Prototype release guide

IBM Engineering Systems Design Rhapsody - TestConductor Add On Version 2.8.4

Embedded Systems AG

BTC Embedded Systems AG
Gerhard-Stalling-Straße 19
26135 Oldenburg
+49 441 969738 0 (voice)
+49 441 969738 64 (fax)

License Agreement

Trademarks

Table of Contents

1 Introduction

This release of Rhapsody ATG and TestConductor provides updates of the ATG and TestConductor Add Ons for the Rhapsody 9.0.1 release. Included are some enhancements, bug fixes and a prototype of a new feature (support for verification of invariants). This document gives a brief overview about the new capabilities of this release. Additional information can be found in the release notes and in the user guide for ATG or TestConductor located in the folders Rhapsody\Doc\pdf_docs and Rhapsody\Doc\html_docs.

2 Integration with Rhapsody 9.0.1

2.1 Installation

This update release of ATG and TestConductor is for the Rhapsody 9.0.1 release, Rhapsody 9.0.1 ifix1, Rhapsody 9.0.1 ifix2 or Rhapsody 9.0.1 ifix3 version. Rhapsody must be installed before installing the update installer(s). The Rhapsody client must not be running while installing the updates of ATG or TestConductor otherwise the installation might fail or be incomplete. The update installers contain the complete Add Ons, including documentation and samples.

When using Rhapsody ATG and TestConductor then both Add Ons must be updated to this release otherwise they will not work correctly. The installers of this update must be uninstalled separately, they are not uninstalled automatically when uninstalling Rhapsody.

When contacting support regarding issues or questions please make sure to include the build number of ATG and/or TestConductor. The build number of ATG can be found in the details of the file Rhapsody\Atg\install\RhapATG.dll or in the text file Rhapsody\Atg\ATGBuildNo.txt. The build number of TestConductor can be found in the details of the file Rhapsody\TestConductor\TestConductor.dll or in the text file Rhapsody\TestConductor\RTCBuildNo.txt.

To switch back to the original release versions of ATG and TestConductor after installing the updates both the updates and Rhapsody must be uninstalled. Then Rhapsody can be installed again including the integrated release version of the Add Ons.

If after installing the updates a previously released ifix for Rhapsody 9.0.1 is installed then the ifix will overwrite ATG or TestConductor files with older versions. In this case the update installers of ATG and TestConductor should be uninstalled before installing the Rhapsody ifix and then the update installers should be installed again.

2.2 SysML profile

The Rhapsody SysML profile modifies the menu structure of the context menu in the Rhapsody browser. The SysML profile from the Rhapsody 9.0.1 installation does not contain menu entries for some elements newly added to the Testing Profile of this prototype release, for example for test sets or for invariant check functions.

An updated file for the SysML profile is part of this prototype release. To make use of it either copy the file SysML.sbs from folder Share\Profiles\InvariantCheckProfile to folder Share\Profiles\SysML\

SysMLProfile_rpy (replacing the original one) or compare the two versions of the file in a diff tool and apply the needed changes to the version located in Share\Profiles\SysML\SysMLProfile_rpy.
The invariant check profile provides two versions of the SysML profile file: SysML.sbs is for Rhapsody 9.0.1 and Rhapsody 9.0.1 ifix1 while SysML_9.0.1.ifix2.sbs is for Rhapsody 9.0.1 ifix2 or ifix3 (please rename it to the original name SysML.sbs before copying it into the Rhapsody installation; when using Rhapsody 9.0.1 ifix3 please also save the file in the new .sbsx format).

2.3 Rhapsody for Java

To be able to start tested Java applications in assertion based testing mode TestConductor needs to pass some command line options to the application. This is not supported by Rhapsody in the 9.0.1 release or in the Rhapsody 9.0.1 ifix1 version. To allow passing command line options to a Java application the file Share/etc/jdkrun.bat needs to be modified. The invariant check profile contains a file Share\Profiles\InvariantCheckProfile\ChangesFor_jdkrun.bat with the necessary changes.
When using Rhapsody 9.0.1 ifix2 or ifix3 then this change is not necessary (Rhapsody ifix2 and ifix3 already includes this change).

2.4 TestConductor adapter for Engineering Test Management, used version of log4j

The TestConductor adapter for Engineering Tests Management is using log4j for writing information into a log file. The version of log4j used in the Rhapsody 9.0.1 release has a security vulnerability which was fixed in version 2.17 of log4j. With this update release the TestConductor adapter for Engineering Tests Management installs and uses version 2.17.1 instead of the older, insecure version, however the installer does not remove the files of the old version of log4j from the Rhapsody TestConductor installation. The now unused files can be deleted from the folder TestConductor\ETM_TestConductorAdapter\lib without affecting functionality of the adapter: log4j-slf4j-impl-2.11.2.jar, log4j-jcl-2.11.2.jar, log4j-core-2.11.2.jar, log4j-1.2.14.jar, log4j-api-2.11.2.jar.
Alternatively you can delete the folder TestConductor\ETM_TestConductorAdapter\lib before installing the update installer.
When using Rhapsody 9.0.1 ifix3 this is not necessary, the installation contains only file of log4j version 2.17.1.

3 Verification of invariants enhancement prototype

This release provides a prototype of a new functionality of TestConductor: Support for verification of invariants by using so called invariant check functions which are invoked during the execution of test cases with a high frequency. This allows verification of conditions which should hold all the time during a test execution in contrast to standard checks which are performed only at a specific location of a test sequence.
Verification of invariants is supported for C++, C and Java when using the assertion based testing mode. Verification of invariants can be enabled by checking the tag “EnableInvariantVerification” on the code generation configuration used for testing.

3.1 Defining and using invariant check functions

Invariant check functions can be created in a test context by right clicking the test context and then selecting Add New → Testing Profile → Invariant Check Function. The check can then be implemented in the body of the function using the language specific TestConductor assert macro or function:

- RTC_ASSERT_NAME for C++ and C
- TestConductor.ASSERT_NAME for Java

These asserts allow defining a string message and an expression. The string message is shown in the test execution window and in the html test result once for each execution of the assert and allows identifying the assert and can be used to visualize data from the assert. The expression is evaluated for the result of the assert.

After defining an invariant check function it can be applied by one or more test cases by adding an <<AppliedInvariantCheck>> dependency from the invariant check function to the test case. Alternatively the dependency can be drawn to the test context, meaning the invariant check function is applied by all test cases of the test context.

The Testing Profiles offers also invariant check matrices or invariant check tables as a convenient way to manage which invariant check functions are applied by which test cases. Right click on the owning test package of the test context and select Add New → Testing Profile → Invariant Check Matrix/Table, then adjust the scope of the created element and open it. The matrix or table shows all invariant check functions and their <<AppliedInvariantCheck>> dependencies and allow adding or removing <<AppliedInvariantCheck>> dependencies from invariant check functions to individual test cases or to the test context.

When executing test cases the results of the executed assert(s) from the applied invariant check functions is shown in the TestConductor test execution UI and in the html test result. If such an assert fails then the test case is failed and the test case ends. A witness scenario shows the location of the failed assert by coloring the preceding message in red, this indicates when the failure happened but it does not necessarily imply the red message has directly caused the failure: A failed check of an an invariant could also be caused by internal activity in the SUT not related to the message.

The verification of invariants is based on an instrumentation of the driver and stub operations and of the code of the SUT, ensuring a high frequency of invocations of the applied invariant check functions both in the test harness and in the SUT. The instrumentation is performed after the code has been generated by Rhapsody. When testing code which has been generated outside of Rhapsody then only the driver and stub operations are instrumented with calls of the applied invariant check functions.

3.2 Helpers provided by the invariant check profile

For easier definition the body of invariant check functions this release of TestConductor contains a helper profile with a Java plugin providing two context menu helpers which are available after adding the profile to a model. To add the profile to a model select the Rhapsody menu File → Add Profile to Model…, in the opened dialog navigate to the folder Share/Profiles/InvariantCheckProfile/InvariantCheckProfile_rpy and select the file InvariantCheckProfile.sbsx.

The first helper of this profile provides the context menu “Copy Attribute Check” for Attributes and Value Properties and the context menu “Copy In-State Check” for states of statecharts, actions of activities and similar elements. When invoking Copy Attribute Check the helper generates an assert to check the attribute value against a value and puts it into the copy/paste buffer of the system. After this the string can be pasted into the body of an invariant check function and modified, like editing the information string of the assert or adjusting the expected value of the attribute.
When invoking “Copy In-State Check” the helper generates an assert to check if the statechart (or activity) currently is in the state (or action etc.) and puts it into the copy/paste buffer of the system. After this the string can be pasted into the body of an invariant check function and modified.
The second helper of this profile offers a UI to define an invariant check function. After invoking “Invariant Check UI” on an invariant check function the UI opens and allows to define the body of the function in two separate edit fields. Below “Assertion Name” the identifying string for test excution UI and test result can be entered. Below “Assertion Check Expression” the check expression of the assert can be entered; when clicking the Get State/Attribute button then the helper generates a check condition for the selected element and adds it to the Assertion Check Expression at the end of the current content. Sub-expressions can be combined using boolean operators like &&, ||, ! or ( ). The UI does not perform a syntax check of the expression.
The title of the UI shows the name of the currently edited invariant check function and the navigation expression to it. When clicking on Ok the helper generates the complete body for the invariant check function including the assert macro/function and writes it to the body of the invariant check function, overwriting any existing content, and closes the UI. When clicking on Cancel then the UI closes without changing the invariant check function. When clicking on the Preview Body button the helper generates the complete body for the invariant check function including the assert macro/function and writes it to the Generated Body of Invariant Check Function field.
Both helpers compute the navigation expression from the test context to attributes or states according to the test architecture with the currently active code generation configuration.

4 Other new features and enhancements

4.1 Test Sets

With this release TestConductor supports grouping multiple test cases of the same test context together into so called test sets. For details please refer to the TestConductor user guide document.

4.2 Assertion based testing for Java

With this release TestConductor supports the assertion based testing mode also for Rhapsody in Java. New test architectures for Java will be tailored for assertion based testing per default. For details please refer to the TestConductor user guide document.

4.3 Browser views and property perspectives

With this release the Testing Profile provides views for the Rhapsody browser allowing to focus the browser on certain kinds of test artifacts. These views can be selected after clicking on “Entire Model View” at the top of the Rhapsody browser.

With this release the Testing Profiles provides perspectives for properties shown in the Rhapsody feature dialog allowing to focus on properties to customize TestConductor options. These property perspectives can be selected in the filter options of the properties page of the feature dialog of elements like test packages or test configurations.

4.4 ATG support for Cygwin 64 bit GNU g++

With this release ATG supports also 64 bit GNU g++ when using Cygwin compile environments.

4.5 Enhancements from the ifix2 release

ATG supports Cygwin compile environments with GNU g++ compilers up to version 10.

TestConductor supports computation of code coverage for GNU gcc/g++ compilers up to version 10.

4.6 Enhancements from the ifix1 release

ATG supports Visual Studio version 2017 compile environments.

Support for computation of model coverage when testing via a target proxy.