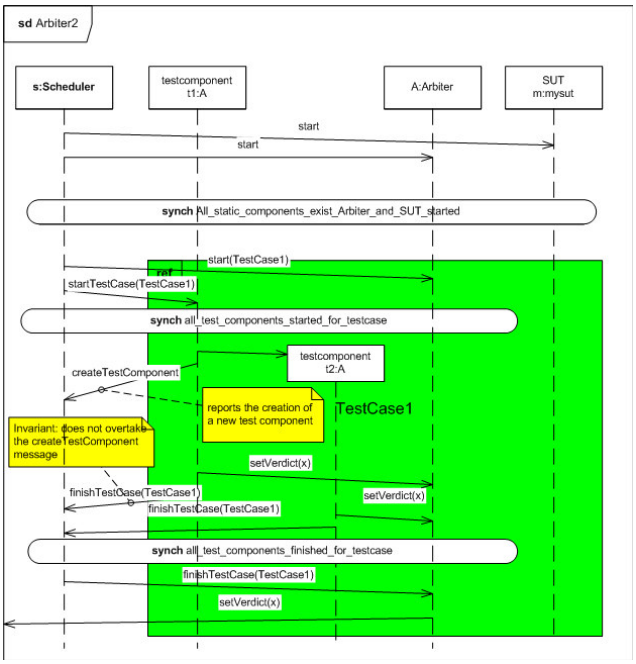
This annex shows how the Scheduler and the Arbiter should work together with the test components and the SUT such that the execution of the test cases are performed properly and the verdict delivered at the right time. Itis not necessary that a valid implementation conforms in detail to these sequence diagrams, but the net effect should be the same.

In the diagrams we have in addition to standard notation also applied a special kind of continuations that we have called synchronous continuations denoted by the prefixing “synch” keyword. This is a shorthand for introducing a number of general ordering relations such that all events above the synchronous continuation precedes all events below the synchronous continuation. This holds for all events on those lifelines that are covered by the continuation.



Scheduler/Arbiter protocol #1

The protocol scenario in Figure 55 shows the normal situation when the test components are static. Notice that the Scheduler starts the SUT and the Arbiter explicitly before any test case can be started. When a test case is initiated the Scheduler will give notice to the test components that are involved in it. They will in turn notify the Scheduler when they are at the end of executing that test case. None of this is seen explicitly in the user specifications. Finally when the Scheduler has recorded that no test components have pending results, the Arbiter will be asked to produce the final verdict for the test case.



Scheduler/Arbiter protocol #2

The scenario in Figure 56 shows what should happen when test components are created dynamically within the execution of the test case. The test component that creates another test component will notify the Scheduler about the creation. Notice that the later notification from that test component that it is finished with the test case must go along the same communication channel as the creation notification. This is to avoid possible race conditions between the creation notification and the finishing notifications. Such race ondition would have made it theoretically possible to create a situation where the Scheduler knows about no pending test components, while the newly created test component is still running. The Arbiter could therefore have been instructed to give final verdict before it should.

In case of test component termination (destruction) this must also be notified to the Scheduler by the test component. It is assumed that the test component being destroyed is able to transmit its last verdict to the Arbiter before it is deleted.

In some test cases not all eisting test components will take part. It is assumed that the Scheduler has proper information about this from its description of the test case such that it will not initiate more test components than necessary for a particular test case.

Add to the JUnit Mapping

Scheduler	The scheduler can be realized as a property of Test Context. There should be a default scheduler along the protocol defined in Appendix C.

Add to the TTCN-3 Mapping

Scheduler	There is a default scheduler built in TTCN-3. User defined schedulers can be realized by the MTC.

OMG Issue No: 7193

Title: Reference to TTCN-3

Source:

University of Göttingen, Jens Grabowski, grabowski@informatik.uni-goettingen.de

Summary:

Technical references to TTCN-3 should not be part of the U2TP specification

Resolution:

Remove footnote

> The Testing Profile therefore has a different approach to defaults than TTCN-3 where the defaults are dynamically included and excluded.

Remove also the mentioning of MSC in the main text: remove

> The notations are based on the MSC timer notation.

OMG Issue No: 6303

Title: Activity Diagrams (Profile)

Source:

U2TP Consortium, u2tp@fokus.fraunhofer.de

Summary:

The current specification of the UML Testing Profile does not address the specification of test behaviors with activity diagrams, although users see a need to have also activity diagrams been supported by the testing profile.

Resolution:

Accommodate the testing profile specific behavioral concepts in activity diagrams being

- Defaults and default applications
- Validation action
- Finish action
- Timers

Revised Text:

Change in the profile behavior from

The Testing Profile has chosen to associate the default applications to static behavioral structures. In Interactions we may apply defaults to interaction fragments, and in State Machines to StateMachines, States or Regions. Since each default applies only to one test component, we attach the defaults on interaction fragments to the intersection between the fragment and the test component.

We said above that default behavior is invoked when the main description cannot describe the observed behavior. More precisely the default mechanism is invoked when a trigger (or message reception) is not defined by the main description or an explicit runtime constraint is violated. The point of default invocation will be well-defined as an event occurrence in Interactions or a State (-stack) in State Machines.

To

The Testing Profile has chosen to associate the default applications to static behavioral structures. In Interactions we may apply defaults to interaction fragments, in State Machines to StateMachines, States or Regions, and in Activties to Actions and Activities. Since each default in an Interaction applies only to one test component, we attach the defaults on interaction fragments to the intersection between the fragment and the test component.

We said above that default behavior is invoked when the main description cannot describe the observed behavior. More precisely the default mechanism is invoked when a trigger in a State Machine or message reception in an Interaction or an action in an Activity is not defined by the main description or an explicit runtime constraint is violated. The point of default invocation will be well-defined as an event occurrence in Interactions or a State (-stack) in State Machines or an accept event action in Activities.

Change Default Semantics from

Semantics

We describe the semantics of defaults differently for Interaction and State Machines since UML itself describes the semantics of these concept in different terms.

To

Semantics

We describe the semantics of defaults differently for Interaction, Activities and State Machines since UML itself describes the semantics of these concepts in different terms.

Add to the default semantics section

For defaults that are described on Activities, we consider the actions of the Activity together with the actions of the Default. The simple rule to combine the default with the main description is that the result is the union of all actions, but such that initial actions being part of the Default can only occur (and by doing so triggering the execution of that default) if there are no equal initial accept event in the Activity where the default is attached to.

Change default semantics from

We may have hybrid defaults where a default may be applied within an Interaction, but defined as a State Machine. In such cases the default State Machine is considered equivalent to the set of traces that it can produce (or said differently, the strings that the automata may accept).

If there is no user-defined default that applies, what happens is a Semantic Variation Point.

To

We may have hybrid defaults where a default may be applied within an Interaction, but defined as a State Machine. In such cases the default State Machine is considered equivalent to the set of traces that it can produce (or said differently, the strings that the automata may accept). We may have also hybrid defaults where a default may be applied within an Interaction or State Machine, but defined as an Activity. In such cases the default Activity is considered equivalent to the set of traces the Activity can produce (along the Petri-net like semantics of Activities).

If there is no user-defined default that applies, what happens is a Semantic Variation Point.

Add

The Semantic Variation Point may have the following interpretations or other interpretations decided by the tool vendors.

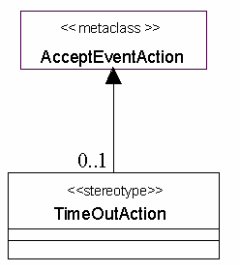
…

The activity may conclude.

Define notation for FinishAction in Activity diagrams:

In Activity diagrams, the FinishAction is shown as a black quadrat.

Add TimeOutAction to the time section



TimeOutAction

Description

Extends AcceptEventAction. A timeout is enabled by a timer when it expires. An activity having the TimeOutAction as input condition occurs, when the timeout is enabled (and when all further input conditions are satisified).

Notation

The notation for the TimeOutAction is an empty hourglass (it reuses the syntax for the accept time event action in activities). An arrow with an unfilled head connects the hourglass and the activity to which the timeout is an input condition.

Semantics

A timeout is enabled when the timer expires. It may trigger an associated activity. The TimeOutAction occurs, when all input conditions for that activity are satisfied (including the TimeOutAction).

OMG Issue No: 6304

Title: Issues with the Load Testing Example

Source:

U2TP Consortium, u2tp@fokus.fraunhofer.de

Summary:

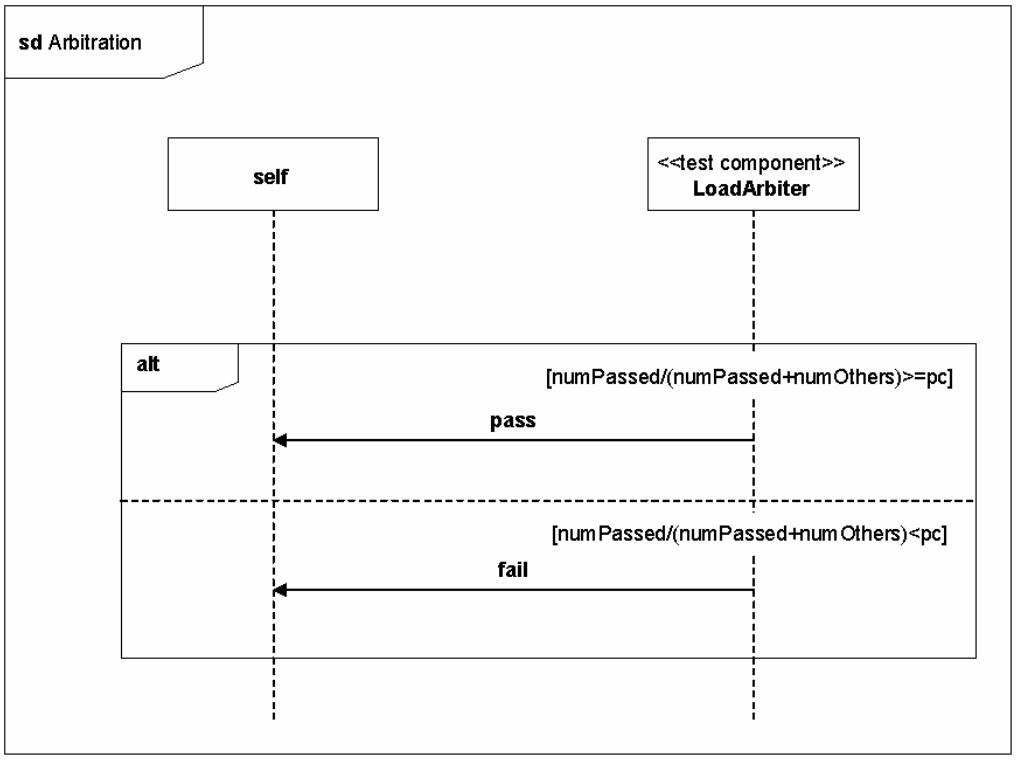
The arbitration interaction referenced in Fig. 36 is not shown.
The diagram in Fig. 39 defines a state machines and not an interaction. Hence, the sd in the header should be deleted.

Resolution:

Just remove sd from the diagram as proposed.

Revised Text:

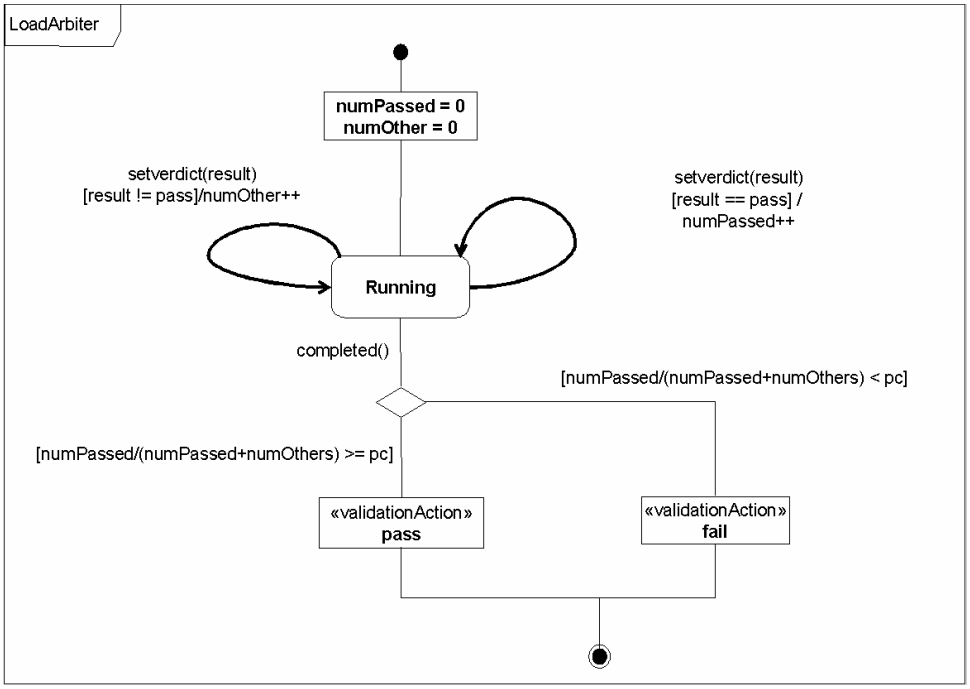
Add an arbitration figure



an explanatory text

The Arbitration behavior is given in Figure 41. A pass verdict is assigned if the number of successful tests exceeds the given threshold pc which is an attribute of the test context. Otherwise fail is assigned.

Update the LoadArbiter figure to



OMG Issue No: 6954

Title: Constrained semantics for UML constructs

Source:

Motorola, Paul Baker, paul.baker@motorola.com

Summary:

Introduction of an ordered alt operator ("if-then-else") operator.

Resolution:

Add the Deterministic Alternative as a new operator of CombinedFragment in Interactions.

Add to Utilities in the Behavior section of the profile:

Utilities

The Testing Profile has defined a few action utilities that may come in handy when defining test behavior.

determAlt is an interaction operator and is a shorthand for an Alternative where the operands are evaluated in sequence such that it is deterministic which operand is chosen given the value of the guards, regardless of the fact that the guard for more than one operand may be true.

Add

determAlt (an interaction operator)

Description

The deterministic alternative is a shorthand for an Alternative where the operands are evaluated in sequence such that it is deterministic which operand is chosen given the value of the guards, regardless of the fact that the guard for more than one operand may be true

Semantics

Textually in prefix notation the definition is as follows:

determAlt([guard1]op1) = **alt**([guard1]op1)

determAlt([guard1]op1, [guard2]op2) = **alt**([guard1]op1, [**else**] **determAlt**([guard2]op2))

In general

determAlt([guard1]op1, [guard2]op2, ..., [guardn]opn) =

alt([guard1]op1, [**else**]**determAlt**([guard2]op2,...,[guardn]opn))

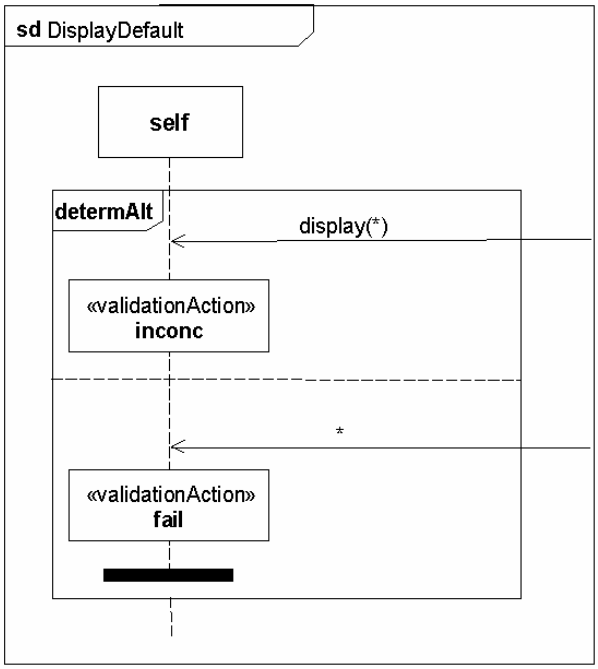
Notation

The determAlt uses the notation for combined fragments in interactions: the determAlt keyword is used in the small compartment in the upper left corner of the CombinedFragment frame.

Examples

An example determAlt operator can be found in Figure 29.

Change the example in Figure 29 to



Change the explanatory text from

Figure 29 specifies the DisplayDefault, a default for the reception of the *display("Transaction accepted")* message in the *validWiring* test case. The DisplayDefault describes what happens when a different message is received.

To

Figure 29 specifies the DisplayDefault, a default for the reception of the *display("Transaction accepted")* message in the *validWiring* test case. The DisplayDefault describes what happens when a different message is received. An inconc verdict is assigned if a display message is received with a parameter different to the expected one. Otherwise, fail is assigned and the test component finishes.

OMG Issue No: 7104

Title: Default/state machine syntax

Source:

FOKUS, Ina Schieferdecker, schieferdecker@fokus.fraunhofer.de

Summary:

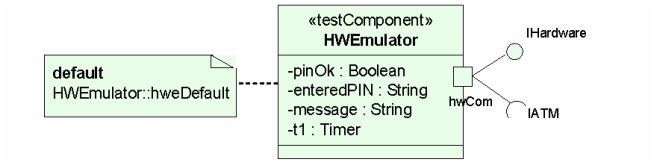
Wrong syntax in some figures: Fig. 28 and Fig. 29 use a syntax that have not been defined by U2TP

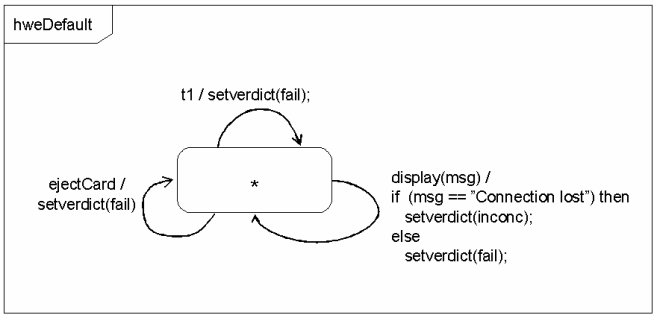
The special comment declares a default - its behavior is given in a behavioral diagram.

Hence the right rectangular box in Fig. 28 is superfluous. Also, the definition of the statemachine for the default (in Fig. 28 and Fig. 29) should simply contain the default identifier - i.e. hweDefault - and not <<default>> statemachine hweDefault - both <<default>> and statemachine are superfluous?!

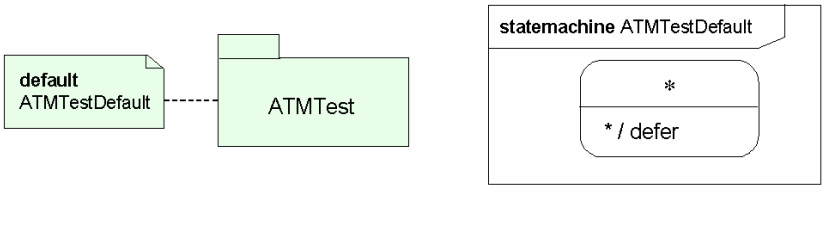
Resolution:

Change the figure on "A statemachine default applied to a test component" to

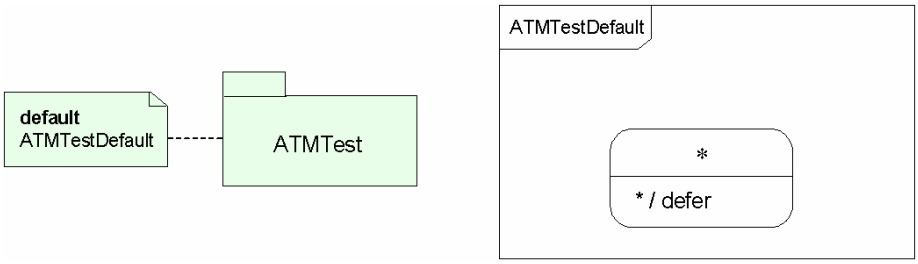




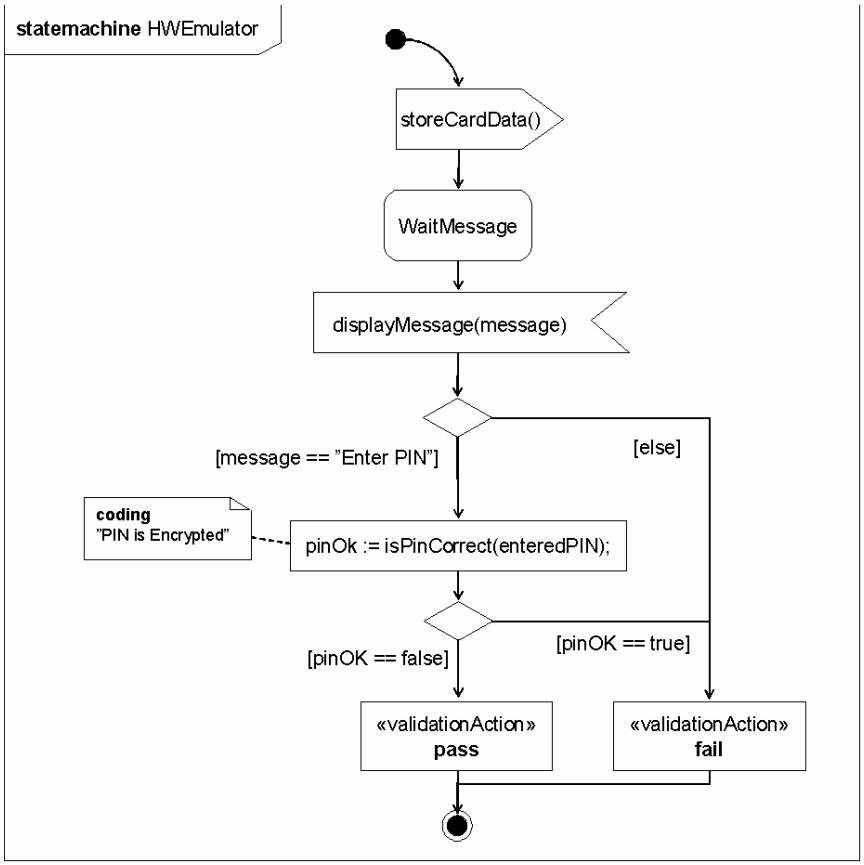
Change the figure on Package level default to



Change the figure on Package level default to



Change the figure on test component behavior to



Clarify also the notation for all other stereotypes of U2TP – even if it is the standard UML stereotype syntax by adding a notation section to all stereotypes in U2TP.

OMG Issue No: 7218

Title: Editorial comments

Source:

FOKUS, Ina Schieferdecker, schieferdecker@fokus.fraunhofer.de

Summary:

In the editing process for U2TP several editorial comments to be solved by the FTF have been made. These have to be addressed.

Resolution:

In Scope:

Remove

This proposed UML Testing Profile is in response to the Object Management Group Request For Proposal ad/01-07-08 on a UML Testing Profile.

Add

The UML Testing Profile extends UML with test specific concepts like test components, verdicts, defaults, etc. These concepts are grouped into concepts for test architecture, test data, test behavior and time. Being a profile, the UML testing profile seamlessly integrates into UML: it is based on the UML metamodel and reuses UML syntax.

In Normative References change to

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

UML 2.0 Infrastructure Specification

UML 2.0 Superstructure Specification

UML 2.0 OCL Specification

MOF 2.0 Specification

In Terms and Definitions: nothing to be changed

In References: change from Superstructure/Infrastructure final submission to final adopted specification.

Remove:

The following list identifies issues that could not be concluded by the consortium and will be considered in the FTF work:

- the use of activity diagrams for test behavior specifications. The consortium did not concluсde this work because of the late consolidation of activity diagrams within the UML 2.0.
- aspects of synchronization and coordination between test components such as the finish action. The consortium did not concluсde this work because of problems in the UML2.0 profiling mechanisms related to the definition of actions.
- aspects of test deployment and logs. The consortium did not concluсde this work because of the late consolidation of deployment concepts within the UML 2.0 specification.

Disposition: Unresolved

Disposition: Deferred

OMG Issue No: 6956

Title: Grey box testing

Source:

Motorola, Paul Baker, paul.baker@motorola.com

Summary:

Monitoring of interfaces between components within the SUT.

Discussion:

The suggestion is to add a stereotype <<Monitor>> that is a specialization of a port and to add a stereotype <<MonitorConnection>> that is a specialization of a connection. By these two it will be possible to seamlessly “tee” all events via the monitored port into an “observing” test component.
Although this concept is needed, it was felt to be rather complex and was therefore deferred to the second version of U2TP.

Disposition: **Deferred**

OMG Issue No: 6955

Title: Data guards on observations

Source:

Motorola, Paul Baker, paul.baker@motorola.com

Summary:

For each operand the guard has to be on each leading event within an alternative.

Discussion:

Although data guards on observations are needed in testing, the resolution within UML would impose major changes to the underlying semantics of interactions. Therefore, it is proposed to reconsider this issue in the second version of U2TP.

Disposition: **Deferred**

OMG Issue No: 7195

Title: **UML 2.0 Alignment**

Source:

FOKUS, Ina Schieferdecker, schieferdecker@fokus.fraunhofer.de

Summary:

U2TP has to be aligned with the finalized UML 2.0

Discussion:

The UML 2.0 FTF is not yet ready, but a final aligment of the testing profile with UML 2.0 will be needed. In discussion with the UML 2.0 FTF: no substantial changes are to be expected from the UML 2.0 FTF related to U2TP but just renamings. Hence, the alignment is proposed to be done in an U2TP RTF.

Disposition: **Deferred**

Disposition: Transferred

OMG Issue No: <none>

Title:

Source:

Summary:

Discussion:

Disposition: **Transferred to {name of RTF/FTF}**

Disposition: Closed, no change

OMG Issue No: 6294

Title: Commonalities between test suite and test case (Standalone model)

Source:

IBM, Serge Lucio, slucio@us.ibm.com

Summary:

Test suites and test cases share certain characteristics, i.e. a behavior, test objective, and trace. Hence, a common superclass should be introduced.

Discussion:

The decision is made not to introduce a common superclass to avoid confusion between Test Case and Test Suites.

Disposition: **Closed, no change**

OMG Issue No: 6953

Title: Test Case execution in a Suite or Test Case Context

Source:

IBM, Serge Lucio, slucio@us.ibm.com

Summary:

In some contexts where a Test Case (B) is reused (i.e. invoked) from another Test Case (A), there is an ambiguity to the actual intent of the tester

> "A" needs to assess that "B" passes to set its own verdict
>
> "A" is reusing the behavior of "B". The verdict set for "B" by the arbiter is not relevant for either "A" or "B"

Proposal: A Test Case has a mandatory argument, which decides if its verdict should be logged in the Test Log. If the boolean is true, the verdict is logged, otherwise it is not

Discussion:

This issue is tool vendor specific and should not be resolved on test specification level.

Disposition: **Closed, no change**

OMG Issue No: 7194

Title: Hybrid Defaults

Source:

University of Göttingen, Jens Grabowski, grabowski@informatik.uni-goettingen.de

Summary:

Hybrid defaults (i.e. defaults in state machines for test cases defined in interactions) cause the problem of defining the semantics for a hybrid behavior definition. In particular, the interworking of diagrams of different types is in general is not well defined in UML 2.0.
Hence, hybrid defaults should be removed from the U2TP specification

Discussion:

The integrated use of different behavioral diagrams is one objectives of UML 2.0. Even if this has not yet been completely defined, the testing profile should not exclude it but rather await the solutions coming up within UML 2.0.

Disposition: **Closed, no change**

Disposition: Duplicate/merged

OMG Issue No: <none>

Title:

Source:

Summary:

Discussion:

Disposition: **Closed, no change**