

**Rational.** Rhapsody

**IBM.**

## **IBM® Rational® Rhapsody® Automatic Test Generation Add On**



**User Guide**



***Rhapsody<sup>®</sup>***

**IBM<sup>®</sup> Rational<sup>®</sup> Rhapsody<sup>®</sup>  
Automatic Test Generation  
Add On User Guide**

**Release 3.6.4**



## **License Agreement**

No part of this publication may be reproduced, transmitted, stored in a retrieval system, nor translated into any human or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of the copyright owner, BTC Embedded Systems AG.

The information in this publication is subject to change without notice, and BTC Embedded Systems AG assumes no responsibility for any errors which may appear herein. No warranties, either expressed or implied, are made regarding Rhapsody software and its fitness for any particular purpose.

## **Trademarks**

IBM<sup>®</sup> Rational<sup>®</sup> Rhapsody<sup>®</sup>, IBM<sup>®</sup> Rational<sup>®</sup> Rhapsody<sup>®</sup> Automatic Test Generation Add On, and IBM<sup>®</sup> Rational<sup>®</sup> Rhapsody<sup>®</sup> TestConductor Add On are registered trademarks of IBM Corporation.

All other product or company names mentioned herein may be trademarks or registered trademarks of their respective owners.

© Copyright 2000-2011 BTC Embedded Systems AG. All rights reserved.

# Contents

<b>Contents</b>	<b>5</b>
<b>Document Structure</b>	<b>7</b>
Contacting IBM® Rational® Software Support	8
Conventions	9
Where to Find More Information	9
<b>Introduction</b>	<b>10</b>
ATG is Based on UML	11
Structural Testing and the Role of Models	12
ATG is for ...	13
What are the Approaches behind ATG?	13
Sample Model: TheVendingMachine	13
<b>Rhapsody</b>	
<b>UML Testing Profile</b>	<b>16</b>
Automatic Test Generation (ATG) Package	17
<b>Test Case Generation for Unit Testing</b>	<b>19</b>
Automatically Creating a Test Architecture	19
Generate and Build the Test Context	23
Apply ATG	24
Test Generation Configuration	25
Generate Test Cases	28
Export Test Cases to Rhapsody TestConductor	30
Execute an ATG Test Case	34
<b>Test Case Generation for Integration Testing</b>	<b>38</b>
Manually Creating a Testing Component	38
Generate and Build the Test Component	40
Apply ATG	41
Test Generation Configuration	43
Generate, Export, and Execute Test Cases	50
<b>ATG Management</b>	<b>61</b>
The Test Generation Component	61
Create a Test Generation Component	61
Delete a Test Generation Component	62
Clear All	63

Clear Test Cases .....	63
Test Definition Options .....	63
The Test Generation Configuration.....	65
Delete a Test Generation Configuration .....	65
The General Definition Tab.....	66
The Interface Definition Tab .....	68
The Coverage Definition Tab .....	70
Clear All.....	70
Clear Test Cases .....	71
Test Definition Options .....	71
Sync ATG Data with Application .....	73
Rhapsody ATG settings .....	74
Test Case Generation .....	76
The Rhapsody in C++ Automatic Test Generation Dialog.....	76
View Customization.....	78
One-click Expand/Collapse Hierarchical View .....	78
Change between Hierarchical and Flat View.....	79
Show Combinations of Test Cases .....	81
Exporting Test Cases.....	83
Export Formats .....	83
XML .....	83
Test Scenarios.....	84
TestConductor .....	84
Exporting a Single Test Case.....	84
Exporting Test Cases on Configuration Level.....	86
Exporting Test Cases on Test Component Level.....	87
Report Generation .....	88
Test Generation Configuration Report .....	88
Testing Component Report.....	91
Test Execution.....	91
<b>Advanced Features .....</b>	<b>92</b>
Specifying Interfaces in the Model .....	92
Provided Interfaces and Required Interfaces .....	92
Operations and Events – Argument Constraints .....	93
User-Defined Constraints on Types.....	95
Working with Libraries .....	97
<b>Coverage Measurement with Third-Party Tools .....</b>	<b>100</b>
<b>Appendix .....</b>	<b>101</b>
Restrictions .....	101
Frequently Asked Questions .....	101

# Document Structure

---

This user guide is organized as follows:

- ◆ **Chapter 1, Introduction**, provides an introduction to Rhapsody ATG through a high-level overview of the main features.
- ◆ **Chapter 2, Rhapsody UML Testing Profile**, provides an overview about the concepts of the Testing Profile as implemented in Rhapsody.
- ◆ **Chapter 3, Test Case Generation for Unit Testing**, provides an overview on creating Test Architectures, defining Test Cases, and executing Test Cases for unit testing.
- ◆ **Chapter 4, Test Case Generation for Integration Testing**, provides an overview on creating Test Architectures, defining Test Cases, and executing Test Cases for integration testing.
- ◆ **Chapter 5, ATG Management**, provides information on how to administrate ATG, generate test cases, generating text and html reports for the generated test cases, exporting test cases into desired formats.
- ◆ **Chapter 6, Advanced Features**, provides information on defining constraints.
- ◆ **Chapter 7, Coverage Measurement with Third-Party Tools**, provides course information on doing external coverage measurements.
- ◆ **Chapter 8, Appendix**, provides information on further documents concerning restrictions and frequently asked questions.

## Contacting IBM® Rational® Software Support

IBM Rational Software Support provides you with technical assistance. The IBM Rational Software Support Home page for Rational products can be found at <http://www.ibm.com/software/rational/support/>.

For contact information and guidelines or reference materials that you need for support, read the [IBM Software Support Handbook](#).

For Rational software product news, events, and other information, visit the [IBM Rational Software Web site](#).

Voice support is available to all current contract holders by dialing a telephone number in your country (where available). For specific country phone numbers, go to <http://www.ibm.com/planetwide>.

Before you contact IBM Rational Software Support, gather the background information that you will need to describe your problem. When describing a problem to an IBM software support specialist, be as specific as possible and include all relevant background information so that the specialist can help you solve the problem efficiently. To save time, know the answers to these questions:

What software versions were you running when the problem occurred?

Do you have logs, traces, or messages that are related to the problem?

Can you reproduce the problem? If so, what steps do you take to reproduce it?

Is there a workaround for the problem? If so, be prepared to describe the workaround.

## Conventions

The following table lists the conventions used in the Rhapsody documentation.

Style	Description
command1 > command2	The greater-than (>) symbol leads you through the steps in a menu or key sequence. For example, <b>Add New &gt; Package</b> means that you should first select <b>Add New</b> , then select <b>Package</b> from the <b>Add New</b> submenu.
<b>Bold type</b>	Bold type indicates items that you should select, such as buttons or checkboxes in dialog boxes. For example: Click <b>Apply</b>
<i>Italic type</i>	<i>Italic type</i> is used for emphasis, titles of referenced documents and new terms.
Courier type	Courier type is used for file names and directory paths, user input, and code-related items such as instance names and properties.
<filename>	Angle brackets surround variable names that you should replace with actual names. For example, you should replace <filename> with the actual name of a file.

## Where to Find More Information

The following documents provide additional information on Rhapsody ATG and the development process concepts used in this user guide:

- ◆ W. Damm, M. Cohen, Advanced validation techniques meet complexity challenge in embedded software development, *Embedded System Journal*, 2001.
- ◆ B.P. Douglass, ROPES: Rapid Object-oriented Process for Embedded Systems, I-Logix Inc., 1999 (adaptation of the material from book: *Doing Hard Time: Developing Real-Time Systems using UML, Objects, Frameworks, and Patterns* Reading, MA: Addison-Wesley, 1

# Introduction

---

Welcome to the user guide for IBM® Rational® Rhapsody® Automatic Test Generation Add On (Rhapsody ATG). Rhapsody ATG is a test case generation tool using standard Unified Modeling Language™ (UML™) design notations. Using ATG, you can automatically generate test suites and perform test execution for your applications developed with the Rhapsody in C++ design tool at any stage in your development cycle.

The typical UML development process (such as the Rapid Object-Oriented Process for Embedded Systems, ROPES) is iterative, starting with an early, fairly abstract version and progressing to more and more concrete prototypes. To test a System Under Test, use ATG in your development process to do unit testing, integration testing, or regression testing.

Rhapsody ATG complements Rhapsody® TestConductor. TestConductor automatically generates Test Architectures for the system under test (SUT), and creates test monitors and test drivers from Rhapsody sequence diagrams (SDs). During automated test execution, the generated monitors determine whether the executed model satisfies the selected sequence diagrams. ATG generates test cases that can be exported to Rhapsody in order to execute test cases with TestConductor.

Rhapsody ATG is a tool that enables *Design for Testability (DFT)*. DFT is a process capability that enables repeatable and cost-effective testing throughout the development process. Thus, DFT is an approach including:

- ◆ Model-Driven Development (MDD) with visual modeling and animation in Rhapsody
- ◆ Structural testing with ATG
- ◆ Requirements-based testing with TestConductor

If users apply DFT, they:

- ◆ Get repeatable tests based on models and software.
- ◆ Expand the level of achieved design coverage.
- ◆ Are able to perform regression testing.
- ◆ Speed up tests on subsequent versions of the design.

ATG offers test case generation to perform structural testing. It extends Model-Driven Development to include both Model-Driven *Code* Generation and Model-Driven *Test* Generation. ATG automatically generates test cases with high coverage of the design, including:

- ◆ Model element coverage—States, transitions, operations, event generation
- ◆ Model code coverage—All relevant combinations of inputs for full model code coverage sufficient to fulfill criteria such as Modified Condition Decision Coverage (MC/DC)

Test cases consist of input stimuli, and output reactions as computed from a given model. The generated test cases can be used to test the production code. Test case execution will lead to a high degree of coverage of the production code, and the execution will give pass/fail results. A fail result indicates that the actual results of a test execution run do not match the expected output reactions as computed from the model.

ATG generated test cases can be used for unit testing, integration testing, regression testing, and target-based testing. It works with third-party tools for code coverage analysis and test execution.

This user guide takes you through some stages of a design and testing process supported by ATG, such as unit testing and integration testing, and shows the various capabilities of ATG.

## ATG is Based on UML

Rhapsody ATG is based on the Unified Modeling Language (UML). Rhapsody supports UML-based development processes throughout all development stages—from system modeling to behavioral modeling down to code. Model execution with Rhapsody provides developers a very efficient way to verify and validate their models and the generated implementation-level code—both on host and target computers. By doing interactive model execution, and in particular combined with the animation capabilities of Rhapsody, developers:

- ◆ Can ensure that the modeled behavior satisfies the given requirements.
- ◆ Can find errors at the very early stages of the development process.

The UML standard provides some answers concerning software testing based on UML artifacts, i.e. the UML Testing Profile. The UML testing profile provides mechanisms to define test infrastructures, test definitions, and test executions, but the challenge of test case generation and execution automation is not tackled by the UML standard.

ATG is a new, model-based test generation product for the automatic generation and application of test cases for testing applications generated from Rhapsody UML models.

Model-based test generation means that the UML model information is used in order to structure the test case generation for actual software testing. ATG computes test cases (sequences of operation calls and expected reactions), such that black-box testing and white-box testing can be applied on the final implementation (the *production code*), both on host and on target computers.

The benefits of this model-based test automation product are as follows:

- ◆ More efficient testing activities
- ◆ Reduced overall testing time and cost
- ◆ Increased test coverage quality
- ◆ Early failure detection and correction
- ◆ More extensive and repeatable testing
- ◆ Improved product quality

## Structural Testing and the Role of Models

Safety standards, such as airborne standard DO-178B, require certain specific software verification activities to be done. Highest emphasis is on requirements-based testing. Normal range tests and robustness tests have to be generated. The tests are applied in order to perform low-level tests, software integration tests, and HW/SW integration tests. When the testing is done, it is necessary to perform a *requirements-based test coverage* analysis to assess which requirements have been tested, and a *structural coverage* analysis to assess the degree of structural code coverage. If structural coverage analysis reveals code that was not exercised, either more test cases have to be generated, or the unnecessary code must be removed. Structural testing is usually applied in the phase of unit testing. In this case, testing is driven by the structure of the code. Hence, it is a white-box testing method, where the internals of the system under test are visible. Structural testing is also known as *coverage testing*.

Models help to describe and understand what the system under test is supposed to do. The complexity of software requires development of models to support design and testing activities. For the testing, it is important to find those few input and system state combinations that will reach, trigger, and propagate bugs out of all those that will not. *Model-level test case generation* means that the model is the reference specification used for actual test case generation. Its key characteristic is that test case generation goals are expressed in terms of high-level model elements such as states and transitions. Generated model-level test cases can be applied on implementation models on real test nodes to check correctness and completeness of the developed products under real-time conditions.

ATG analyzes both UML models and the generated C++ code to automatically generate sets of model-level test cases that cover the full source code. The achieved level of source code coverage, when using the generated test cases for execution, must satisfy standard criteria such as Modified Condition Decision Coverage (MC/DC).

Consider the following source code fragment:

```
if (a || b)
    c = 0;
else
    c = 1;
```

To exhaustively test this piece of code, it is necessary to execute both branches of the if-then-else statement. This can be achieved with four different combinations of possible values for decision (*a || b*). MC/DC justifies that it is sufficient to select three out of four possible combinations in order to perform a thorough testing of this if-then-else statement.

When test cases are executed on an application, third-party tools can be applied to measure the achieved source code coverage while testing a system under test.

## ATG is for ...

ATG enables Rhapsody users to generate test cases for single classes in order to perform unit testing, or for a set of classes in order to perform integration testing. The considered class or set of classes is the *System Under Test (SUT)*.

In order to generate test cases for an SUT, ATG uses both UML model and generated C++ code for actual test case generation. C++ code generation within Rhapsody uses code generation components and configurations. An SUT must be represented as an executable code generation component. This means if test cases should be generated for an SUT (either a single class or a set of classes), the SUT must be compiled and linked into an executable.

**Note:** To be able to apply ATG on an SUT, you must be able to interactively execute and simulate the considered SUT within Rhapsody.

## What are the Approaches behind ATG?

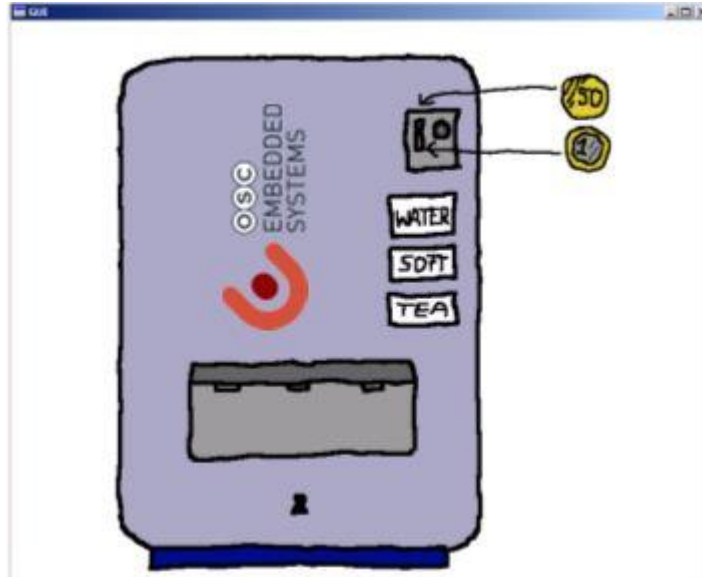
ATG takes the generated C++ source code and performs a *virtual model execution*. Virtual model execution means that a C++ program is not executed on a real physical device with a real processor, but it is executed by a *virtual machine*. In this case, the virtual machine behavior is realized by ATG. Using virtual execution, you can assume a more ideal world for the sake of test case generation. In an ideal world, you can abstract from concrete (real-time) operating system features, actual processor speed, available physical memory, and so on. In a virtual machine, these parameters are configurable.

Virtual model execution is key for ATG in order to explore the reachable state space of the SUT. It enables the application of various test generation strategies on a model under test such that test cases can be generated in an incremental manner. In addition, ATG does not assume that users specify concrete values for inputs, but ATG can handle arbitrary values until a test case has been generated.

To apply the virtual machine to the C++ source code, the source code must be translated into a format understood by ATG. This is very similar to a compiler that translates source code into machine code understood by a normal processor. The ATG C++ translator can handle a large subset of C++ language constructs, but there are some limitations. See the Rhapsody ATG Limitations document for more information.

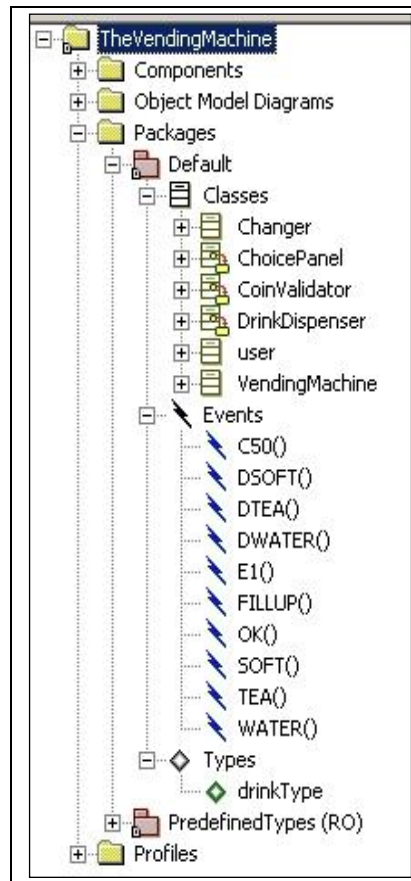
## Sample Model: TheVendingMachine

This user guide uses the Rhapsody in C++ TheVendingMachine sample model. It can be found in <Rhapsody>\Samples\CppSamples\Atg\TheVendingMachineStart>.



**Figure 1: TheVendingMachine**

If a user inserts coins and selects a drink then the vending machine delivers either water, tea, or soft drinks,. The behavior of the vending machine is realized by a set of collaborating objects. The VendingMachine Classes and Events as shown in Figure 2 are Changer, ChoicePanel, CoinValidator, DrinkDispenser, User, and VendingMachine. The Figure also shows the available events used to trigger the objects. Some of the classes are modeled with statecharts, whereas others do not have statecharts. Inserting coins or selecting a drink causes external stimulus. Coin insertion is converted into an operation call to the machine, whereas drink selection is mapped to events that are sent to the machine.



**Figure 2: TheVendingMachine Classes and Events**

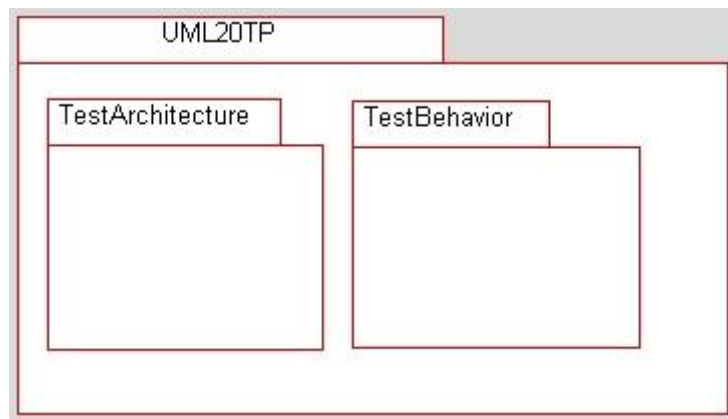
# Rhapsody UML Testing Profile

---

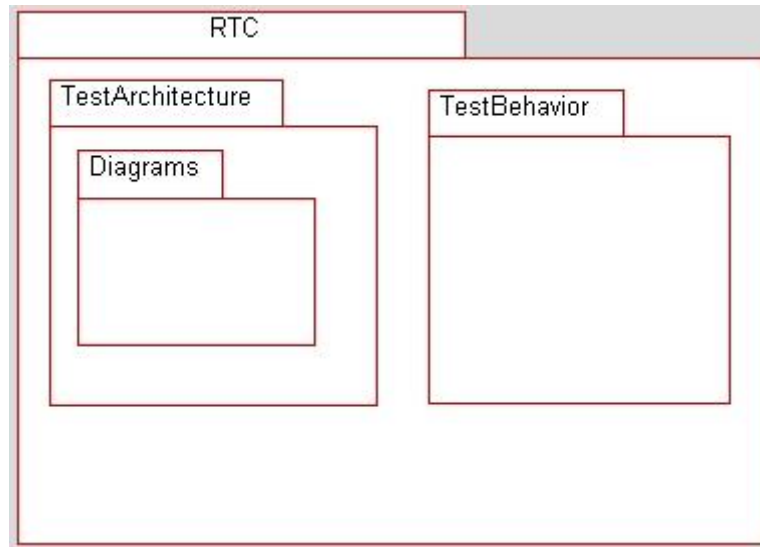
The *Rhapsody UML Testing Profile* is based on the official UML Testing Profile. It contains new terms and stereotypes that can be utilized for model testing artifacts in Rhapsody. A couple of elements defined in the UML Testing Profile are presently not part of the Rhapsody Testing Profile. However, the Rhapsody Testing Profile includes supplementary elements that are not part of the UML Testing Profile. Stubbing, for example, is one of these additional elements that are used for test activities not addressed by the UML<sup>2.0</sup> Testing Profile.

The Rhapsody Testing Profile is prearranged in three major packages with additional sub-packages and the *TestingProfile* stereotype.

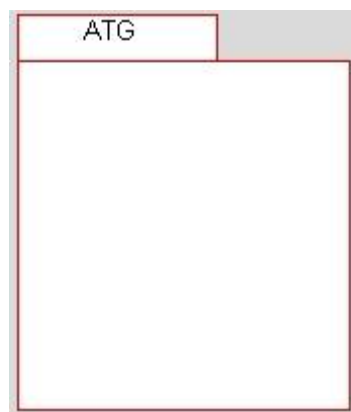
- ◆ Rhapsody UML2.0 Testing Profile (UML20TP)
  1. *TestArchitecture*
  2. *TestBehavior*



- ◆ Rhapsody TestConductor (RTC)
  1. *TestArchitecture*
  2. *TestBehavior*



◆ Automatic Test Generation (ATG)



The Rhapsody Testing Profile is automatically utilized by Rhapsody TestConductor.

In the next section the package *Automatic Test Generation (ATG)* will be described. For further information about the profile Packages *Rhapsody UML2.0 Testing Profile (UML20TP)* and *Rhapsody TestConductor (RTC)* refer to the *Rhapsody TestConductor User Guide*.

## Automatic Test Generation (ATG) Package

The *ATG package* consists of several stereotypes which are enhancements to the UML Testing Profile. Using these stereotypes in the model means that Rhapsody Automatic Test Generation (ATG) is able to interpret defined input/output interface information and constraints to define default settings for a test generation configuration.

Two constrained stereotypes are key: *argRangeConstraint* and *argValueConstraint*. *argRangeConstraint* can be used to define value range constraints. *argValueConstraint* can

be utilized to define single value constraints of enumerations constraints. These constraints can be used on operation or event arguments.

Available are also the interface stereotypes *providedInterface* and *requiredInterface*. Those stereotypes help to remove from the ATG view those classes which are not used as interface classes.

Furthermore, the ATG package contains a number of type constraint stereotypes that can be used to define range or value constraints on type definitions. The following figure provides an overview about the ATG package.



# Test Case Generation for Unit Testing

---

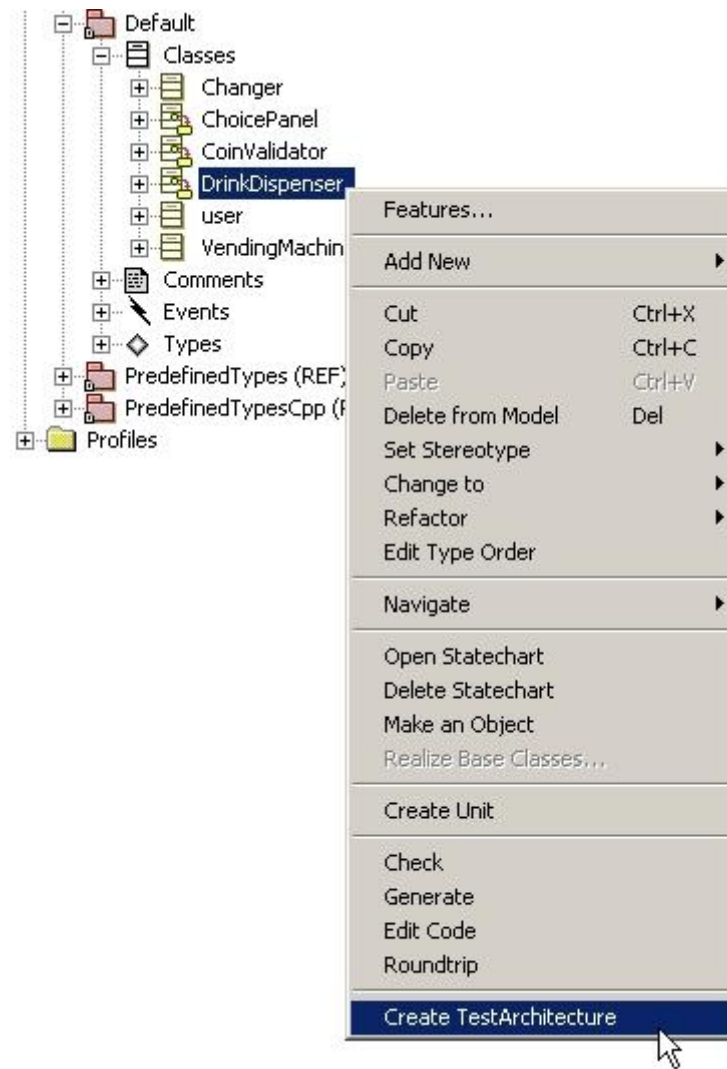
This chapter shows how to process a unit test on an automatically created test architecture with a single SUT class. We will use the Rhapsody Automatic Test Generation (ATG) to generate test cases and execute them with TestConductor.

For the next steps open the VendingMachine Model from the `<\Samples\CppSamples\ATG\TheVendingMachineStart>` folder.

## Automatically Creating a Test Architecture

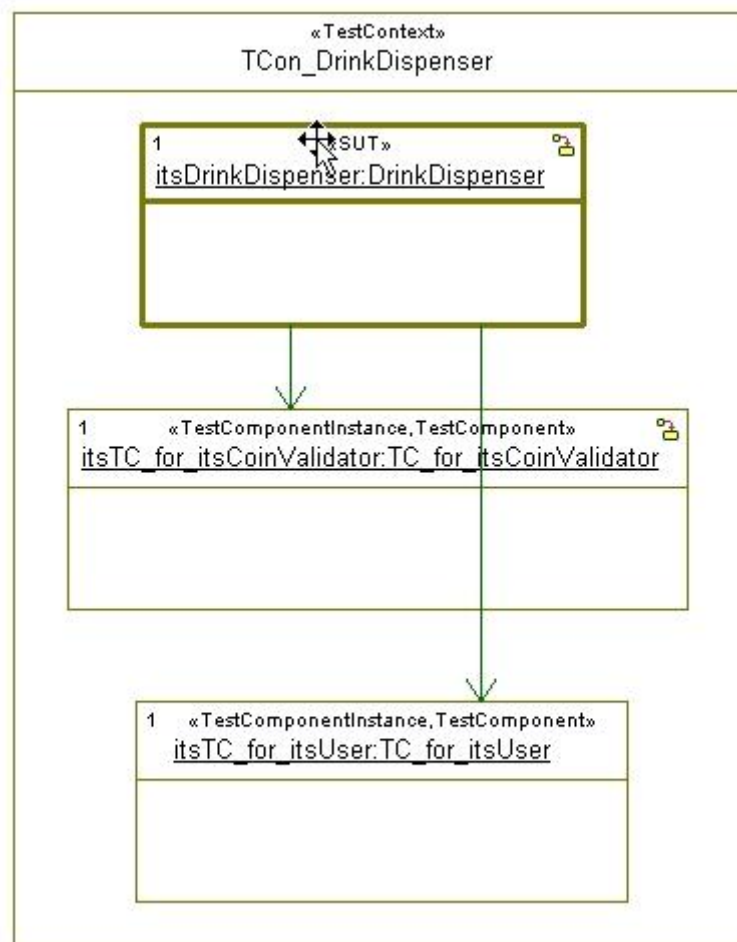
To create a *test architecture* for the class `DrinkDispenser`:

- ◆ Right-click on the `DrinkDispenser` class in the Rhapsody browser and select **Create TestArchitecture**

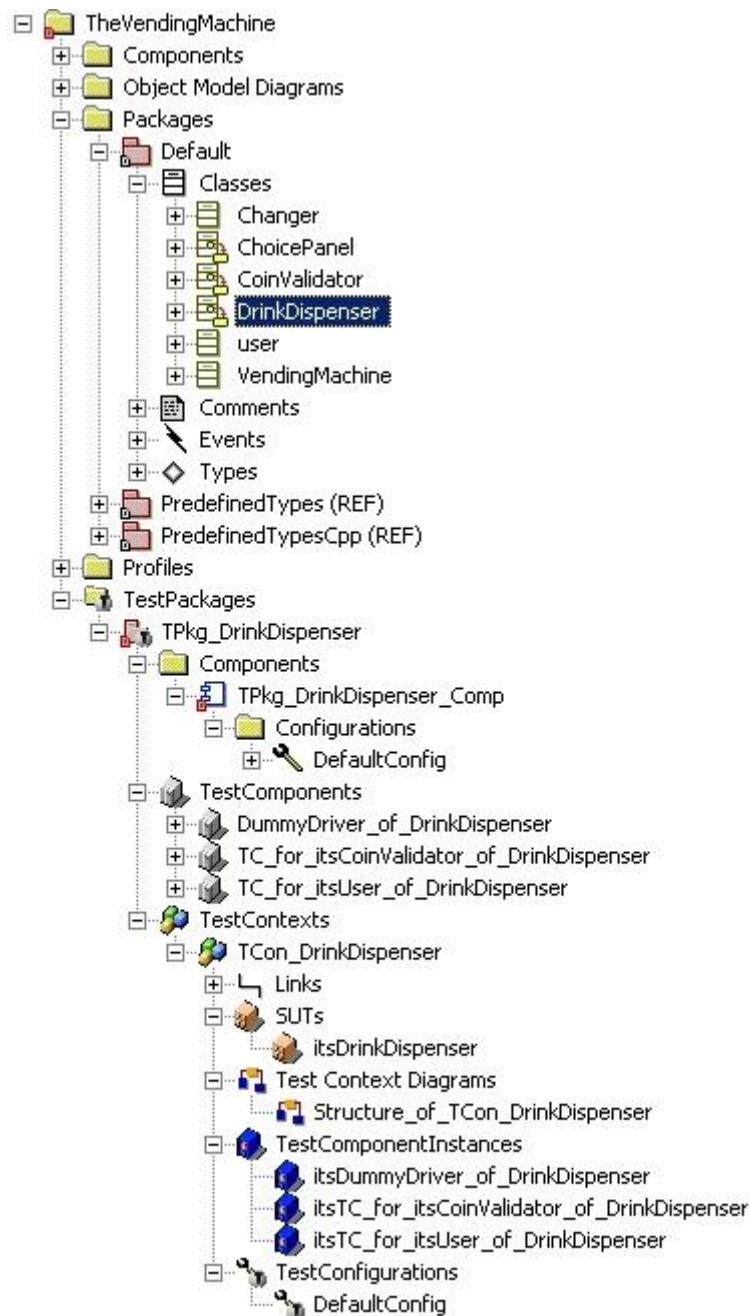


TestConductor automatically creates the complete test architecture which consists of:

- ◆ A new *test context diagram* with the test context `TCon_DrinkDispenser` containing the `DrinkDispenser` object `itsDrinkDispenser` itself as SUT, and all necessary test component instances which are derived from the SUT associations and ports.



- ◆ A new test package TPkg\_DrinkDispenser, which contains all generated test components, the test context TCon\_DrinkDispenser with the SUT itsDrinkDispenser, the test context diagram and the test component instances



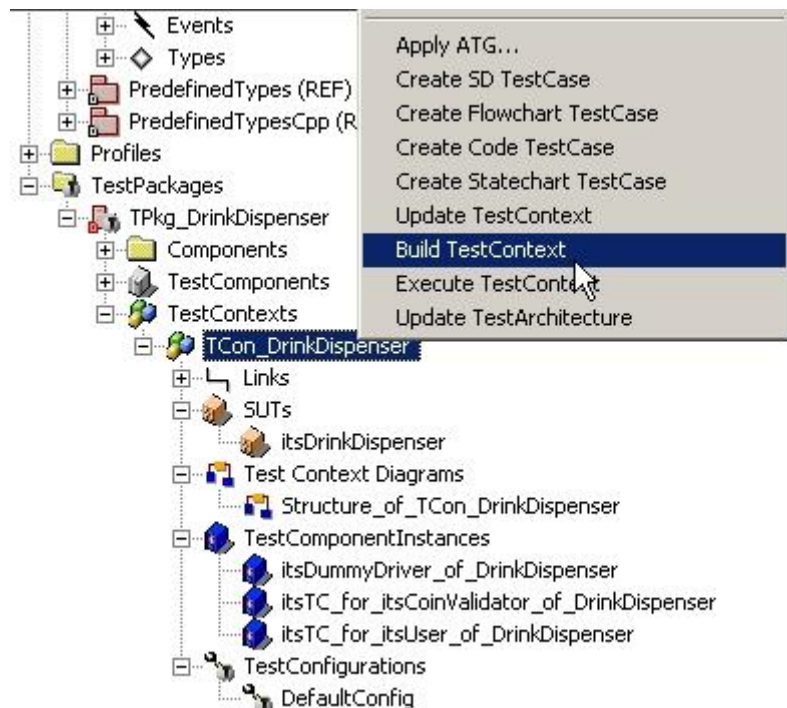
- ◆ A new *test configuration* DefaultConfig (see figure above) with a stereotyped dependency to a new code generation component TCon\_DrinkDispenser\_Component (see figure below).



## Generate and Build the Test Context

After generation of the new *test context* you should check whether it is complete and consistent. Therefore you should generate und build the test context to get information about potential compile or link warnings or errors.

- ◆ Right-click on the test context `TCon_DrinkDispenser` and select **Build TestContext** from the context menu.

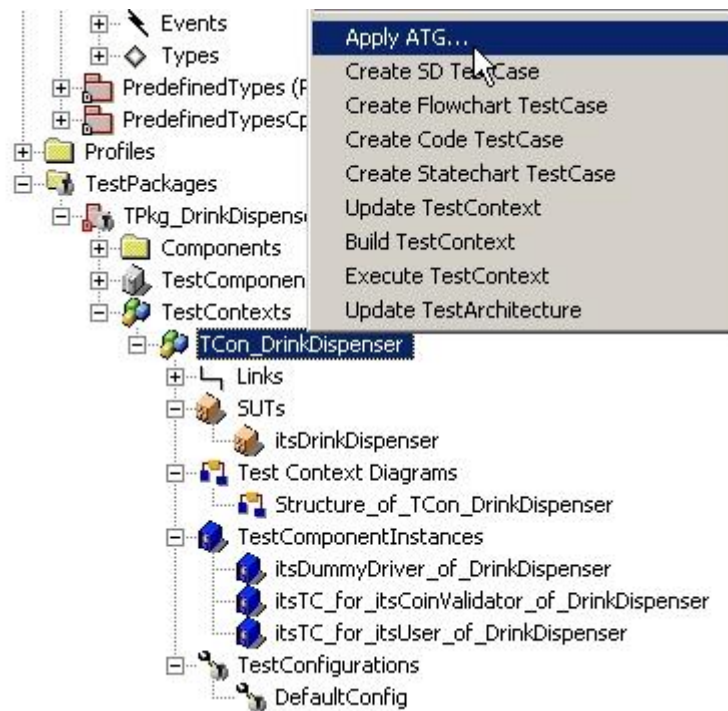


If the generate, compile and link (GMR) procedure produces an executable you are able to execute and test it.

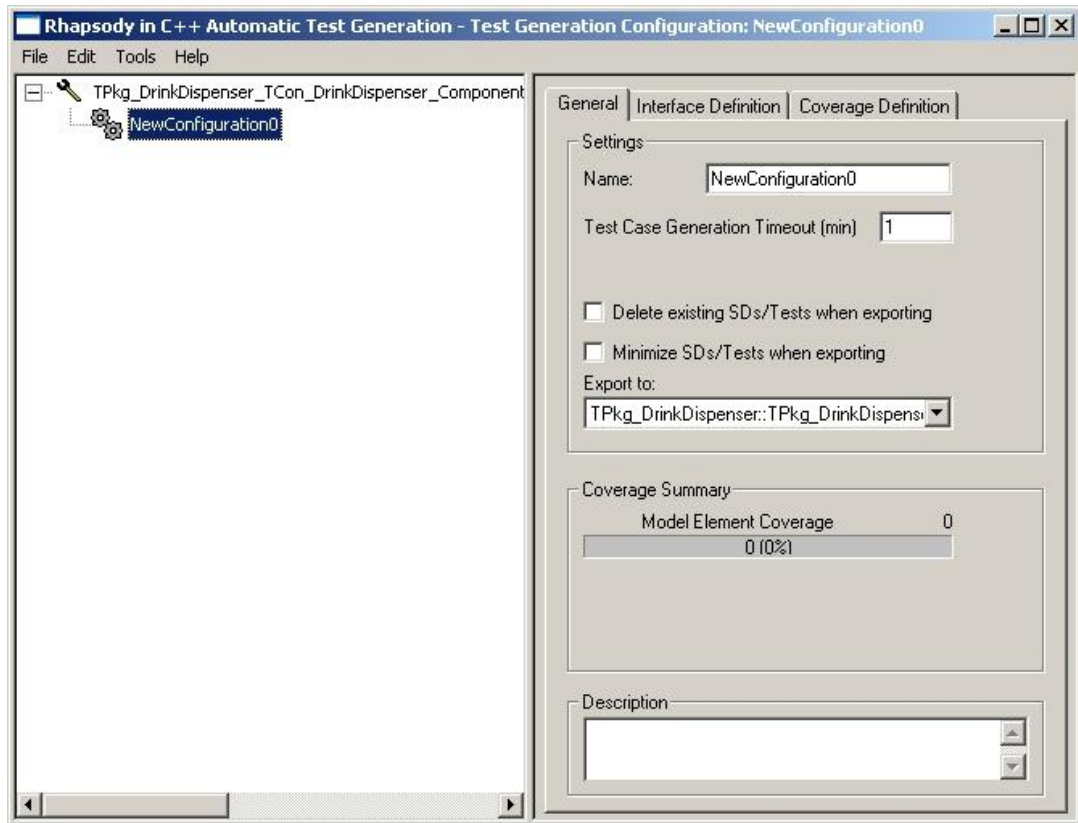
## Apply ATG

After checking whether the new *test context* it is complete and consistent, we should apply ATG to generate test cases.

- ◆ Right-click on the test context `TCon_DrinkDispenser` and select **Apply ATG...** from the context menu.

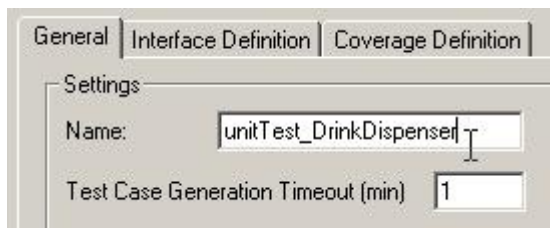


The „Rhapsody Automatic Test Generation“ dialog opens. In the main ATG dialog the user is able to adjust some settings, which mainly affect the test case generation process. A new ATG configuration “NewConfiguration0” was created.



## Test Generation Configuration

The **General** tab of ATG defines the name of the configuration and provides a description box to notice its purpose. The **Timeout** field controls ATG on how much time to spend finding the best coverage. The default is one minute. Increase the value when dealing with complex models. Rename the configuration to “unitTest\_DrinkDispenser” and leave the timeout unchanged.



The next step is the interface definition for this unit test definition. The objective is to perform a unit test for the single class `DrinkDispenser`. Therefore, ATG will generate only test case input sequences for events and operations of this class, which are designated to be inputs. The following Figure specifies the Input interface definition of `unitTest_DrinkDispenser`. Interface settings for the other classes must be explicitly unset using right-button mouse capabilities.

General   Interface Definition   Coverage Definition		
[-] <b>Input Interface</b>		
[+] [Icon]	Changer in Default	
[+] [Icon]	ChoicePanel in Default	
[+] [Icon]	CoinValidator in Default	
[-] [Icon]	DrinkDispenser in Default	TCon_DrinkDispenser.itsDrinkDispenser
[Icon]	DrinkDispenser::Prepare_Soft()	<input type="checkbox"/>
[Icon]	DrinkDispenser::Prepare_Tea()	<input type="checkbox"/>
[Icon]	DrinkDispenser::Prepare_Water()	<input type="checkbox"/>
[Icon]	evDSOFT()	<input checked="" type="checkbox"/>
[Icon]	evDTEA()	<input checked="" type="checkbox"/>
[Icon]	evDWATER()	<input checked="" type="checkbox"/>
[Icon]	evFILLUP()	<input checked="" type="checkbox"/>
[Icon]	DummyDriver_of_DrinkDispenser in ...	
[+] [Icon]	TC_for_itsCoinValidator_of_DrinkDis...	
[+] [Icon]	TC_for_itsUser_of_DrinkDispenser i...	
[Icon]	TCon_DrinkDispenser in Tpkg_Drink...	
[+] [Icon]	user in Default	

**Figure 3: Interface Definition of a Unit Test Configuration**

Events DSOFT, DTEA, DWATER, and FILLUP will be automatically generated by ATG. Operations Prepare\_Soft, Prepare\_Tea, and Prepare\_Water are not selected. These operations are called internally by this class if events are injected, although these operations are marked as public in the Rhapsody model.

Coverage definitions are also focused on this unit under test (DrinkDispenser). You should shrink the range of the test goals to that class, as shown in the following Figure.

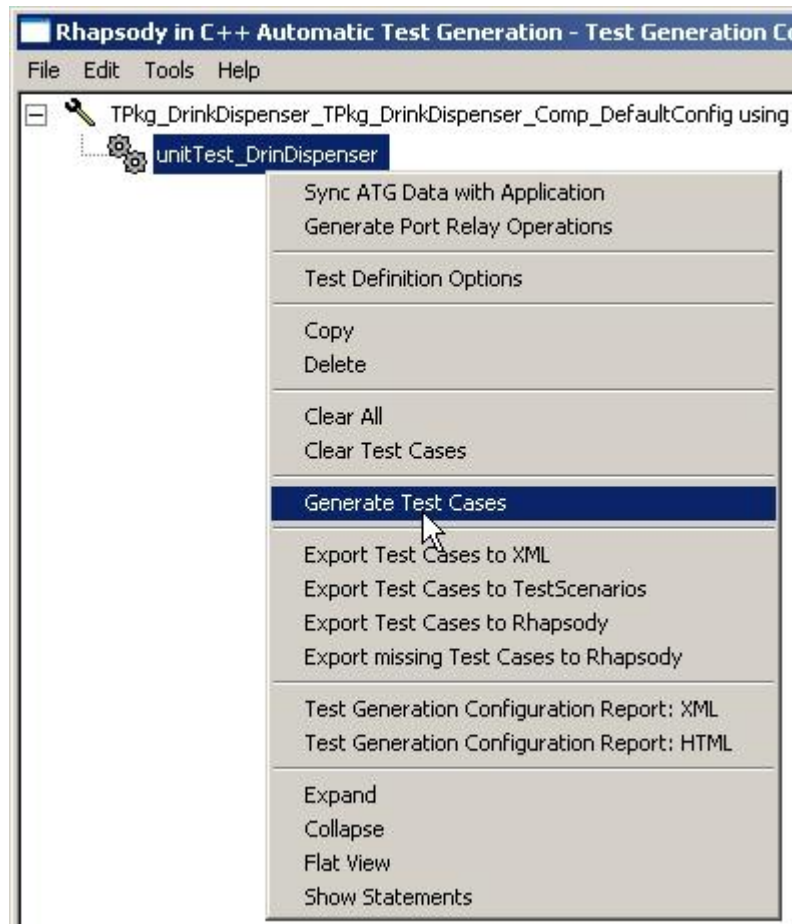
General	Interface Definition	Coverage Definition
<input checked="" type="checkbox"/> Model Element Coverage <input checked="" type="checkbox"/> Model Code Coverage		
<input checked="" type="radio"/> Test Classes <input type="radio"/> Test Specific Instances		
<b>Coverage of Classes</b>		
	Changer in Default	<input type="checkbox"/>
	ChoicePanel in Default	<input type="checkbox"/>
	CoinValidator in Default	<input type="checkbox"/>
	DrinkDispenser in Default	<input checked="" type="checkbox"/>
	DummyDriver_of_DrinkDispenser in ...	<input type="checkbox"/>
	TC_for_itsCoinValidator_of_DrinkDis...	<input type="checkbox"/>
	TC_for_itsUser_of_DrinkDispenser i...	<input type="checkbox"/>
	TCon_DrinkDispenser in TPkg_Drink...	<input type="checkbox"/>
	user in Default	<input type="checkbox"/>
<b>Events</b>		
	Default	<input type="checkbox"/>
	TPkg_DrinkDispenser	<input type="checkbox"/>

**Figure 4: Coverage Definition with Focus on the Class under Test**

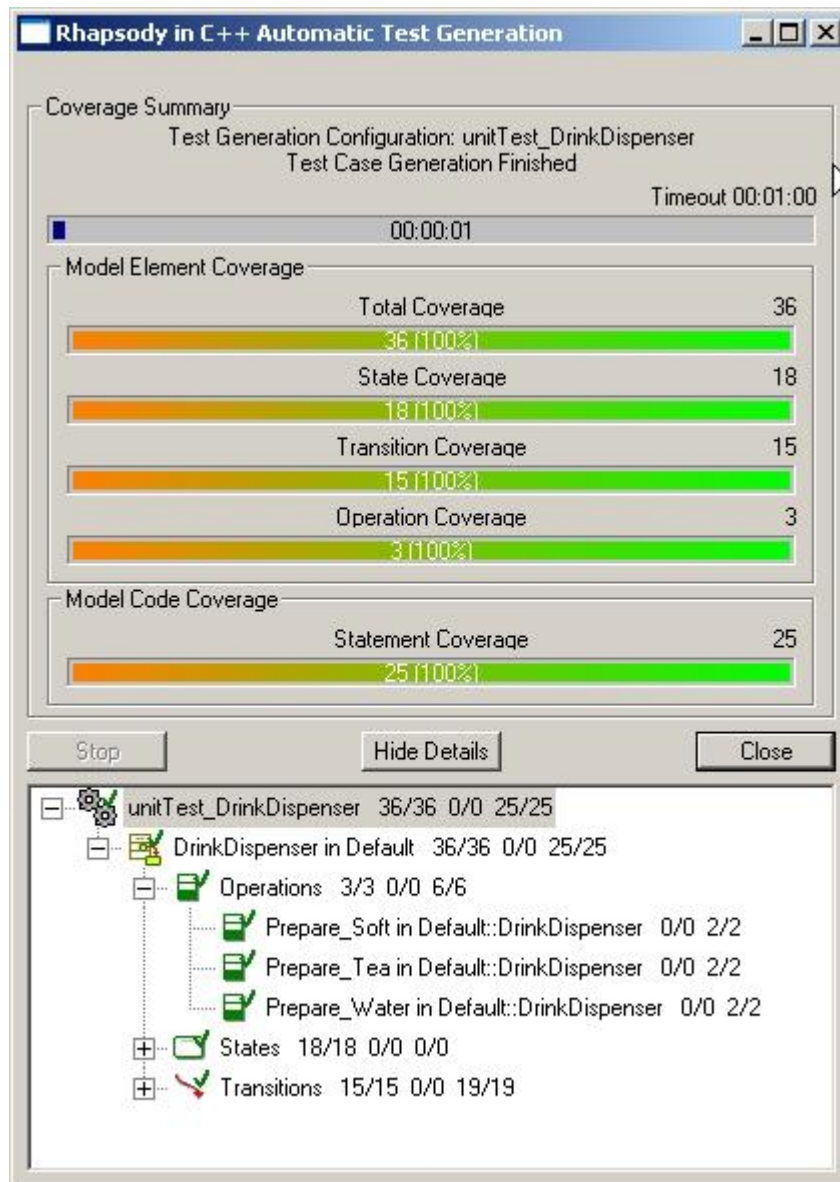
This Test Generation Configuration is now ready for actual ATG test case generation.

## Generate Test Cases

Right-click on the ATG configuration and select `unitTest_DrinkDispenser`.

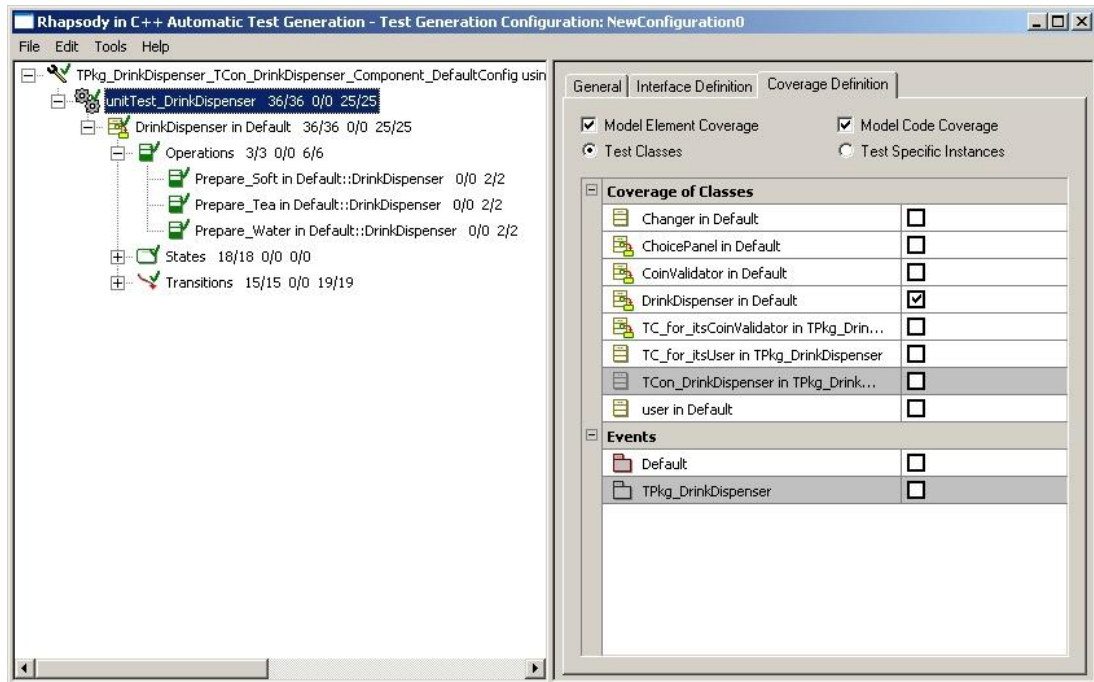


The run-time dialog for actual test case generation will appear.



Obviously it was an easy task for ATG to find full model element coverage for this unit test. ATG reached 100% model element coverage after one second.

Close the dialog by pressing the button **Close**. Control will go back to the ATG main dialog. In the ATG browser you will see the results of the test case generation run.

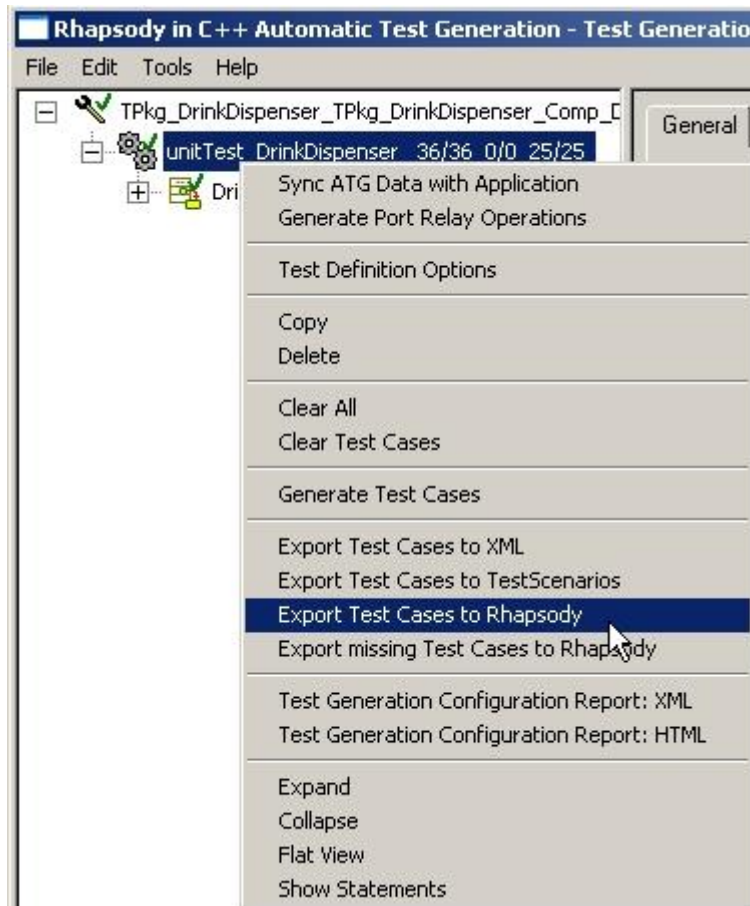


To validate the results ATG gives detailed graphical and textual information about covered and not covered elements. Explore in the ATG browser the detailed information for every element. Covered elements are marked with a green check and not covered elements with a red cross.

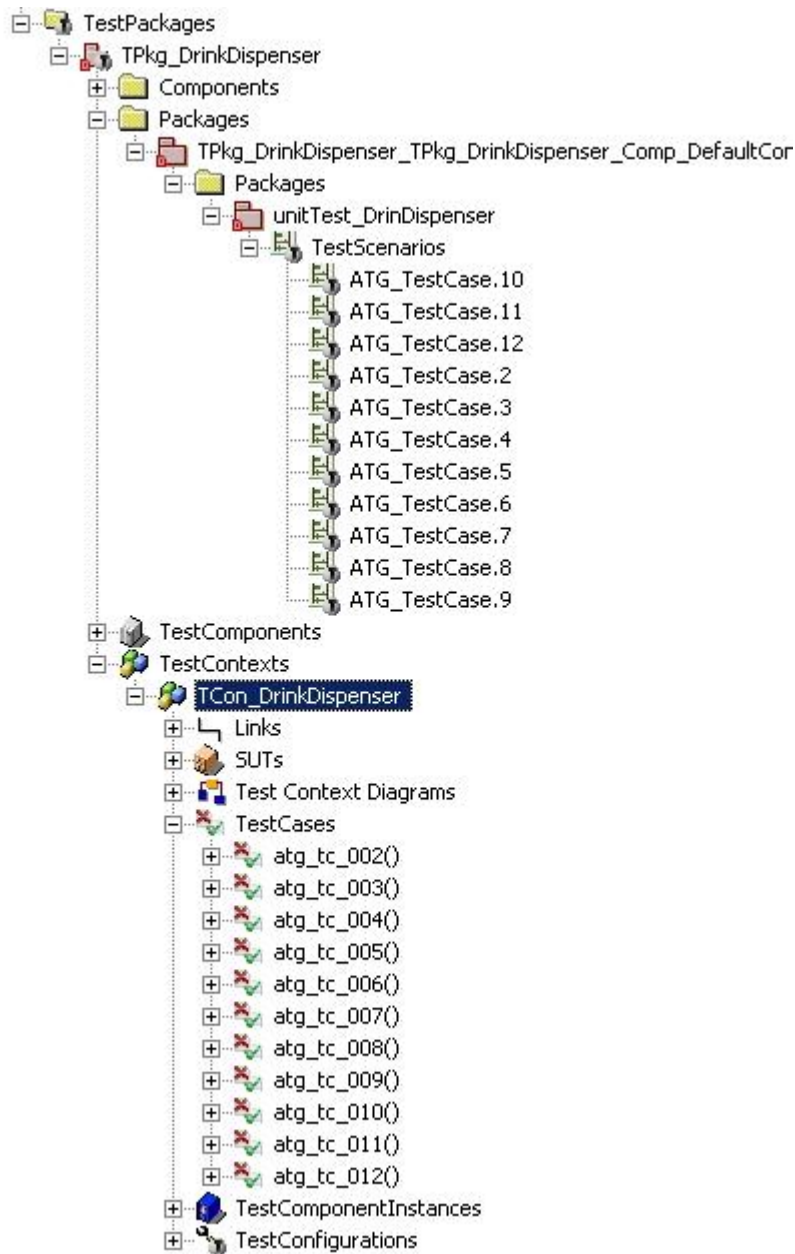
## Export Test Cases to Rhapsody TestConductor

ATG provides the export formats XML, sequence diagrams and directly to the Rhapsody browser. To get further information about exporting to various formats reference the section Exporting Test Cases at page 80. To export the generated test cases to Rhapsody TestConductor

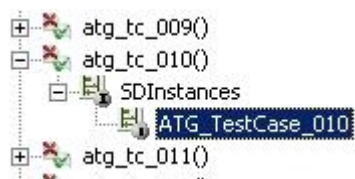
- ◆ Right-click on the configuration item in the ATG browser and select **Export TestCases to Rhapsody**.



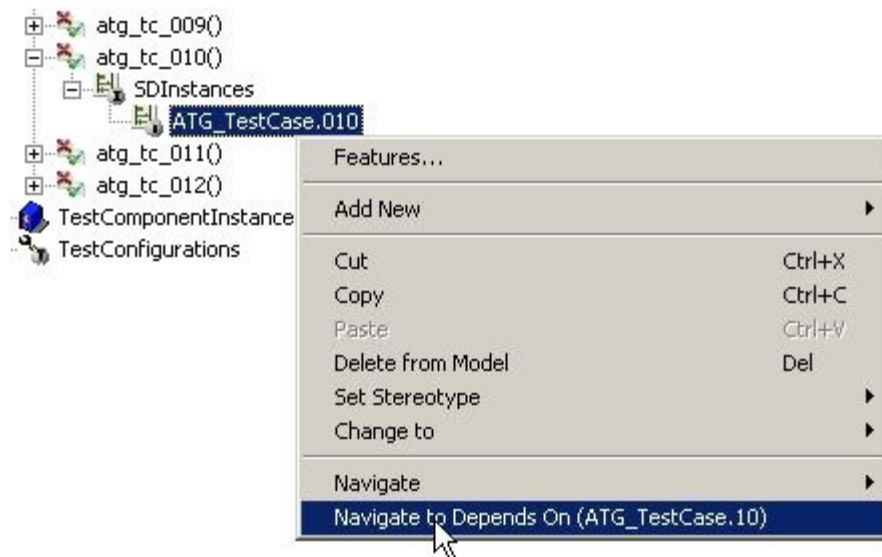
Due to full Rhapsody integration of test cases in the Rhapsody browser ATG inserts the test cases under the folder TestCases in the related test context TCon\_DrinkDispenser. Each test case includes a link to a test scenario, which are stored in the TestScenarios folder.



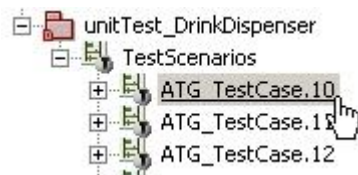
Close the ATG main dialog and focus on the test context `TCon_DrinkDispenser`. Explore the test case `atg_tc_010` by expanding the tree structure and follow the link to the corresponding test scenario.



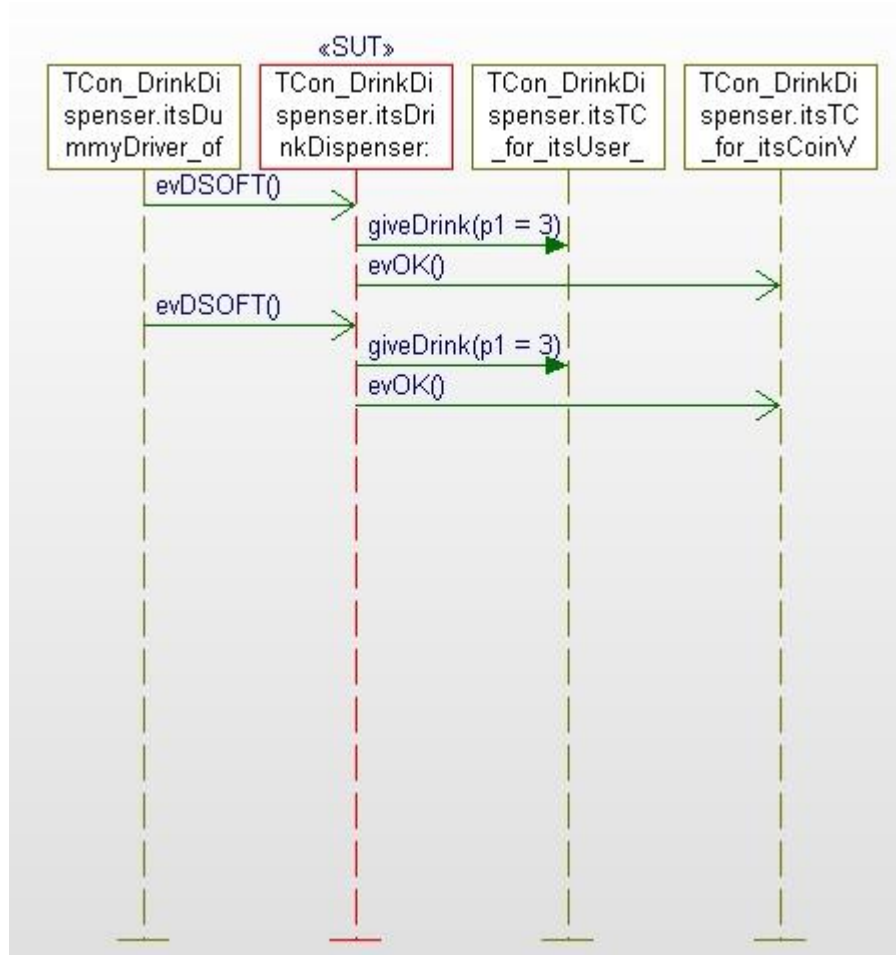
- ◆ Follow the link “ATG\_TestCase\_010” by right-clicking and selecting **Navigate to Depends On (ATG\_TestCase.10)** from the context menu.



Rhapsody selects and high-lights the corresponding test scenario.



To open the test scenario double-click on the item “ATG\_TestCase.10”. The test scenario is a stereotyped sequence diagram.

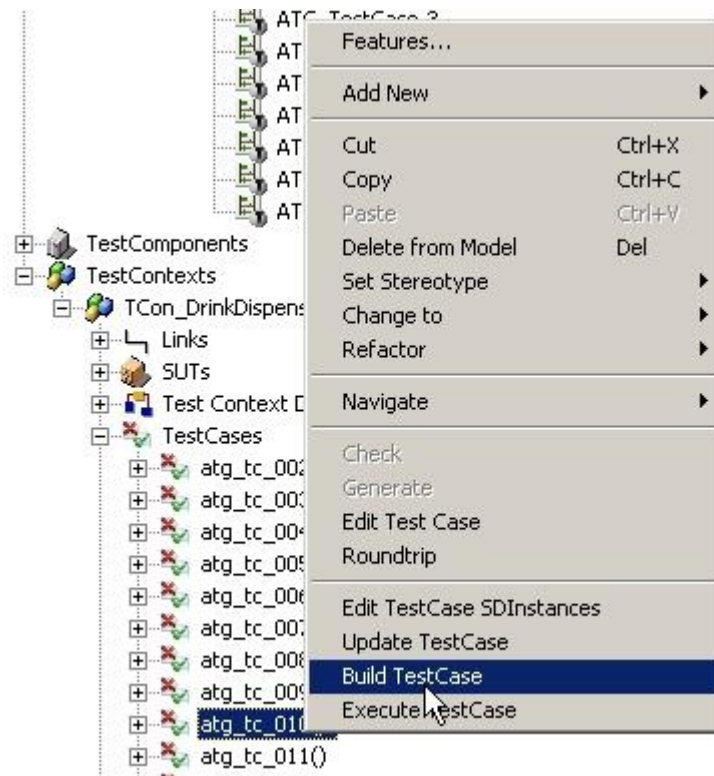


To get familiar with the created structures and the generated test cases, open some test cases and test scenarios. Test scenarios are merely sequence diagrams, which will be executed by the test cases with TestConductor as shown in the next section exemplarily.

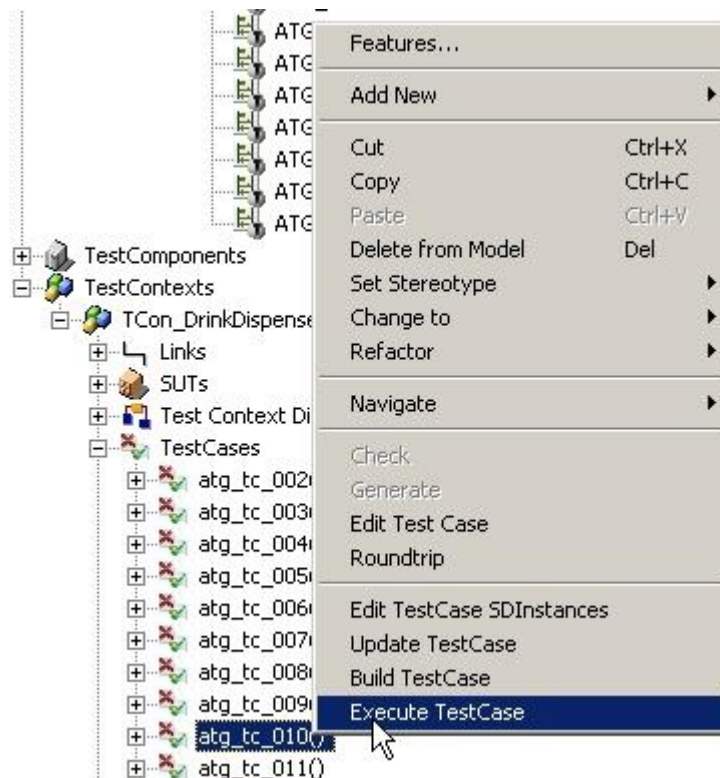
## Execute an ATG Test Case

To execute the test case “ATG\_TestCase\_010” we will start TestConductor. The first step is to compile the test case and link it to an executable. Execute the following steps:

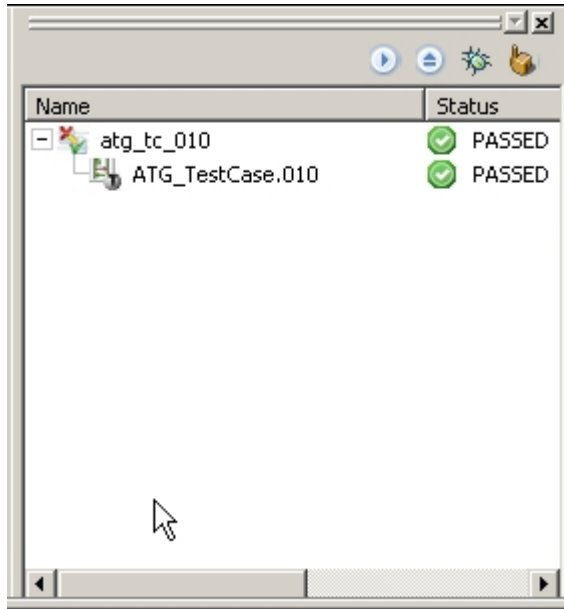
- ◆ To build the single test case “ATG\_TestCase\_010” right-click on a test case “ATG\_TestCase\_010” in the Rhapsody browser and select from the context menu **Build TestCase**.



- ◆ To execute the single test case “ATG\_TestCase\_010” right-click on the test case “ATG\_TestCase\_010” in the Rhapsody browser and select from context menu **Execute TestCase**.



TestConductor finished the test case execution successfully.



For further information on how to execute all test cases of a test context, a test package or in a batch mode reference the Rhapsody *TestConductor User Guide*. You will also find information how to find and correct model and code errors during test execution with the TestConductor feature **Show as SD**.

# Test Case Generation for Integration Testing

---

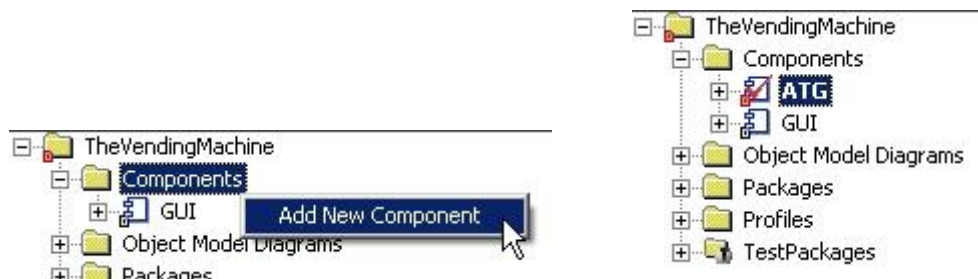
Compared to unit testing the following task of integration testing has a completely different scope. One goal is to leave the model untouched, and we will work not on test packages, test context etc., but construct this time manually a test component and a corresponding test configuration. This means the tests will execute the original integrated classes and objects. In other words, all the classes and objects integrated in such a test component will be considered to be the SUT.

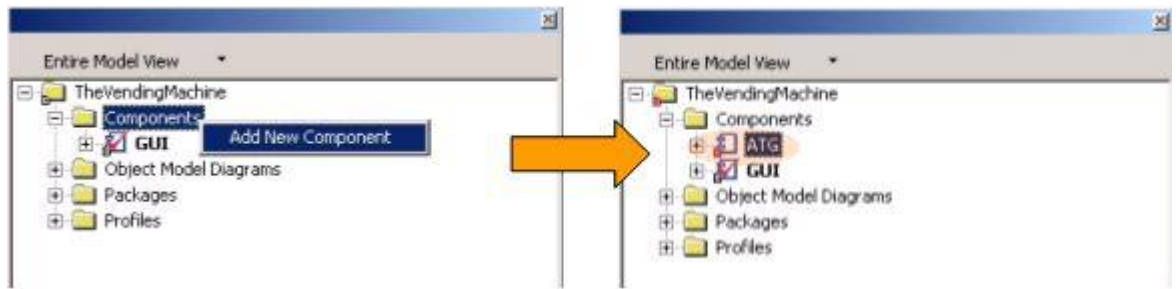
## Manually Creating a Testing Component

A Testing Component describes the scope of a system under test. It contains all classes and packages that must be considered for test case generation. Testing Components are derived from Rhapsody code generation components and configurations (see Rhapsody documentation [Creating a Component](#) and [Creating a Configuration](#)).

To use the integration test capabilities of ATG, you must define at least one Rhapsody component to ensure an executable code generation on the host. In other words, if a Rhapsody component is defined, which can be used to generate C++ code without any errors, and the compiled unit is executable in the simulator, this component can be used for automatic test case generation.

Given the original TheVendingMachine model, create a new Rhapsody component called ATG.





**Figure 5: Creation of a New Rhapsody ATG Component**

Specify the following component features (see Figure 10):

1. Name: **ATG**
2. Directory: **ATG**  
This is the default value.
3. Type: **Executable**  
This is the default value, which is correct for this example.
4. Scope: **Selected Element**: “Default”  
This is necessary for this example.

To build a valid code generation component for the whole user guide example, define the Configuration settings (double-click the `DefaultConfig` component) as follows (see Figure 6):

1. Initialization tab settings:
  - For the **Initial instances** field, select **Explicit**.
  - Select **VendingMachine** in the tree of instances.
  - Select **Generate Code For Actors**.
2. Settings tab settings (mandatory, especially for ATG):
  - For the **Instrumentation** field, select **Animation**.
  - For the **Time Model** field, select **Simulated**.
  - For the **Statechart Implementation** field, select **Flat**.
  - Accept the default values for the **Environment**, **Build Set**, **Compiler Switches**, and **Link Switches** fields.
  - In addition, under **Instrumentation**, click **Advanced**. In the Advanced Instrumentation Settings dialog box, select **All** for the **Enable Operation Calls** field. This enables you to call operations such that you can test the application.

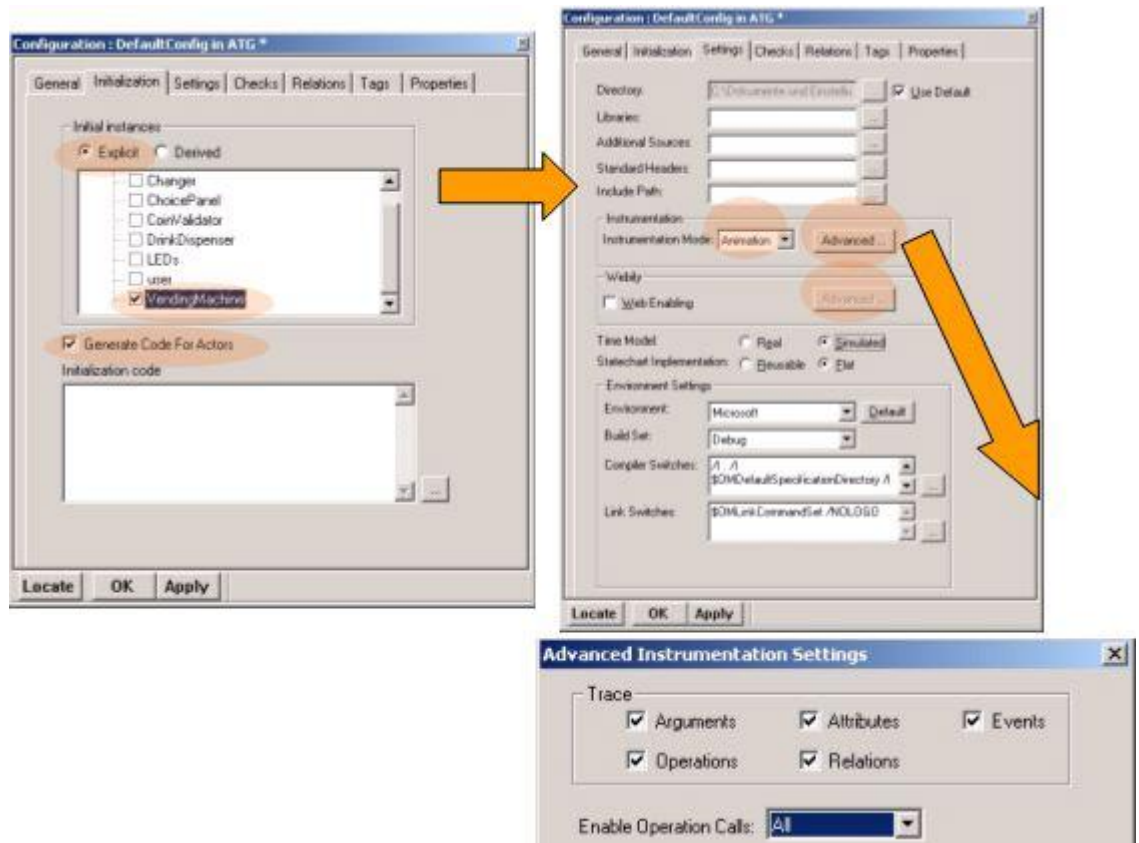


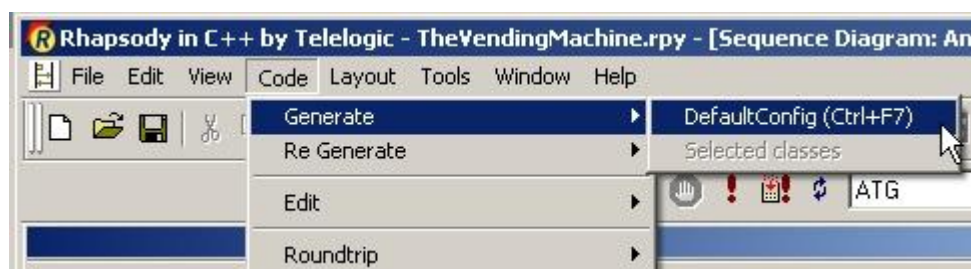
Figure 6: Rhapsody ATG Testing Component Configuration Settings

**Note:** If the model you want to test has already an executable configuration, you can either use this configuration directly or you can create a copy of this configuration in order to apply ATG. The steps described above are only necessary if there is no executable configuration in your model.

## Generate and Build the Test Component

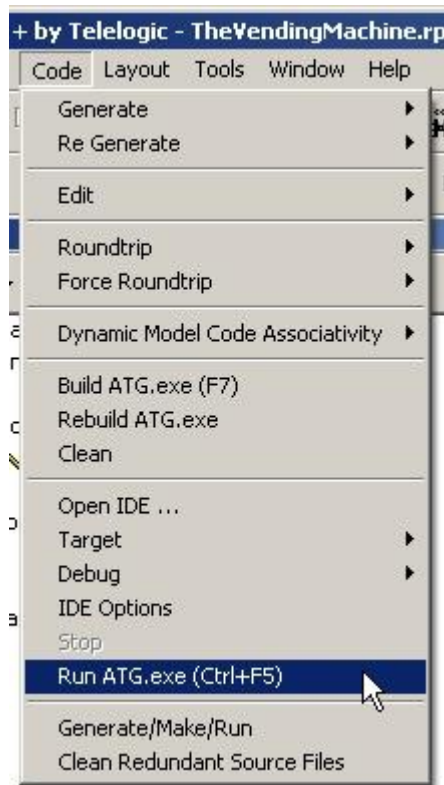
The new component was made automatically the active component. To ensure the new component was setup correctly generate and build the configuration.

- ◆ To generate the code select from the **Code** menu the item **Generate > DefaultConfig**.



**Figure 7: Code Generation and Compilation of the ATG Component**

- ◆ To execute the component select from the **Code** menu the item **Run ATG.exe**.

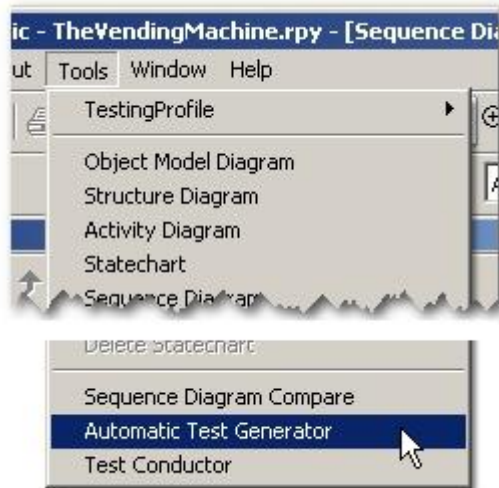


**Figure 8: Code Generation and Compilation of the ATG Component**

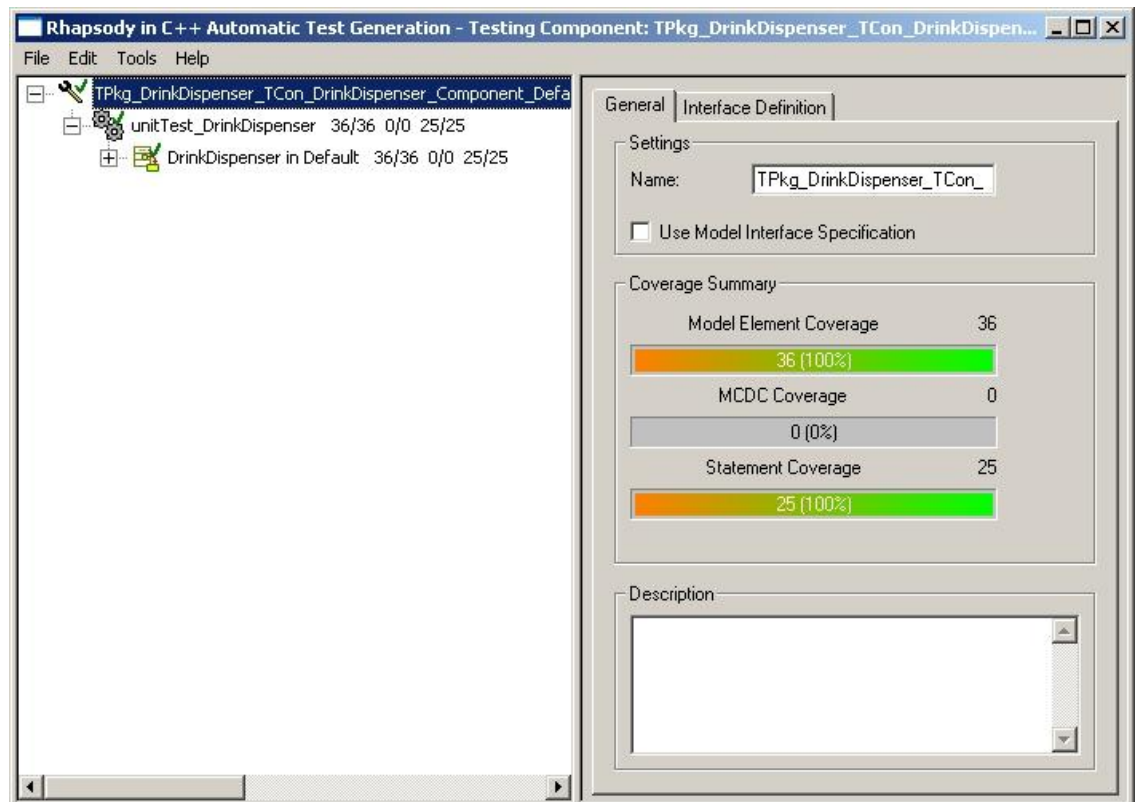
## Apply ATG

Once the component has been compiled into an executable unit, you can use ATG to generate tests.

- ◆ In Rhapsody, select **Tools > Automatic Test Generator**.



The main ATG window appears. Showing the unit test ATG configuration we used during unit testing before. It essential to create a new ATG configuration to fit the needs for integration testing on the created test component.



**Figure 9: Starting ATG**

In the main ATG window, select **File > New > Testing Component**. Another window pops up that asks the user to select the Rhapsody Configuration to which the newly

generated Testing Component should belong. Select the newly created configuration ATG::DefaultConfig.

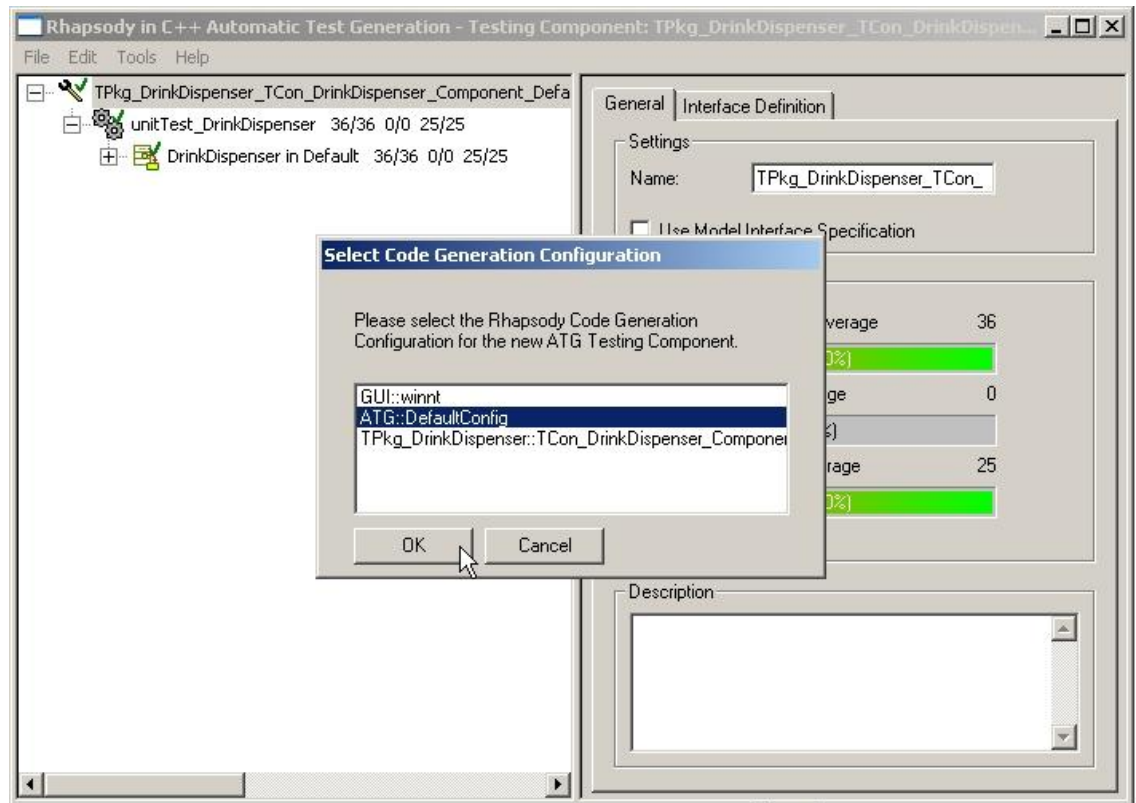
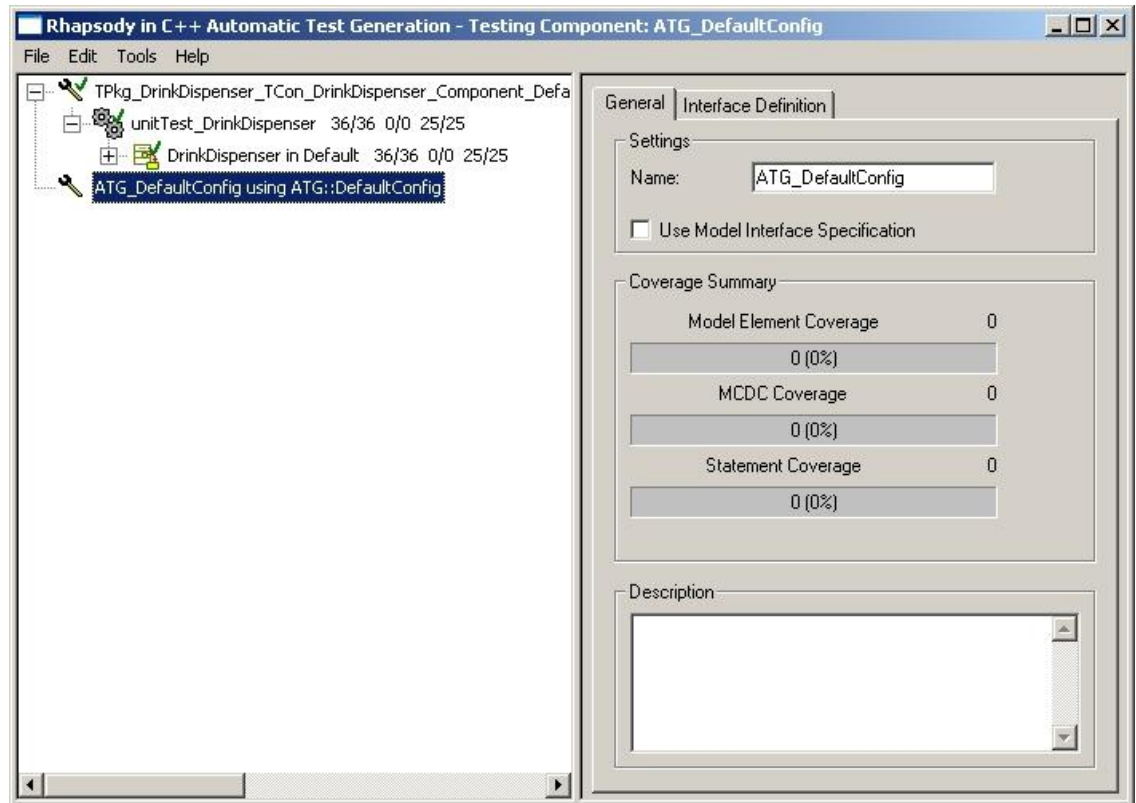


Figure 14: Selecting Rhapsody Configuration for ATG Testing Component.

## Test Generation Configuration

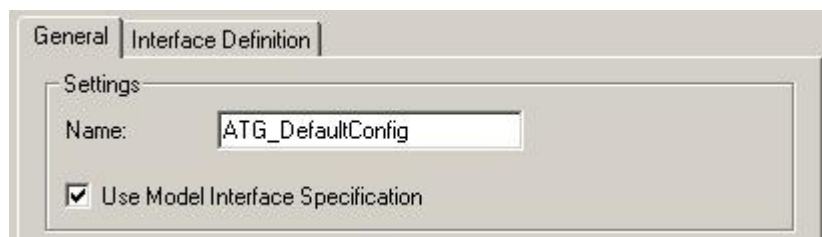
After pressing **OK**, the main ATG window shows the newly created Testing Component.



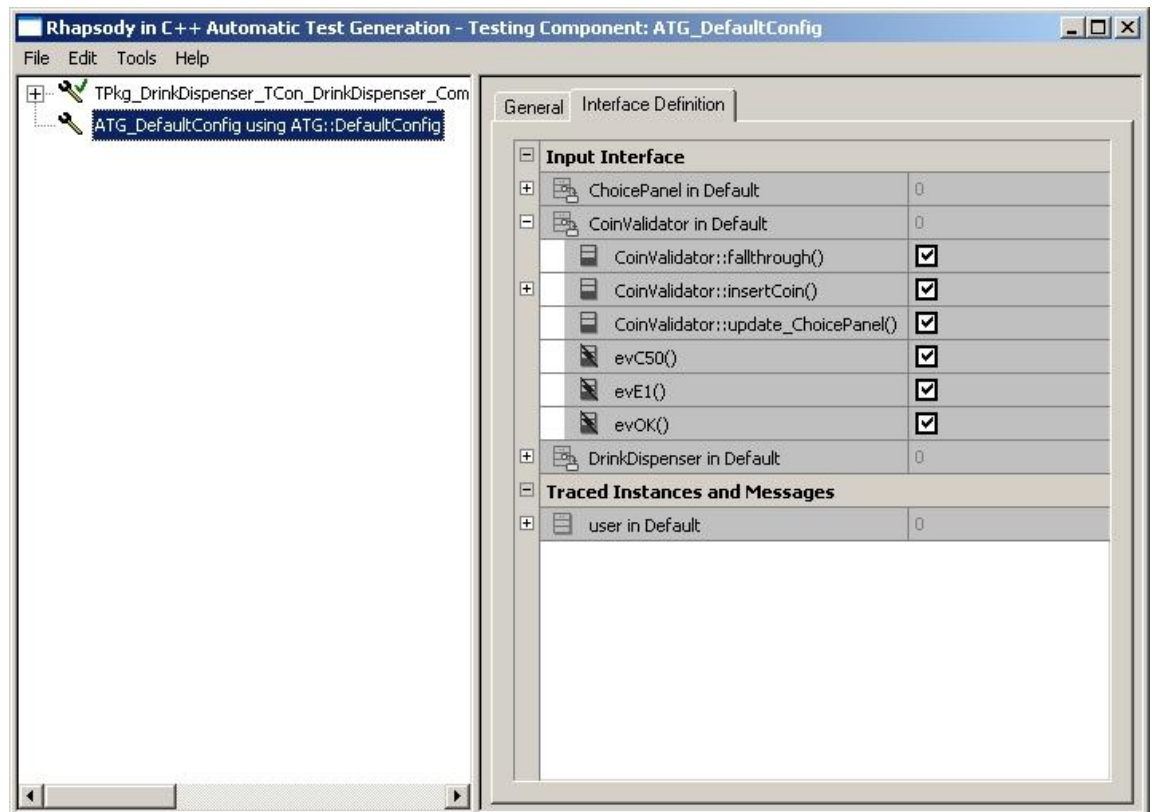
**Figure 15: Selecting Rhapsody Configuration for ATG Testing Component**

As shown in figure 16, `ATG_DefaultConfig using ATG::DefaultConfig` is displayed in the ATG browser. `ATG` is the Rhapsody component, and `DefaultConfig` is the Rhapsody configuration that belongs to the `ATG` component. `ATG_DefaultConfig` defines a *Testing Component* for `ATG`. Testing Components are the basic entities for actual test case generation. They define the scope of the model that shall be considered, i.e. the set of packages and classes contained in the scope.

TheVendingMachine example has some classes which provide an interface to an external user. Associations to and from these classes are labeled in the UML model as either provided or required interfaces. ATG provides a mode in which only these classes are shown in the ATG GUI. This mode can be switched on and off separately for each Testing Component by checking **Use Model Interface Specification** on the General tab of the `ATG::DefaultConfig` component as shown in Figure 17.



To view the filtered interface classes, select the Interface Definition tab.

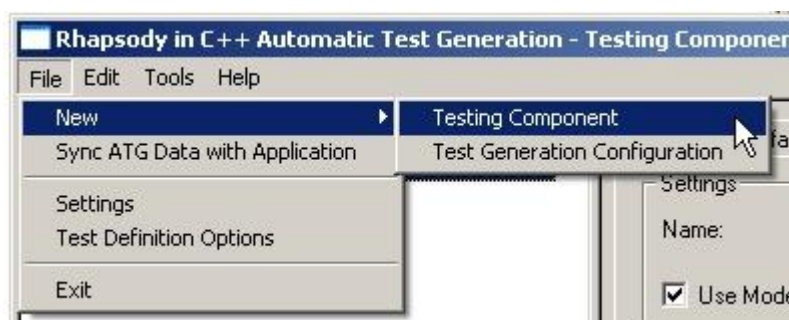


**Figure 16: Model-Derived Interface in ATG on the Testing Component Level**

If you do not set the check-box **Use Model Interface Specification** all classes contained in the Testing Component will be listed, for instance also class Changer.

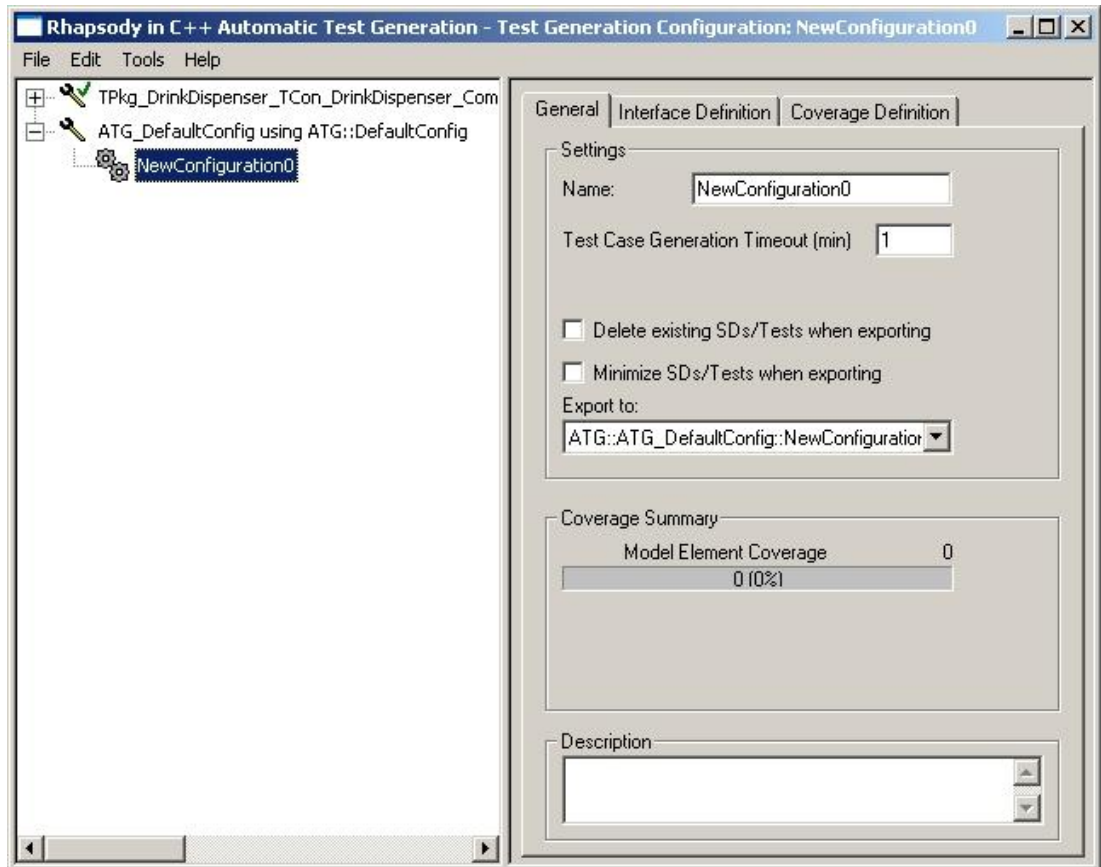
**Note:** In ATG 3.0, in the interface tab of ATG main window “provided interface” is named “Input Interface” and “Required Interface” is named “Traced Instances and Messages”.

After selecting a Testing Component “ATG\_DefaultConfig”, select **File > New > Test Generation Configuration**.



**Figure 18: Creating a Test Generation Configuration**

The new entry is displayed in the browser under the Testing Component.



**Figure 19: Test Generation Configuration Dialog**



























Next step is to specify a name for the new configuration. It shall be named as `integration_test`, because this is the use case considered in this test case generation. You can specify this definition in the Settings section of the General tab of the Test Generation Configuration dialog box. You can also specify the following fields:

- ◆ **Test Case Generation Timeout (min)**—Specifies a timeout value. By default, this is set to 1, which means 1 minute run-time unless you stop the generation, or the ATG algorithm cannot cover more model elements. Specify 2 minutes in this case as shown in Figure 19.
- ◆ **Delete existing SDs/Tests when exporting**—Specifies whether to delete any existing sequence diagrams when you export the tests (to Rhapsody or TestConductor). In the user guide, this option is set because only the latest results are of interest.
- ◆ **Minimize SDs/Tests when exporting**—Specifies whether the set of sequence diagrams/Test Cases that shall be minimized when you export the tests (to Rhapsody or TestConductor).

- ◆ **Export to**—Specifies the package of the resulting test cases (sequence diagrams). This user guide uses the default value.

Click on the Testing Component name to accept the specified values.

Use the Interface Definition tab of Test Generation Configuration `integration_test` to specify the model-derived interface classes. Click the plus symbol (+) corresponding to the class to view the list of all class public operations and events. Each operation or event has its check box, which is used for user interface definition. Each checked (✓) box of the **Input Interface** section of a class operation or event enables ATG to invoke the operation or event call during test case generation. Each checked box of the **Traced Instances and Messages** section of a class operation or event enables ATG to record it during test case generation.

General			Interface Definition			Coverage Definition		
Input Interface								
ChoicePanel in Default						0		
 ChoicePanel::ChoicePanel()						<input type="checkbox"/>		
 ChoicePanel::disable_all()						<input type="checkbox"/>		
 ChoicePanel::enable_Soft()						<input type="checkbox"/>		
 ChoicePanel::enable_Tea()						<input type="checkbox"/>		
 ChoicePanel::enable_Water()						<input type="checkbox"/>		
 ChoicePanel::getSoft_enabled()						<input type="checkbox"/>		
 ChoicePanel::getTea_enabled()						<input type="checkbox"/>		
 ChoicePanel::getWater_enabled()						<input type="checkbox"/>		
 evSOFT()						<input checked="" type="checkbox"/>		
 evTEA()						<input checked="" type="checkbox"/>		
 evWATER()						<input checked="" type="checkbox"/>		
CoinValidator in Default						0		
 CoinValidator::fallthrough()						<input type="checkbox"/>		
 CoinValidator::insertCoin()						<input checked="" type="checkbox"/>		
 int coinValue						50,100		
 CoinValidator::update_ChoicePanel()						<input type="checkbox"/>		
 evC50()						<input type="checkbox"/>		
 evE1()						<input type="checkbox"/>		
 evOK()						<input type="checkbox"/>		
DrinkDispenser in Default						0		
 DrinkDispenser::Prepare_Soft()						<input type="checkbox"/>		
 DrinkDispenser::Prepare_Tea()						<input type="checkbox"/>		
 DrinkDispenser::Prepare_Water()						<input type="checkbox"/>		
 evD50FT()						<input type="checkbox"/>		
 evDTEA()						<input type="checkbox"/>		
 evDWATER()						<input type="checkbox"/>		
 evFILLUP()						<input checked="" type="checkbox"/>		
Traced Instances and Messages								
user in Default						0		
 user::giveDrink()						<input checked="" type="checkbox"/>		

Vending Machine Input:  
User presses buttons

Vending Machine Input:  
User inserts coin

Vending Machine Input:  
Administrator fills machine

Vending Machine Output:  
User gets drink

**Figure 10: Interface Definition of the integration\_test Configuration**

The field to the right of the class name defines the class instances, which ATG takes into account during test case generation. In this example, only the first class instance for every interface class is used (0). This means that ATG can call operations and send events to

ChoicePanel[0], CoinValidator[0], and DrinkDispenser[0]. The desired interface of the integration test of TheVendingMachine example has to be entered as shown in Figure 20.

Operations and events usually have parameters (arguments). Click the plus sign to display the list of parameters, which can be restricted or defined for each run of ATG. This definition can be an enumerated list (for example, 50, 100) or a definition of a value range (for example, 0 -100). The default settings for the parameter ranges are derived from the Rhapsody model. ATG tries to extract as much information about the parameter ranges as possible. If information is missing, ATG highlights those fields in red. You must add this missing information into the ATG dialog.

Before you can activate test case generation, ATG must know the desired test goals. You define this in the **Coverage Definition** field of the Test Generation Configuration dialog box. The default setting is that no model classes (**Coverage of Classes**) and no **Events** will be covered by ATG. This means that no test cases will be generated.

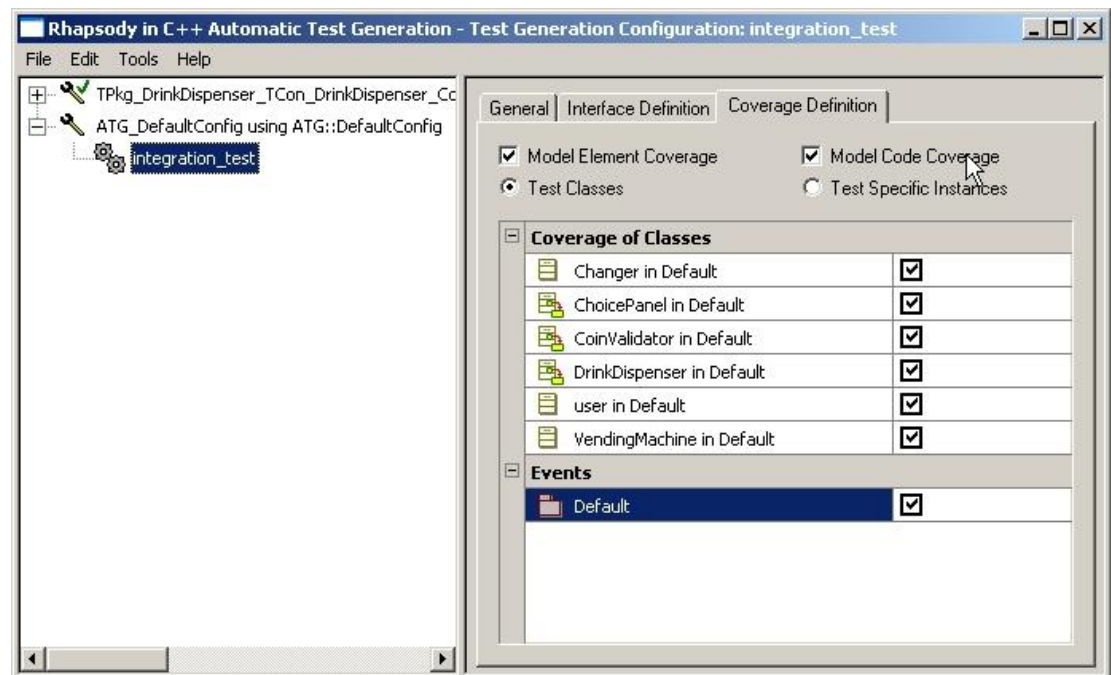


Figure 11: Coverage Definition

Each class and package of the Rhapsody model is listed. To add classes to the list of test goals, add the tick in the check box on the right side of the class row. To remove classes from the list of test goals, delete the tick from the check box on the right side of the class row.

When you switch to **Test Specific Instances** you can specify which of the individual instances of a class shall be covered by ATG. If an instance (such as 0) or list of instances (for example, 0, 1, 5-7) is defined, ATG tries to cover all the states and transitions (not the operations) that are part of the corresponding instances of the class. In addition, you can specify that events of the model are coverage test goals of ATG by selecting the relevant packages.

**Note:** It might happen that a specific instance (e.g. instance '0') cannot cover a particular test goal, e.g. a transition of a class. However, after test case generation

the ATG browser might show that this transition has been covered regardless of the fact that the selected instance '0' could not cover the operation. This can be the case due to other existing instances in the tested system which cover the transition. In this case the ATG browser shows this covered transition directly under the class and not under a specific instance of this class. The other transitions, which are covered by the selected instance, are shown in the specific browser part of the instance.

Check box **Model ElementCoverage** enables test case generation in order to cover states, transitions, and operations of the selected classes and instances. Check box **Model Code Coverage** enables test case generation in order to cover the conditions, decisions, and branches in the source code of the selected classes.

**Note:** For **Model Code Coverage** only those portions of the code are considered that has a associated model element visible in the Rhapsody browser. All implicitly generated code is not taken into account.

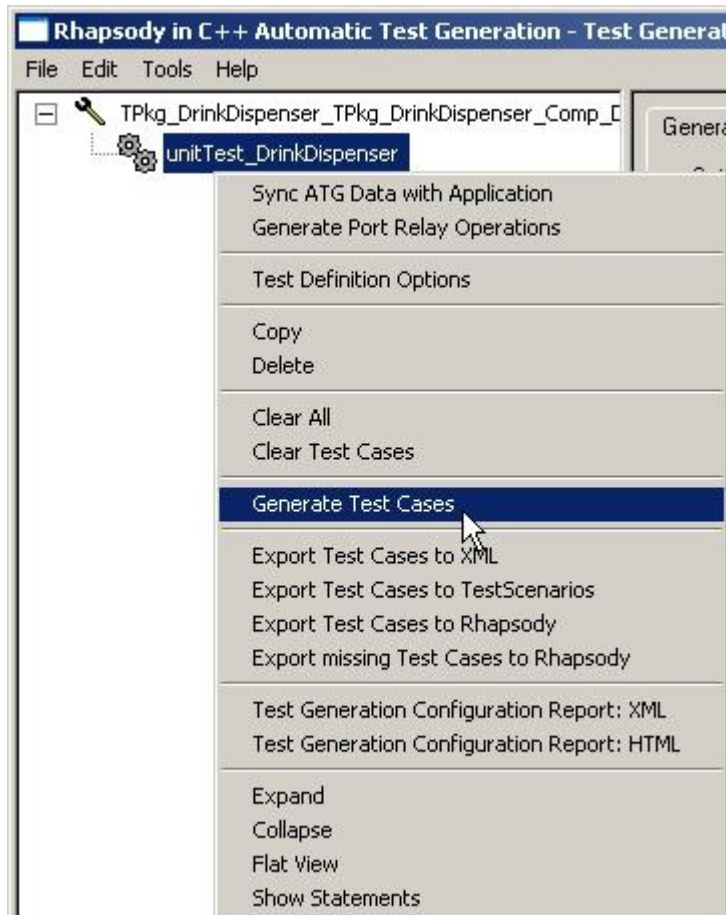
For the first Test Generation Configuration, define the test goals using the default settings, as shown in Figure 11. This Test Generation Configuration is ready for ATG test case generation.

## Generate, Export, and Execute Test Cases

Test Case Generation can be applied on single classes, a set of classes, or on a complete application. ATG supports the use cases Class Testing and Integration Testing.

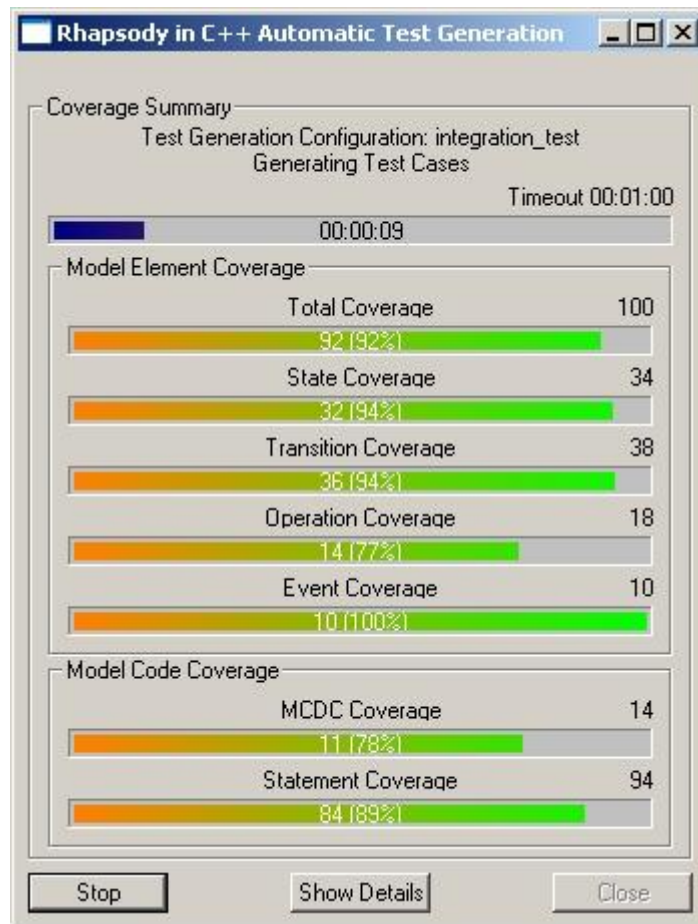
To generate test cases for a user-defined Test Generation Configuration,

- ◆ select the configuration `integration_test`, then select **Tools > Generate Test Cases**
- ◆ or right-click on the configuration `integration_test` and select from the context menu **Generate Test Cases**.



**Figure 12: Generation of Test Cases for whole Vending Machine model**

The test case generation virtual machine takes the C++ code, the interface definition, and the user definition of the desired test goals of the Test Generation Configuration `integration_test` and tries to cover all the goals. During execution, you can view the progress, as shown in the following Figure.



**Figure 13: Progress Dialog**

During the test case generation task of the ATG engine, the following information is given at any point in time during execution:

- ◆ User-defined Time-Out and current Status
- ◆ Elapsed Rhapsody ATG Execution Time
- ◆ Total number of Test Goals and the number of goals currently reached
- ◆ Number of States Activations and the reached number
- ◆ Number of Transition Firings and the reached number
- ◆ Number of performed Operation Calls and the reached number
- ◆ Number of Event Generations and the reached number
- ◆ Number of MDCDC Code Coverage Goals and the reached number
- ◆ Number of Statement Code Coverage Goals and the reached number

During and after execution on a Test Generation Configuration, you can access information about covered goals by clicking Details. The following Figure shows the coverage results after a one minute run on `integration_test`. Details about the covered goals can be seen by clicking the “Show Details” button.

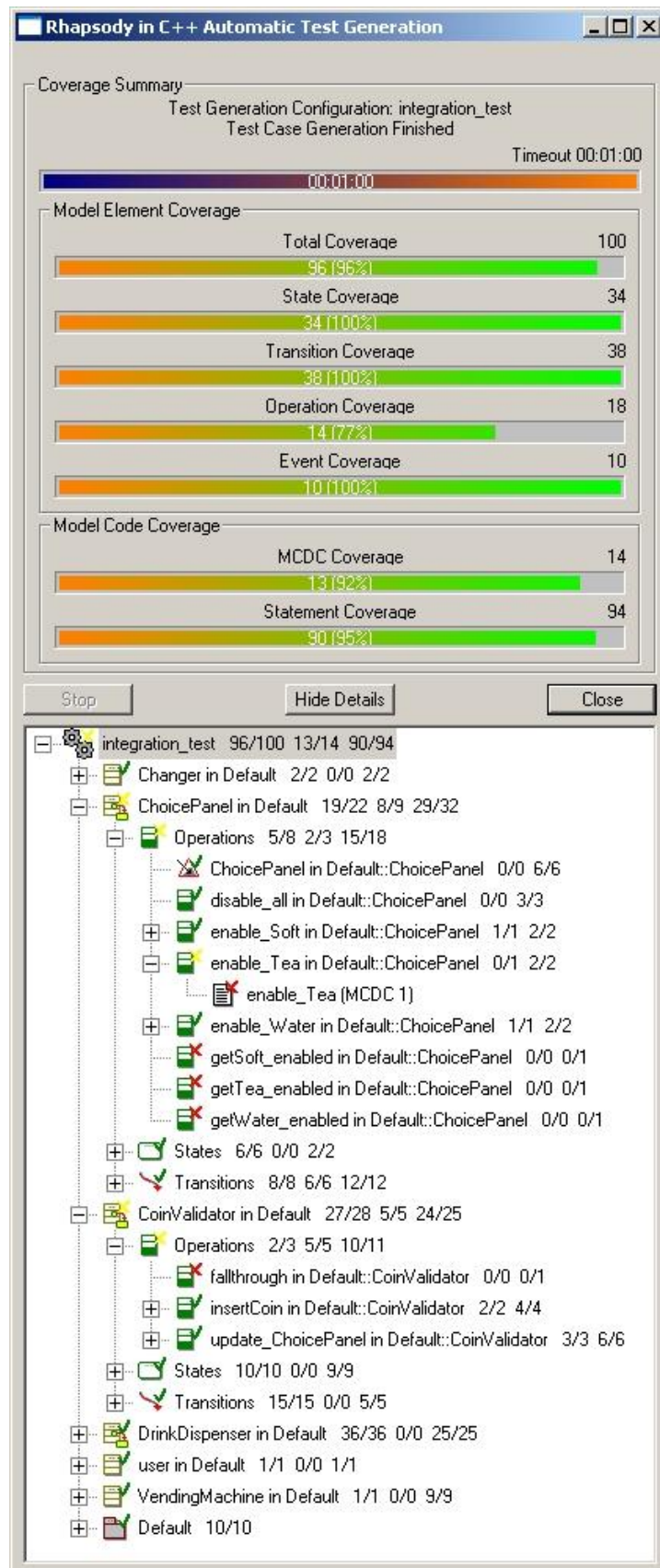


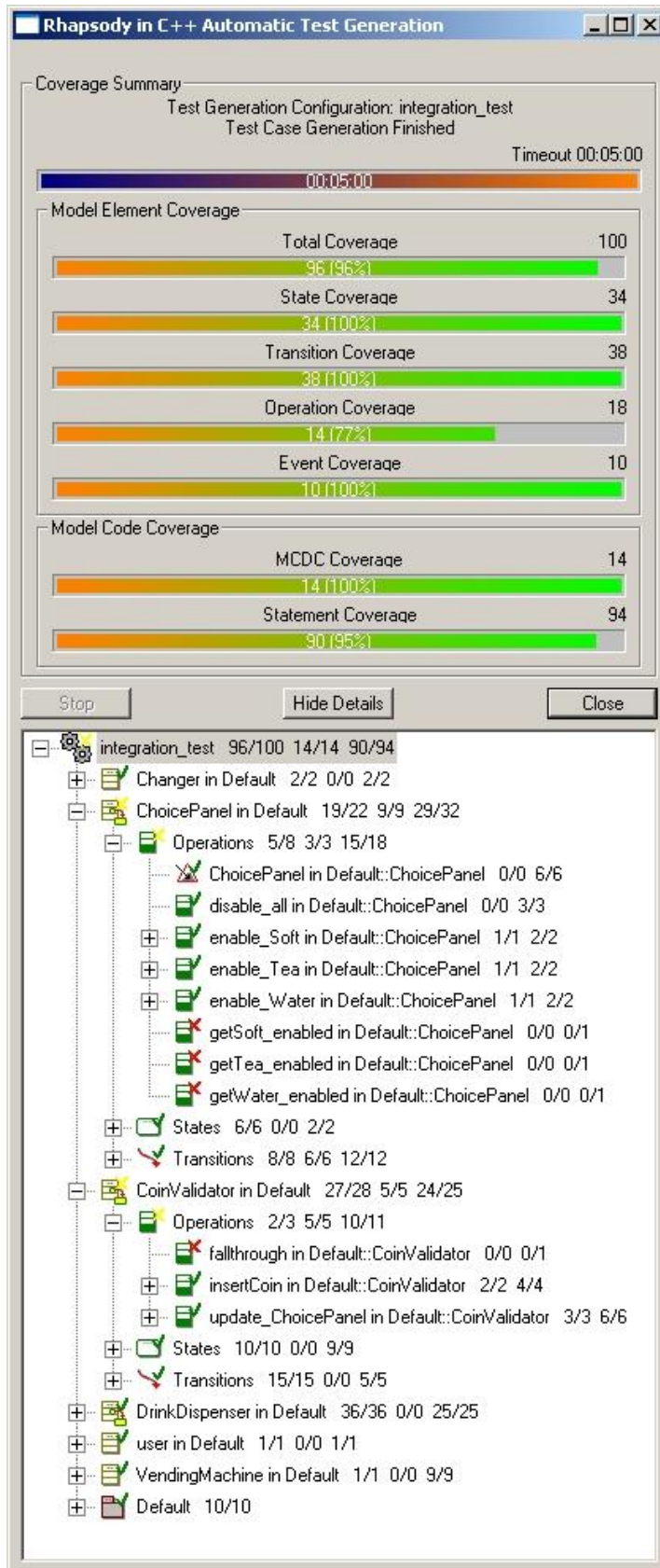
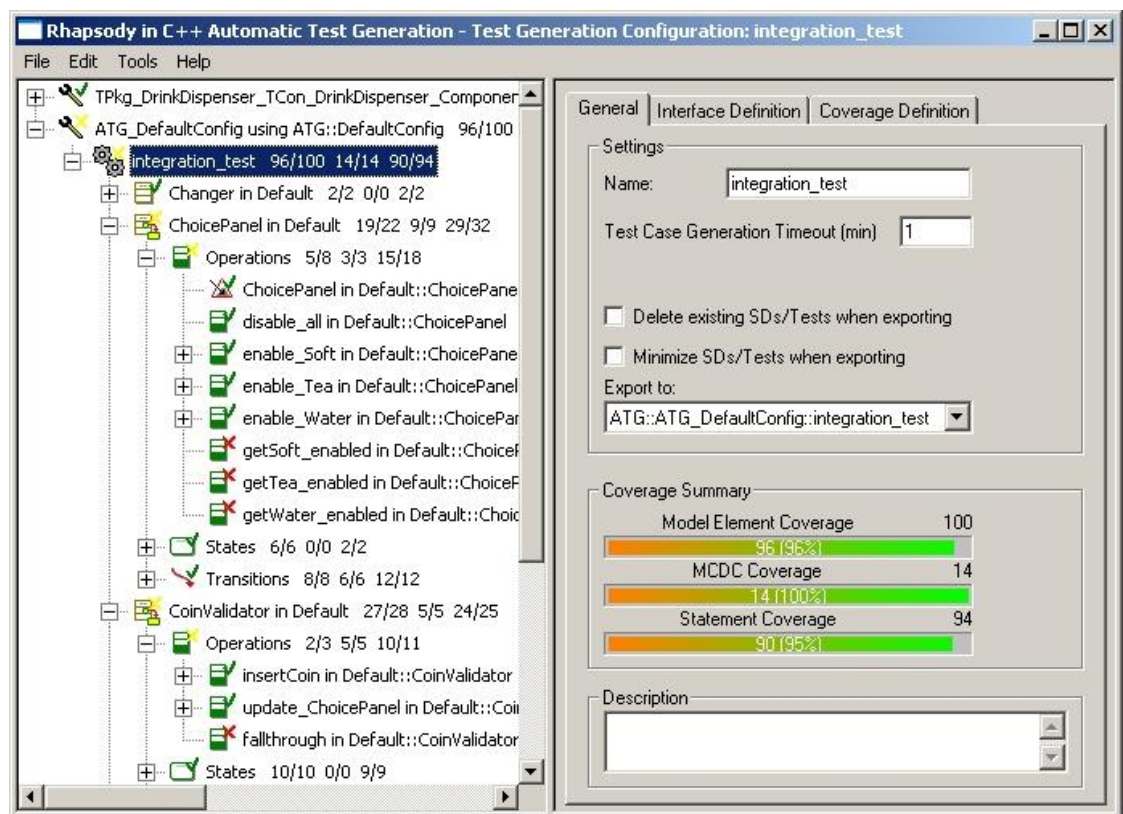


Figure 14: Detailed Coverage Information

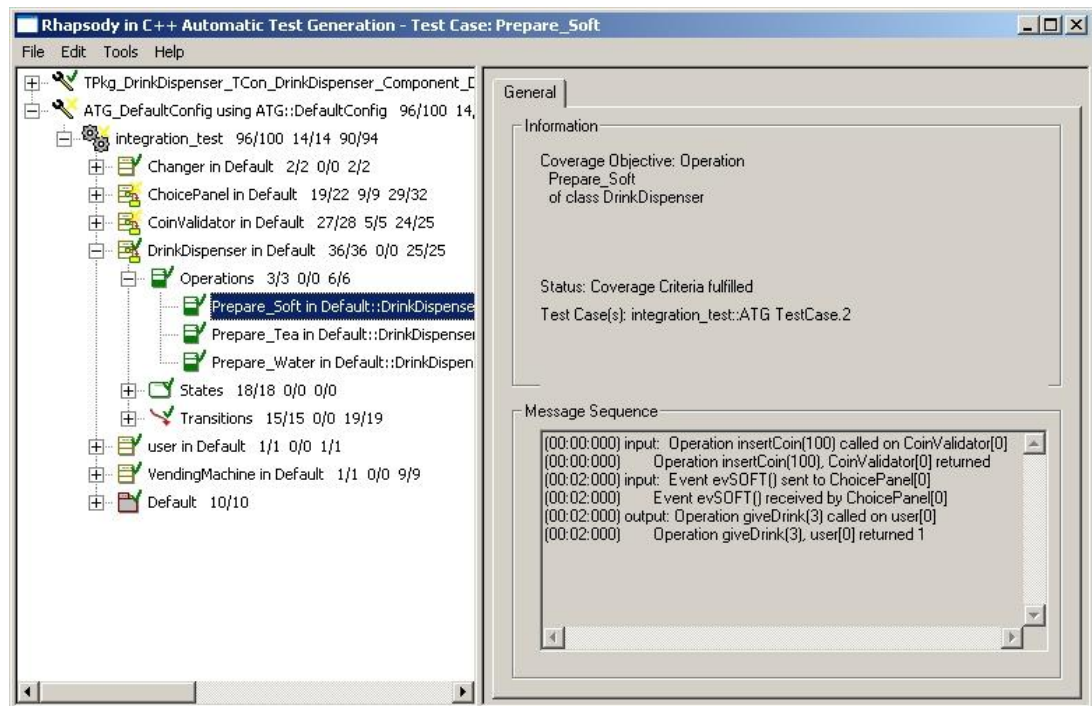
The detailed view can be expanded into a finer display by clicking the plus symbol [+]. The red  indicates uncovered goals, whereas green  symbols denotes successful coverage of a certain test goal. After execution, click **Close** to close the progress dialog and display the ATG main window.





**Figure 15: ATG Result Information**

All test goals and generated test cases are collected and managed under the integration\_test folder, which can be opened by clicking on the plus symbol [+]. The reached test goal coverage rate can be seen in the Coverage Summary section of the General tab of the Test Generation Configuration.



**Figure 16: Test Goal Management and Test Trace Inspection**

If you open the Test Generation Configuration folder, clicking on a test goal displays information about the generated test case. It explains where the test case is stored on the file system and provides details of the generated *message sequence*, which covers the selected test goal. The generated message sequence is time-annotated and can be exported into a Rhapsody sequence diagram. Again, a red **x** indicates uncovered goals, whereas a green **✓** denotes successful coverage of a certain test goal. You can double-click a test goal to show the corresponding model element directly in Rhapsody. For example, double-click the State goal called `Soft_empty` of class `DrinkDispenser` to open the corresponding statechart and highlight the state as shown in Figure 17. This same functionality is provided for all other test goals, including events and operations.

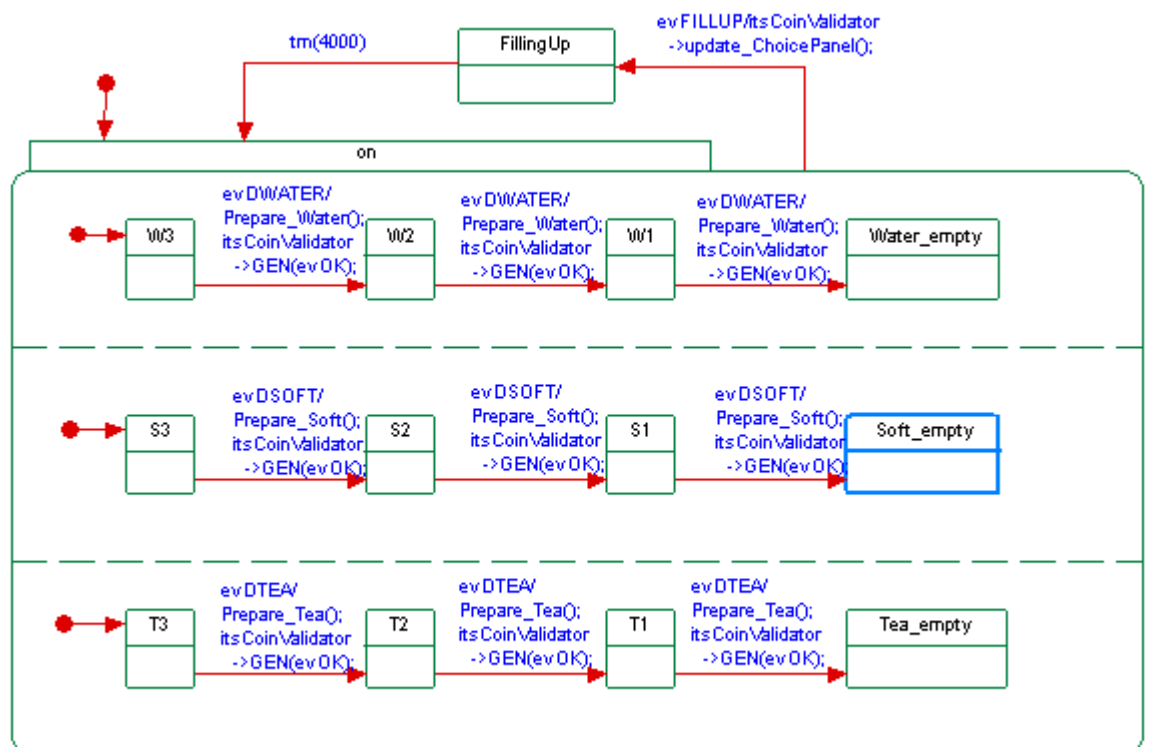
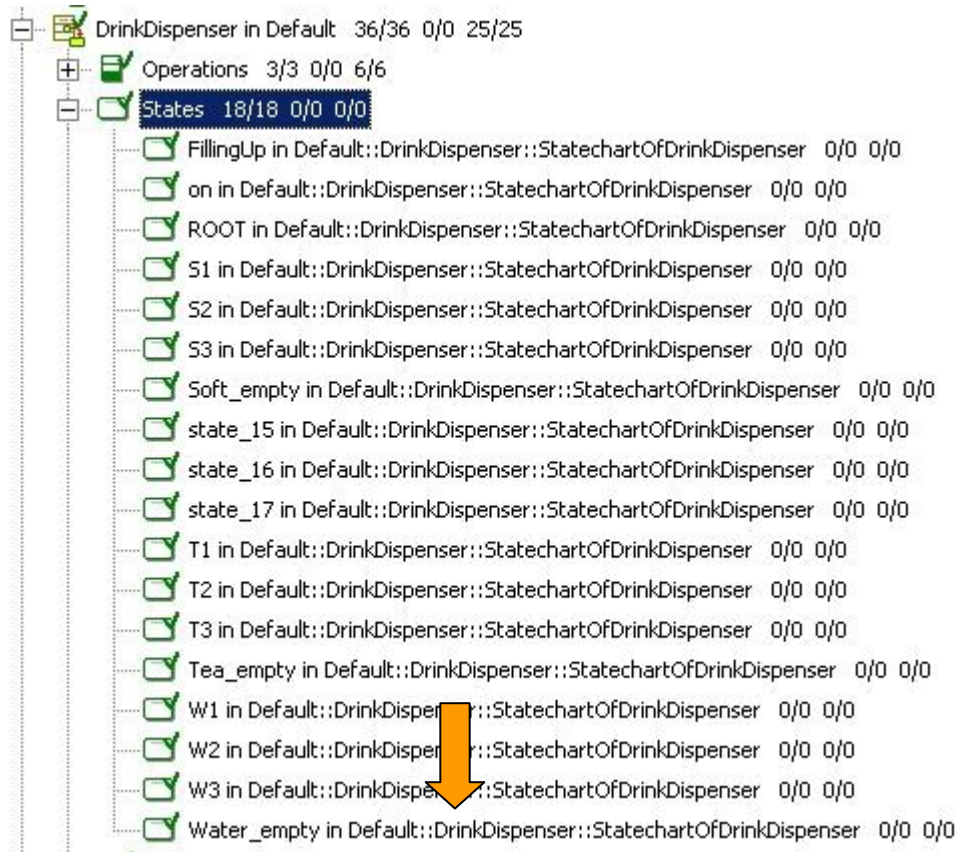


Figure 17: ATG Test Goal Connection to the Rhapsody Model

Using the model link reference feature, you can find out why the four uncovered goals are unreachable in the context of the specified interface.

# ATG Management

---

The test generation component and the test generation configurations specify necessary details for actual test generation. This entails precise interface definitions and selection of test goals. To generate test cases with ATG, you must create a test generation configuration based on a selected test generation component. While a test generation component describes the scope as explained in the previous subsections, test generation configurations specify some necessary details for actual test generation.

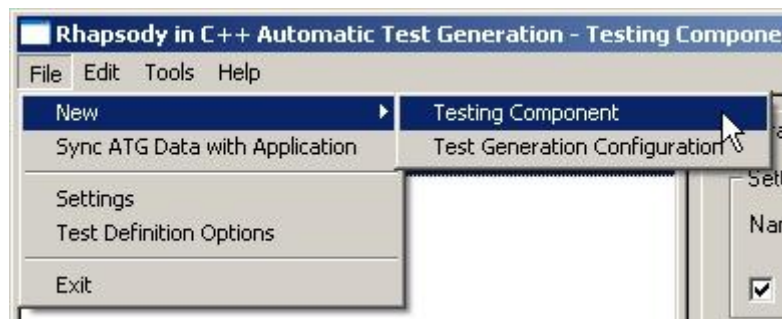
## The Test Generation Component

### Create a Test Generation Component

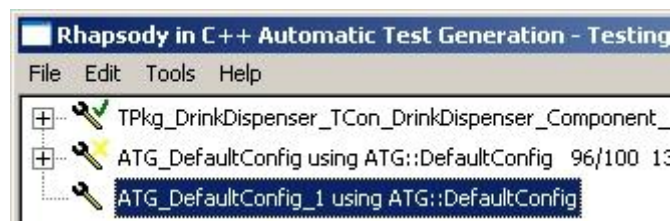
- ◆ To generate a test generation component right-click on an empty area in the ATG browser and choose from context menu **New Testing Component**.



- ◆ Or choose from the **File** menu **New > Testing Component**.

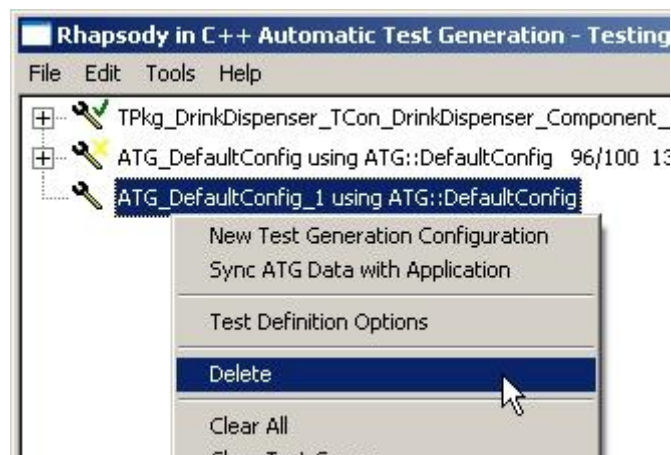


- ◆ Select in the dialog **Select Code Generation Configuration** the Rhapsody component to analyze with ATG and click **OK**

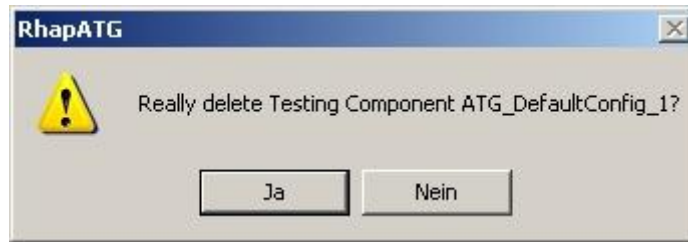


## Delete a Test Generation Component

- ◆ To delete a test generation component right-click on the test generation component to delete in the ATG browser and choose from context menu **Delete**.



- ◆ Or select the test generation component to delete in the ATG browser and choose from the **Edit** menu **Delete**.
- ◆ Or press the **DEL** key.
- ◆ Choose **Yes** to delete the test generation component. Select **No** not to delete test generation component.



In case the user confirms the deletion all configurations and data under the test generation component will be delete and are not restorable.

## Clear All

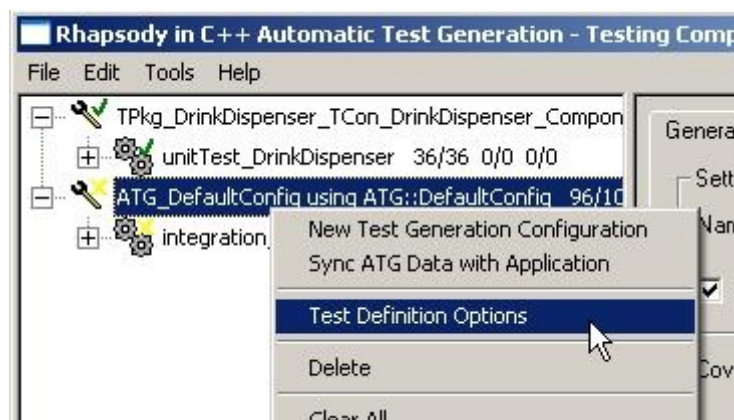
To delete previously generated test cases and also the selected options in all test generation configurations of a test generation component choose from context menu of a test generation component in the ATG browser **Clear All**.

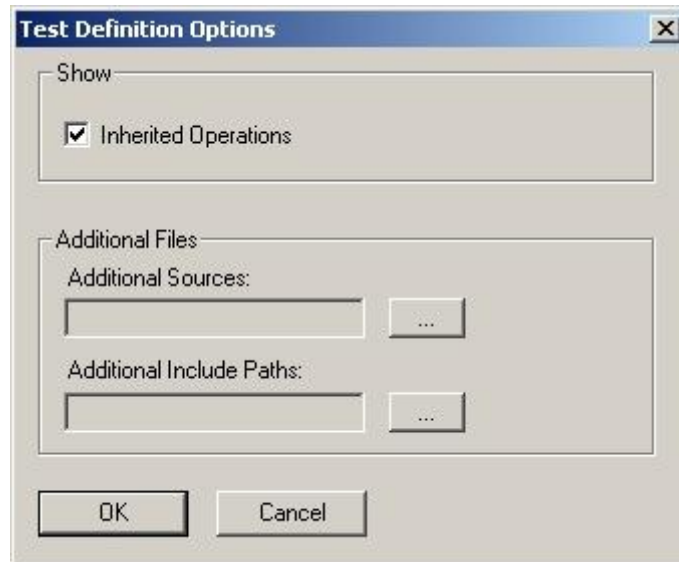
## Clear Test Cases

To delete only previously generated test cases, but to maintain the selected options of all test generation configurations of a test generation component choose from context menu of a test generation component in the ATG browser **Clear Test Cases**.

## Test Definition Options

- ◆ To open the test definition option concerning the test generation component right-click on the test generation component in the ATG browser and choose from context menu **Test Definition Options** or choose **File > Test Definition Options**.





The **Inherited Operation** selection box in the **Show** section provides the possibility to show operations from shown classes in the interface tab or coverage tab that are inherited from their base classes.

The **Additional Files** section give the user the possibility to define **Additional Sources** and **Additional Include Paths**. This is necessary if legacy code or header files from external libraries are used from the generated model code that are not part of the model.

To specify additional files click on the “...” button next to the **Additional Sources** field and write in the file name optionally including a relative or absolute path in windows notation. Relative entries will be interpreted from the path containing your project file (.rpy). Multiple file entries have to be separated by commas.

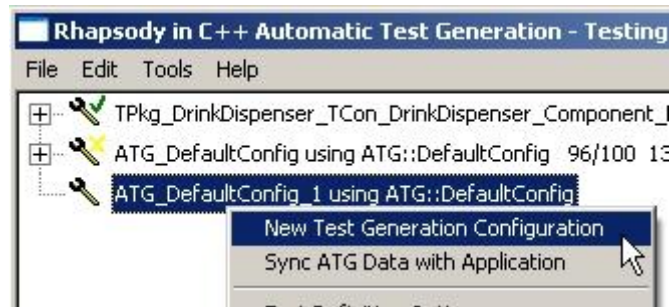
To specify additional include paths click on the “...” button next to the **Additional Include Paths** field and write in the path in windows notation. Relative entries will be interpreted from the path containing your project file (.rpy). Multiple path entries have to be separated by commas.

ATG recognizes per default the content of the `INCLUDE` system environment variable. This path definition will be extended under Rhapsody in the file `vcvars32.bat` in the folder “share\etc” of your Rhapsody installation.

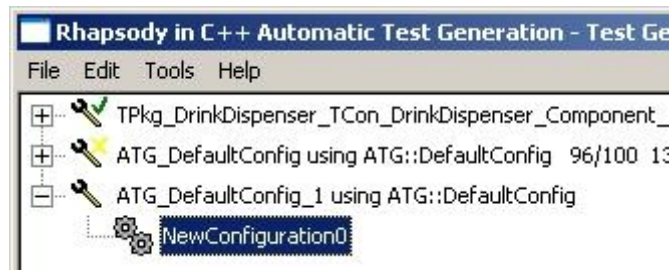
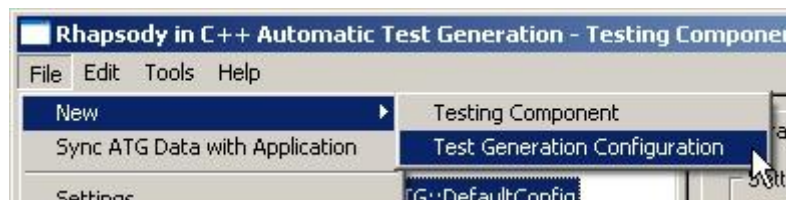
**Note:** You can not use self defined system environment variables e.g. `%myIncludes` directly in the ATG **Test Definition Options** dialog. For that scenario, please extend the `INCLUDE` environment variable directly.

## The Test Generation Configuration

- ◆ To generate a test generation configuration right-click on a test generation component in the ATG browser and choose from context menu **New Testing Configuration**.

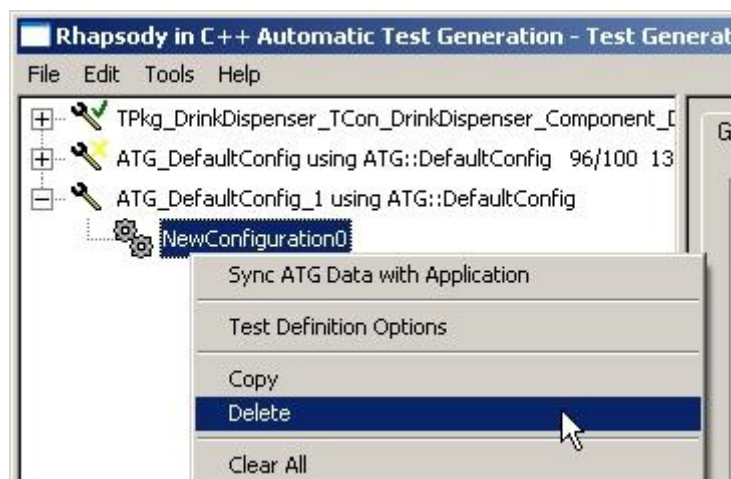


- ◆ Or select the test generation component in the ATG browser to create a configuration for and then choose from the **File** menu **New > Testing Configuration**.

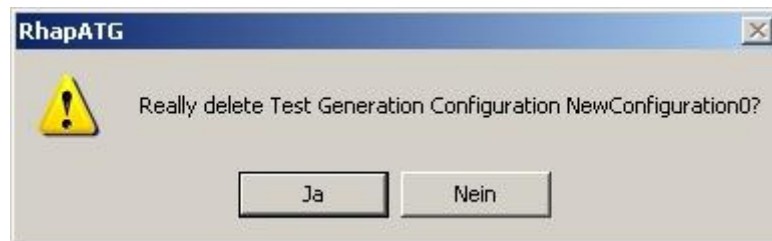


## Delete a Test Generation Configuration

- ◆ To delete a test generation component right-click on the test generation configuration to delete in the ATG browser and choose from context menu **Delete**.



- ◆ Or select the test generation configuration to delete in the ATG browser and choose from the **Edit** menu **Delete**.
- ◆ Or press the **Del** key.
- ◆ Choose **Yes** to delete the test generation configuration. Select **No** not to delete test generation configuration.



In case the user confirms the deletion all configurations and data under the test generation configuration will be delete and are not restorable.

## The General Definition Tab

The **General** tab of ATG defines the name of the configuration and provides a description box to notice its purpose.

The screenshot shows a configuration dialog box with three tabs: General, Interface Definition, and Coverage Definition. The General tab is active. It contains a 'Settings' section with a 'Name' field set to 'NewConfiguration0', a 'Test Case Generation Timeout (min)' field set to '1', and two unchecked checkboxes: 'Delete existing SDs/Tests when exporting' and 'Minimize SDs/Tests when exporting'. Below these is an 'Export to:' dropdown menu showing 'ATG::ATG\_DefaultConfig\_1::NewConfigural'. The 'Coverage Summary' section shows 'Model Element Coverage' at '0' with a progress bar indicating '0 (0%)'. The 'Description' section is a large empty text area.

In the **Settings** section the user can rename the selected test generation configuration. To rename the test generation configuration change the name in the field **Name** and change the focus. Only one word names are allowed. ATG will update the test generation configuration name in the ATG browser immediately.

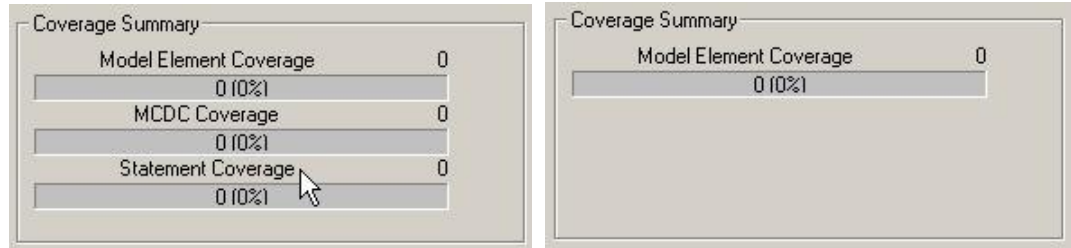
The field **Test Case Generation Timeout (min)** tells ATG how much time to spend finding the best coverage. The default is one minute. Increase the value when dealing with complex models.

The option **Delete existing SDs/Tests when exporting** prevents duplicated sequence diagrams in the model when re-exporting test cases from ATG.

The option **Minimize SDs/Tests when exporting** results in efficient and compact sets of test cases.

In drop-down-box **Export to** the user defines, into which test generation configuration to export the generated test cases.

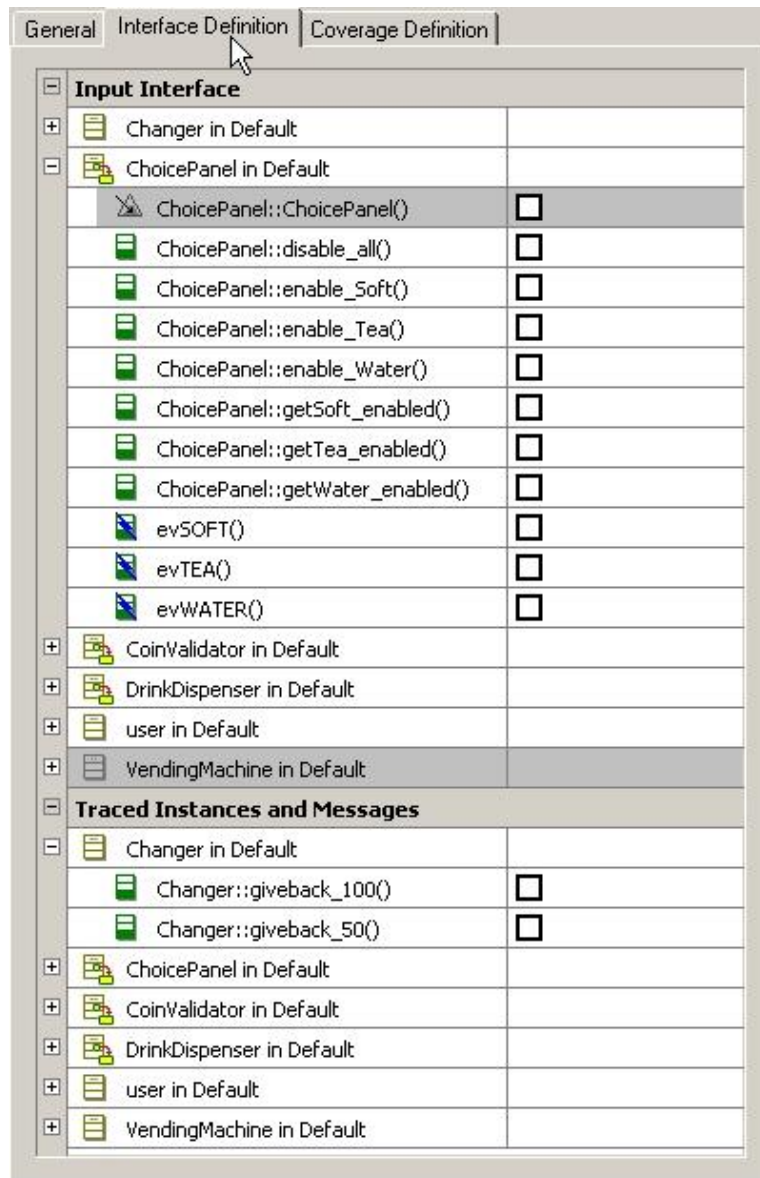
In the **Coverage Summary** section the user can read the coverage ATG reached with the last started test case generation run. The display varies depending on whether which option under the “Coverage Definition” tab is enabled.



In the **Description** section the user can notice the purpose of the test generation configuration. When? Who? What? Why? are the common questions.

## The Interface Definition Tab

The **Interface Definition** tab defines the inputs to be stimulated by ATG during test case generation and the outputs (traced message) used as expected messages of a test case

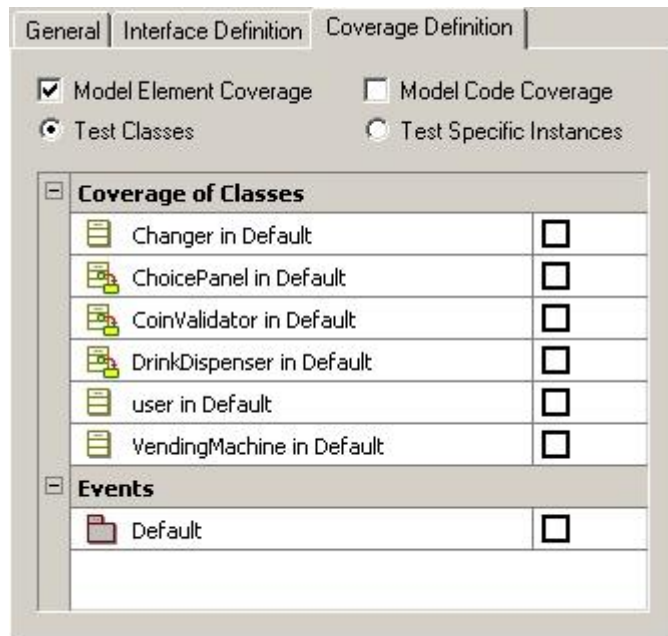


The **Input Interface** section under the **Interface Definition** tab defines the operations and events ATG is allowed to call on the SUT and the test components. TestConductor, the execution engine, will later act as driver for these operations and events.

The **Traced Instances and Messages** section under the **Interface Definition** tab of ATG defines the operations and events ATG has to trace on an incorporated instance. TestConductor, the execution engine, will later act as observer for these operations and events.

## The Coverage Definition Tab

The **Coverage Definition** tab of ATG defines the target classes and events ATG will analyze in terms of the defined SUT.



The option **Model Element Coverage** tells ATG to achieve model element coverage. Model elements are State, Transitions, Operations and Events.

The option **Model Code Coverage** tells ATG to achieve MC/DC and statement code coverage goals for the code parts which have been added to the model by the user.

Furthermore the coverage analysis can be done only for the selected **Test Classes** or for **Specific Instances**. Specific instances means that ATG generates test cases for all instances of selected classes instantiated during ATG execution.

The section **Coverage of Classes** is used to define the classes ATG has to use for test goal computation. Test goals are all model element coverage goals and model code coverage goals.

The section **Events** show all package in which Events are defined in the model. All Events of a selected package have to be covered by ATG during Test Case Generation.

## Clear All

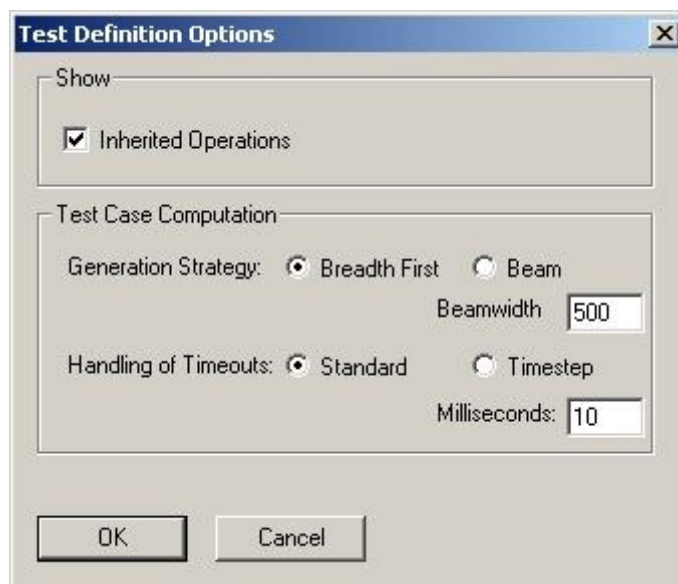
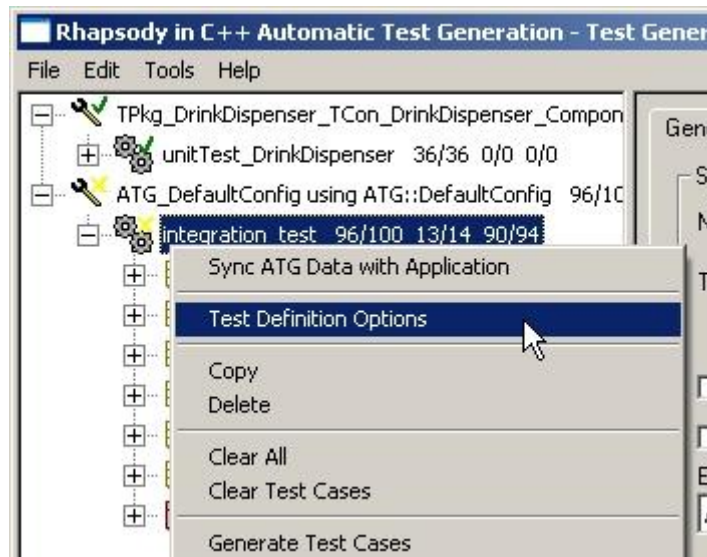
To delete previously generated test cases and also the selected options in a test generation configuration choose from context menu of a test generation configuration in the ATG browser **Clear All**.

## Clear Test Cases

To delete only previously generated test cases, but to maintain the selected options of a test generation configuration choose from context menu of a test generation configuration in the ATG browser **Clear Test Cases**.

## Test Definition Options

- ◆ To open the test definition option concerning the test generation configuration right-click on the test generation configuration in the ATG browser and choose from context menu **Test Definition Options** or choose **File > Test Definition Options**.



The **Inherited Operation** selection box in the **Show** section provides the possibility to show operations from shown classes in the interface tab or coverage tab that are inherited from their base classes.

The **Test Case Computation** section provides setting to control the used ATG engine and the timeout handling. Two **Generation Strategies** are provided by ATG:

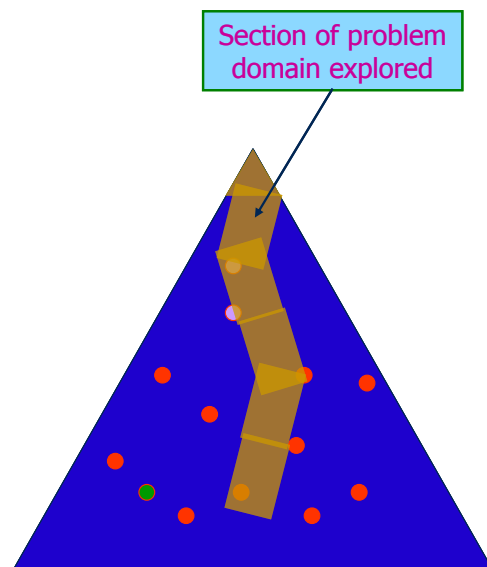
### **Breadth First search:**

Breadth-first generation means that always the shortest test cases that fulfils a particular coverage goal has to be found. ATG analyses the complete breadth of the model. But this also means that the number of states that have to be explored grows exponentially with the search depth.

### **Beam search:**

In beam search the number of states grows only linearly with the search depth. A beam search can find coverage goals with long paths lengths which cannot be found when using breadth-first search. However, the test case found by beam search might not be the shortest test case that leads to the coverage goal.

The value beamwidth sets the breadth of the beam



Also the **Handling of Timeout** can be set. Always when the system is idle which means that no further computation is possible w/o new inputs ATG is doing a timestep to find out whether there are timeout transitions to be fired.

**Standard** means that ATG uses the shortest defined timeout in the model as a timestep during test case generation. For example if there are two timeouts in the model, one with 10ms and one with 1000ms, ATG have to do 100 timesteps to fire the 1000ms timeout transition.

**Timestep** with the corresponding value gives the possibility to define a time step. For example if the time step is set to 1000ms all timeouts (10ms, 100ms ...) are fired in on ATG time step.

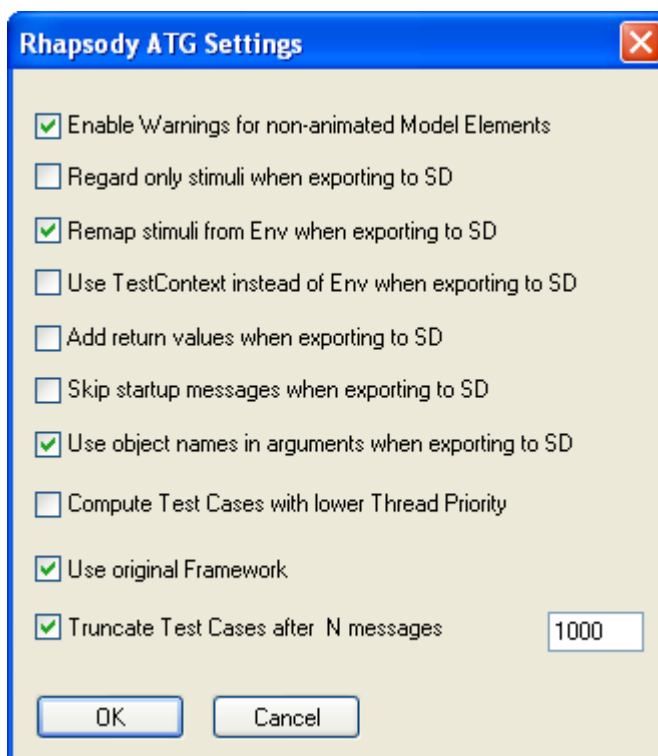
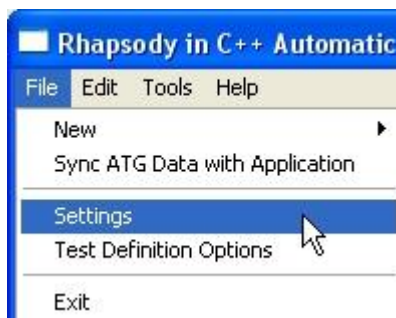
## Sync ATG Data with Application

ATG checks before every test case generation execution if the model data has been changed, and asks the user to update the ATG representation of the model data by using the synchronization function.

To synchronize the ATG representation of the model data manually right-click on the test generation component or the test generation configuration in the ATG browser and choose from context menu **Sync ATG Data with Application** or choose **File > Sync ATG Data with Application**.

## Rhapsody ATG settings

The **Rhapsody ATG Settings** dialog contains properties, which have global influence on the automatic test case generation and the export functionality of ATG. To open the **Rhapsody ATG Settings** dialog choose in the ATG main dialog from the menu **File > Settings**.



In case the option **Enable Warnings for non-animated Model elements** is enabled, ATG will generate a warning if test cases shall be generated for non-animated model elements. The reason is that ATG might not be able to generate test cases due to the missing animation information. In Rhapsody there is the possibility to uncheck the animation property of classes, operations, messages, etc. ATG needs this animation information in the generated CPP code to be able to find the model elements as coverage goal. This option is enabled by default.

In case the option **Regard only stimuli when exporting to SD** is enabled, ATG will export only the input messages when exporting test cases to Rhapsody. This option is disabled by default.

In case the option **Remap stimuli from Env when exporting to SD** is enabled, ATG will map messages that are generated as inputs by ATG to appropriate test components if possible. This option is enabled by default.

In case the option **Use TestContext instead of Env when exporting to SD** is enabled, inputs that can not be mapped to other test components will be mapped to the test context instance line instead of the ENV instance line when exporting test cases. This option is disabled by default.

In case the option **Add return values when exporting to SD** is enabled, ATG computes expected return values for operation calls of all "traced" messages defined in the "Input Definition tab", too. The values will then be exported to sequence diagrams as well. This option allows the user to control if return values are shown in sequence diagrams. This option is disabled by default.

In case the option **Skip startup messages when exporting to SD** is enabled, ATG will suppress messages which are recorded during the initialization phase of a tested system until the system under test is idle the first time. Sometimes such messages lead to stuck test execution with TestConductor due to some inconsistencies between the Rhapsody animation layer and the ATG test case generation. This option is disabled by default.

In case the option **Use object names in arguments when exporting to SD** is enabled, ATG will export object names instead of a 'don't care' value for class/pointer arguments in messages, if available. Messages can have arguments of different types. In the case that a message argument is a class type, the user can choose if exported sequence diagrams use animation names in message arguments (e.g. 'Telephone[0]'). The benefit is that TestConductor can use this animation information for test execution. If the export option is disabled, ATG will replace the animation name with an asterisk symbol (\*), which is interpreted by TestConductor as 'don't care' during test execution. This option is enabled by default.

In case the option **Compute test cases with lower thread priority** is enabled, the task priority assigned to the ATG will be reduced significant. Normal the ATG engine uses 98% of the processor power when generating test cases. To avoid this and to have the possibility to work furthermore with you computer you can enable this property. This option is disabled by default.

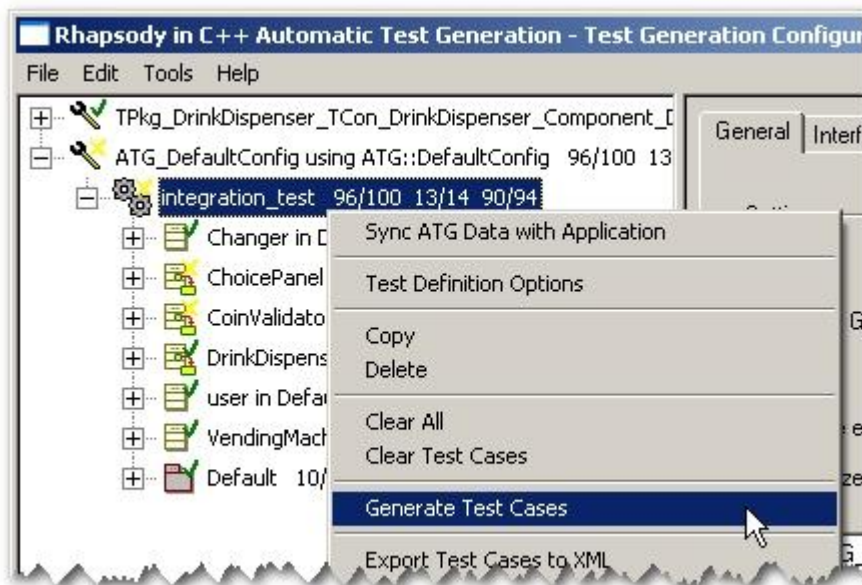
In case the option **Use original framework** is enabled, ATG uses the original Rhapsody OXF framework implementation instead of its own specific OXF framework implementation in order to generate test cases. In some cases ATG might be able to generate test cases for models that use many OXF framework functions, which might not be possible with the ATG specific OXF framework implementation. It should be

mentioned here that using the original framework often may help to overcome limitations, in particular compilation problems without usage of properties. This option is enabled by default.

In case the option **Truncate Test Cases after N messages** is enabled, all message of a trace greater than N will be truncated from the test case. Sometimes the generated test cases are very long and the writing of the test cases takes long time. With this option users can control that test cases are not generated if they have a certain length N. This option is enabled by default with N=1000.

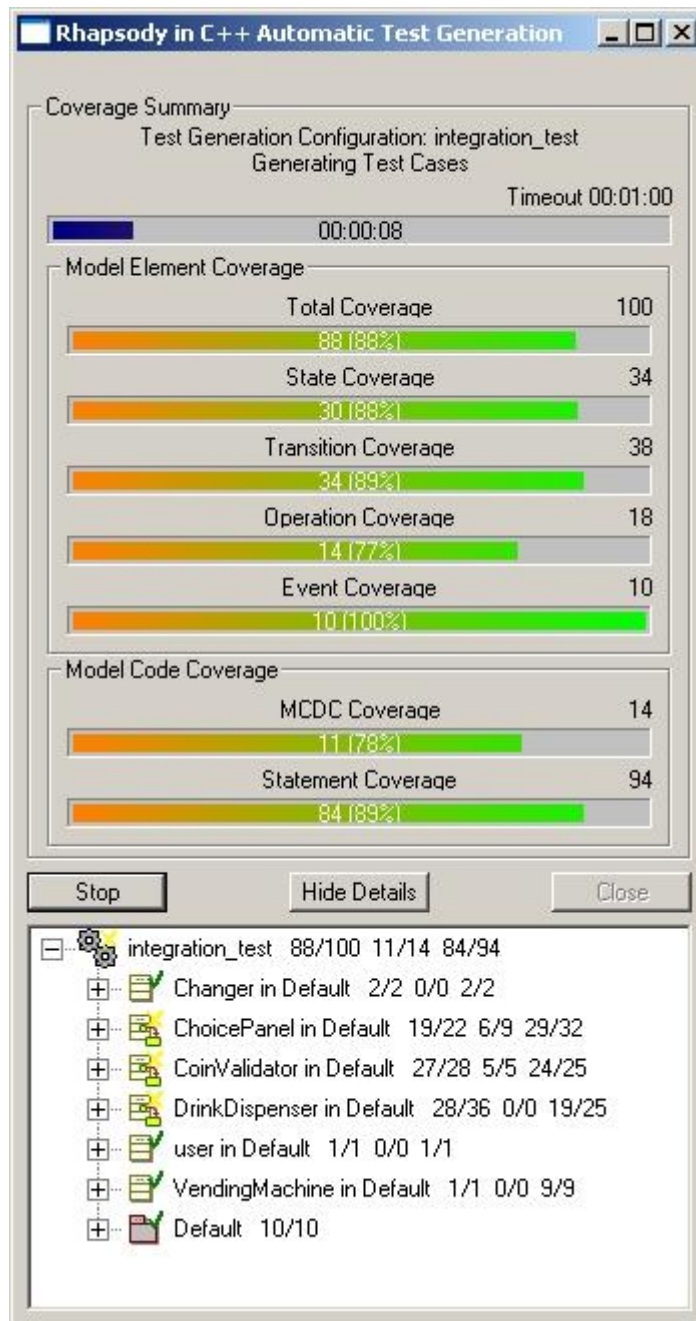
## Test Case Generation

- ◆ To generate test cases right-click on the test generation configuration in the ATG browser and choose from context menu **Generate Test Cases**.



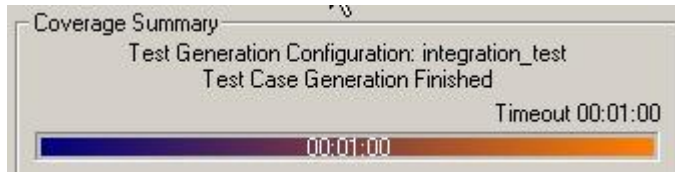
## The Rhapsody in C++ Automatic Test Generation Dialog

The Rhapsody in C++ Automatic Test Generation dialog displays the progress during test case generation.



- ◆ To stop test case generation click the button **Stop**.
- ◆ To abort test case generation and close the dialog click the button **Stop** and then the button **Close**.
- ◆ To hide the detail information section click the button **Hide Details**. To show the detail section click the button **Show Details**.

The test generation will search for test cases until all goals are covered or the specified timeout is reached. The actual status is displayed in the head section of the dialog.



While active search the head section displays **Generating Test Cases** and change this message to **Test Case Generation Finished** if one of the constraints described above is reached.

The Model Element Coverage section give notice about the Total Coverage over the sub coverage indices **State Coverage**, **Transition Coverage**, **Operation Coverage** and **Event Coverage**, which are self-describing. This section is active/visible, when the user activated the option **Model Element Coverage** under the **Coverage** tab.

The Model Code Coverage section gives notice about the **MCDC Coverage** and **Statement Coverage**. This section is active/visible, when the user activated the option **Model Code Coverage** under the **Coverage** tab.

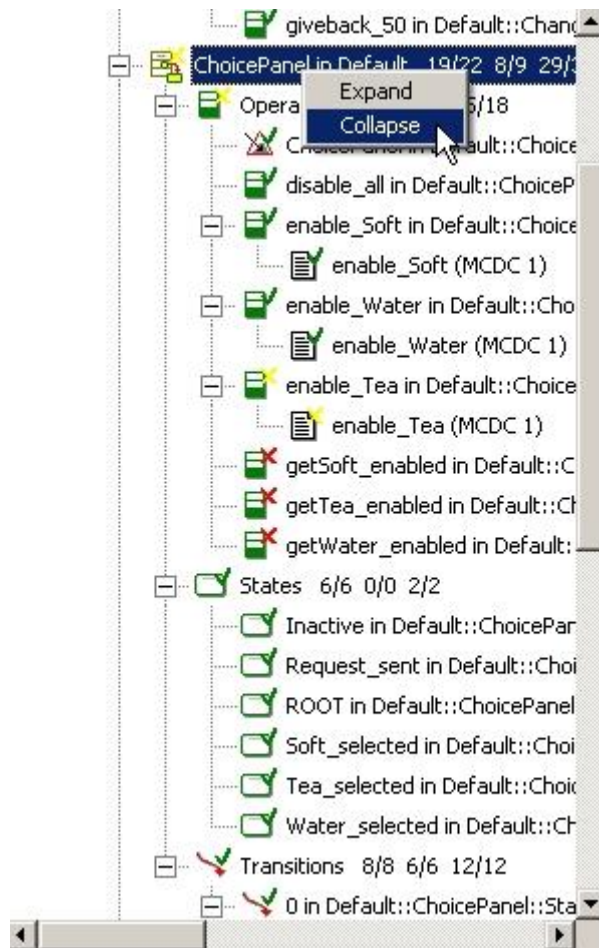
These results have to be regarded as a starting point; start ATG with different properties and you might get better results. Work with different ATG test generation configurations such that you can compare the results afterwards.

## View Customization

The ATG browser provides some mechanisms for view customization.

### One-click Expand/Collapse Hierarchical View

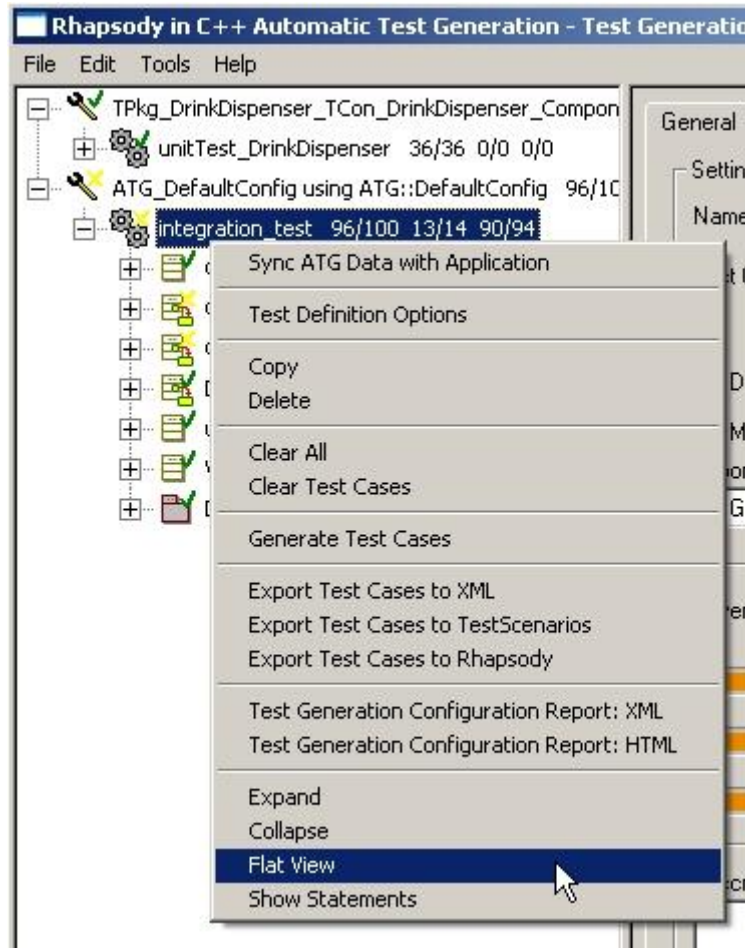
- ◆ To expand the selected item in hierarchical view right-click on the element to expand in the ATG browser and choose from context menu **Expand**.
- ◆ To collapse the selected item in hierarchical view right-click on the element to collapse in the ATG browser and choose from context menu **Collapse**



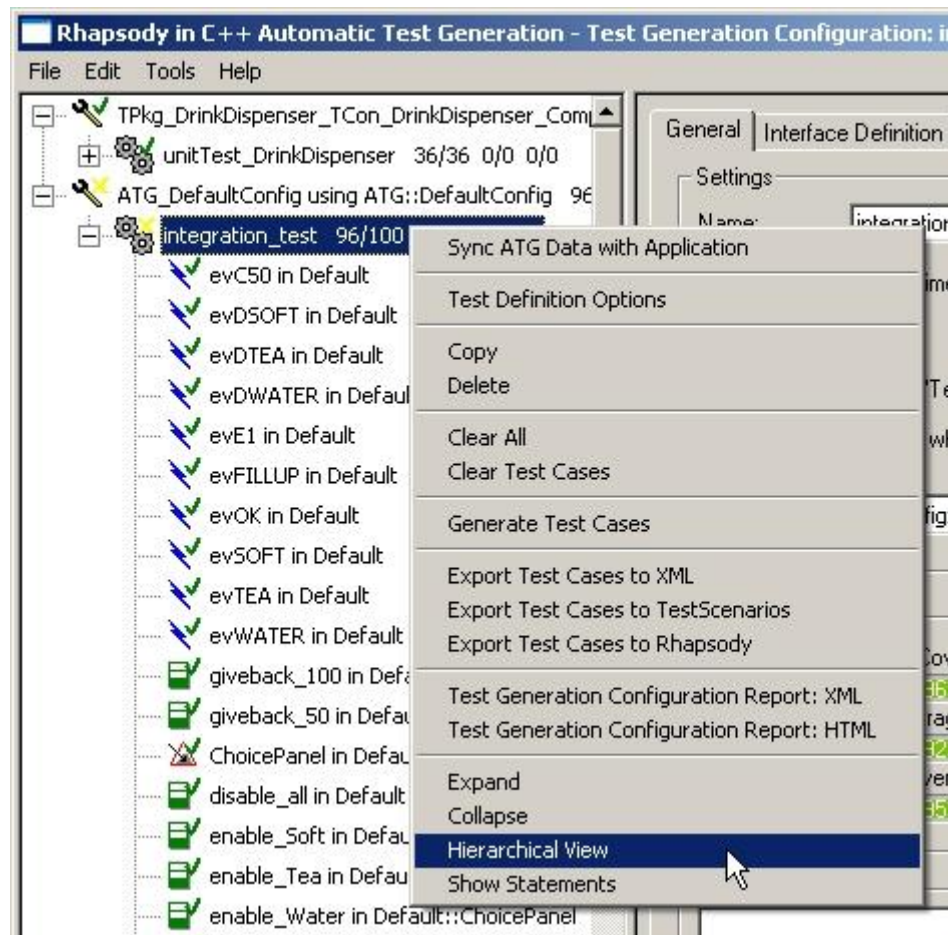
## Change between Hierarchical and Flat View

The ATG browser provides the flat and the hierarchical view.

- ◆ To switch from hierarchical view to the flat view right-click on an element in the ATG browser and choose from context menu **Flat View**.

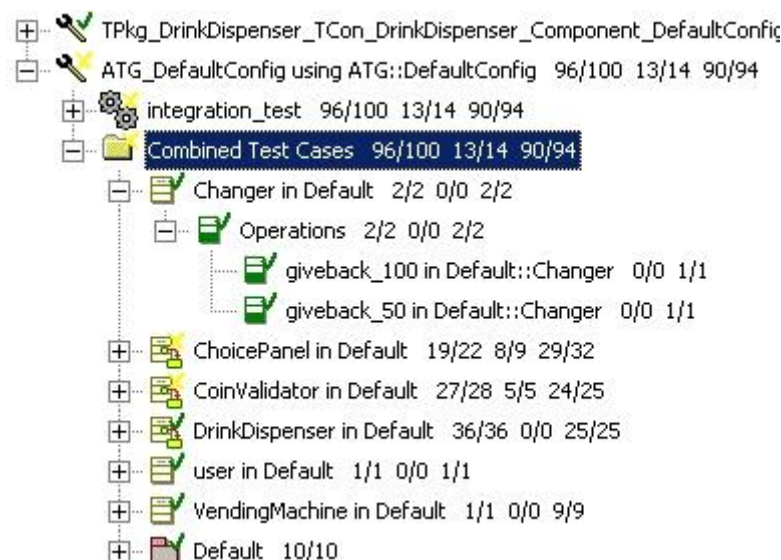
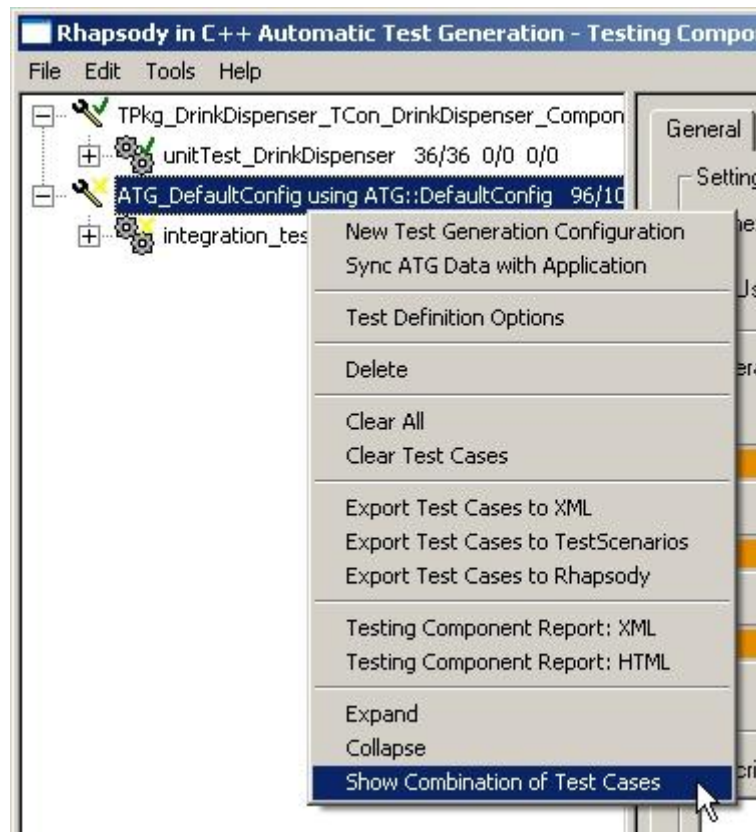


- ◆ To switch from flat view to the hierarchical view right-click on an element in the ATG browser and choose from context menu **Hierarchical View**.



## Show Combinations of Test Cases

- ◆ To show combinations of test cases right-click on a test generation component in the ATG browser and choose from context menu **Show Combination of Test Cases**.



- ◆ To hide the folder combinations of test cases right-click on the test generation component in the ATG browser and choose from context menu **Hide Combination of Test Cases**.

## Exporting Test Cases

You can export automatically generated test cases in different formats on three different levels:

- ◆ **Testing Component level**—Exports all generated test cases under a specific Testing Component
- ◆ **Test Generation Configuration level**—Exports all test cases generated for one Test Generation Configuration
- ◆ **Test goal level**—Exports selected test cases

If you select the first or second level, Rhapsody ATG minimizes the suite of test cases for the export. Redundant test cases are dropped from the suite for efficiency reasons.

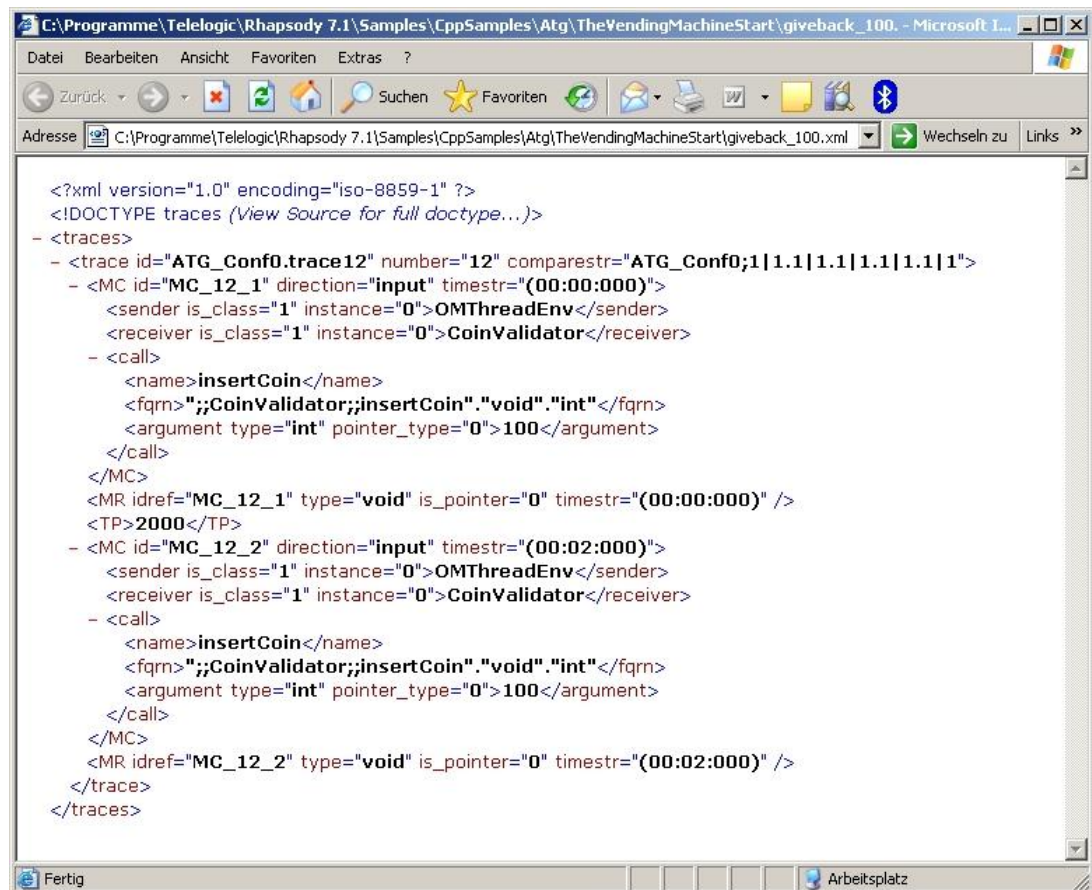
The available test case formats are:

- ◆ XML—Used to display the test case in ATG itself.
- ◆ Test Scenarios (stereotyped sequence diagrams)—Exported into a corresponding Rhapsody model.
- ◆ Rhapsody—Exports the test case directly into Rhapsody. The exported test cases can be executed using Rhapsody TestConductor.
- ◆ Exporting missing test cases to Rhapsody—Only those test cases are exported that will increase the model coverage of the test context to which the test cases are exported. If the test context already contains test cases that cover some model elements, then ATG generated test cases that will not cover new model elements will not be exported.

## Export Formats

### XML

XML is the universal format to import and export, because of its ability to convert the structured document by XSLT. ATG is able to produce XML strict. To get further information contact the Rhapsody support.



## Test Scenarios

ATG will create a test scenario in an ATG package with the corresponding structure, but without creating test cases e.g. in the test context.

## TestConductor

ATG will create a test scenario in an ATG package with the corresponding structure and corresponding test cases e.g. in the test context. Test cases and test scenarios are linked and the user is able to jump from the test case, which can be build as an executable, to the linked test scenario.

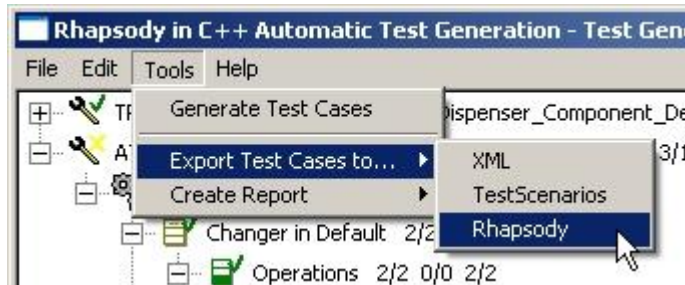
## Exporting a Single Test Case

To export a specific test case into Rhapsody, do the following:

1. In the General tab of the Test Generation Configuration dialog box for `integration_test`, set the export location to `ATG::ATG_DefaultConfig::integration_test`.
2. Open the Testing Component ATG and `integration_test` to select the test goal:

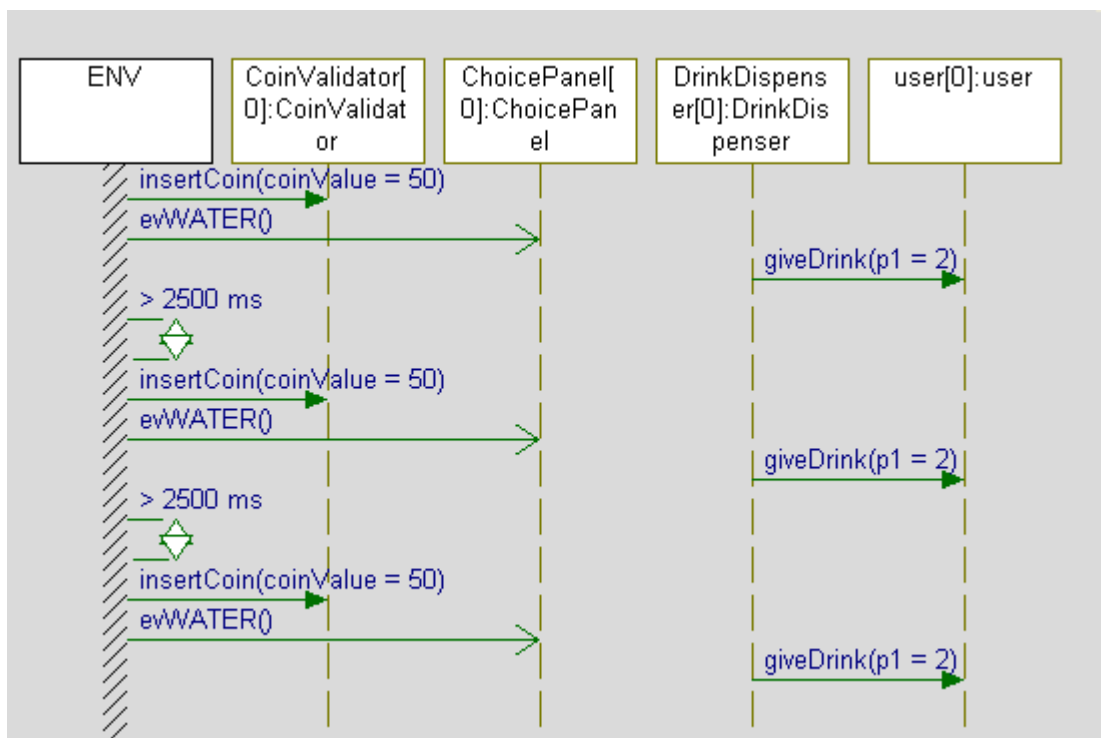
Water\_empty in  
Default::DrinkDispenser::StatechartOfDrinkDispenser.

3. Select **Tools > Export Test Cases to... > Rhapsody** to export the test case as sequence diagram to the specified location,
4. or right-click on the item Water\_empty in  
Default::DrinkDispenser::StatechartOfDrinkDispenser and choose from the context menu **Export Test Case to Rhapsody**.



**Figure 18: Exporting a Single SD into Rhapsody**

Do equivalent steps to export to XML and test scenarios.



**Figure 19: ATG-Generated test scenario test case in Rhapsody**

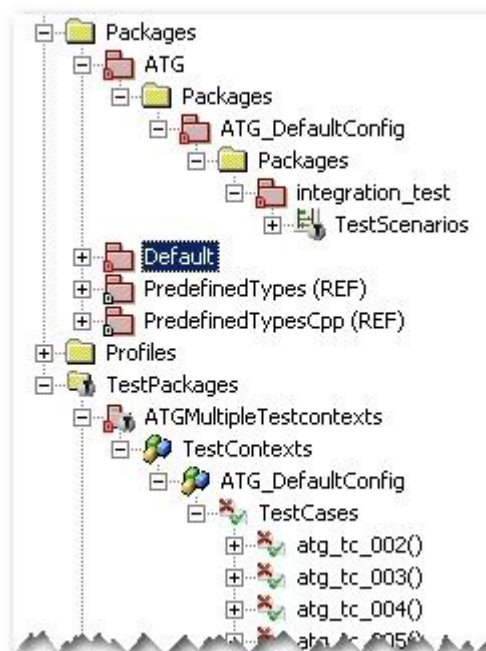
The sequence as shown in Figure 19 brings TheVendingMachine system into the state `Water_empty`, since three demands for water demands without a refill event from the environment will empty the tank.

## Exporting Test Cases on Configuration Level

To export all the generated test cases from the test generation configuration `integration_test`, do the following:

1. In the General tab of the Test Generation Configuration dialog box, set the export location to `ATG::ATG_DefaultConfig::integration_test`.
2. Select the desired Test Generation Configuration in the ATG browser.
3. Select **Tools > Export Test Cases to... > Rhapsody** to export all the generated test cases of that Test Generation Configuration into Rhapsody.
4. Alternatively, select **Tools > Export Test Cases to... > incremental Rhapsody** to export only those test cases that will increase the model coverage of the test context to which the test cases are exported. If the test context already contains test cases that cover some model elements, then ATG generated test cases that will not cover new model elements will not be exported.

Based on the structure of the Testing Component and Test Generation Configuration, Rhapsody browser contains a test folder called `ATG` that contains the same structure. The following Figure shows this folder structure.

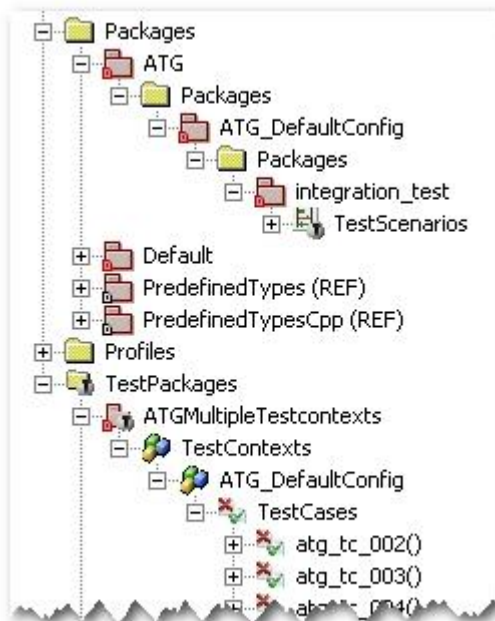


## Exporting Test Cases on Test Component Level

To export all the generated test cases from the test generation component e.g. `ATG_DefaultConfig`, do the following:

5. Select the desired Test Generation Component in the ATG browser.
6. Select **Tools > Export Test Cases to... > Rhapsody** to export all the generated test cases of that Test Generation Component into Rhapsody.
7. Alternatively, select **Tools > Export Test Cases to... > incremental Rhapsody** to export only those test cases that will increase the model coverage of the test context to which the test cases are exported. If the test context already contains test cases that cover some model elements, then ATG generated test cases that will not cover new model elements will not be exported.

Based on the structure of the Testing Component, Rhapsody browser contains a test folder called `ATG` that contains the same structure. The following Figure shows this folder structure.



## Report Generation

Two kinds of reports are available. Test Generation Configuration reports provide information about all ATG entities that are part of the selected Test Generation Configuration, e.g. the elements in the interface and the coverage goals. Reports can be viewed both as XML and also as HTML.

**Note:** Rhapsody TestConductor provides support for the reporting engine Rhapsody ReporterPLUS, which is documented in the Rhapsody TestConductor User guide.

## Test Generation Configuration Report

A *test generation configuration report* documents user settings and selections. Reports can be generated in two formats: XML and HTML. You can generate an HTML test generation configuration report for the `integration_test` Test Generation Configuration.

Select the `integration_test` configuration, then select **Tools > Create Report > HTML**.



**Figure 20: Generating a Test Definition Report**

A dialog pops up that asks you for the location where the report should be saved.



**Figure 21: Select destination folder for report**

After saving, a dialog asks if you want to open the generated report.



Figure 22: Confirm to open generated report

After choosing “yes”, the following report can be seen in your default html viewer.

The screenshot shows a web browser window titled "Rhapsody ATG Test Generation Report - Microsoft Internet Explorer". The address bar shows the path: C:\Programme\Telelogic\Rhapsody 7.1\Samples\CppSamples\Atg\TheVendingMachineStart\RhapATG. The report content is as follows:

## Rhapsody ATG Test Generation Report

**Test Generation Configuration: integration\_test**

13:01:19, Tuesday, April 24, 2007

[Used Project] [Environment Info] [Used Interface] [Coverage Definition] [Applied Abstractions] [Test Generation Summary] [List of Coverage Objectives]

Used Project	
Project:	TheVendingMachine
Component:	ATG
Configuration:	DefaultConfig
Testing Component:	ATG_DefaultConfig
Test Generation Configuration:	integration_test

Environment Info	
Executed on machine:	NBOSC30
Executed by User:	tschrief
Used OS Version:	Windows 2000 / Windows XP
Used Rhapsody Version:	7.1, build 814213
Used Rhapsody ATG Version:	3.2, build 635

Input Interface	
<b>ChoicePanel</b>	
evWATER	*
evSOFT	*
evTEA	*
<b>CoinValidator</b>	
insertCoin	*
coinValue	50,100
<b>DrinkDispenser</b>	
evFILLUP	*

Figure 23: HTML-Test Definition Report of ATG

The report includes the following information:

- ◆ The project used
- ◆ Environment information
- ◆ Inputs and Traced Instances and Messages
- ◆ A summary about test case generation
- ◆ A detailed list of the test generation goals and the generated test cases

The test cases are represented as hyperlinks. Click on a link to open the corresponding description of the automatic generated test case. This test case covers the described test goal.

## Testing Component Report

The testing component report contains the reports of all test generation configurations within the selected testing component. To generate a testing component report, select the Testing Component, then select **Tools > Create Report > HTML** to create a complete report.

## Test Execution

Using TestConductor, you can execute single test case, all test cases of a test context or of a test package, and also batch tests. For further information concerning test execution reference in the *Rhapsody TestConductor User Guide* the chapter *Test Case Execution*.

# Advanced Features

---

This section explains some advanced features of Rhapsody ATG.

## Specifying Interfaces in the Model

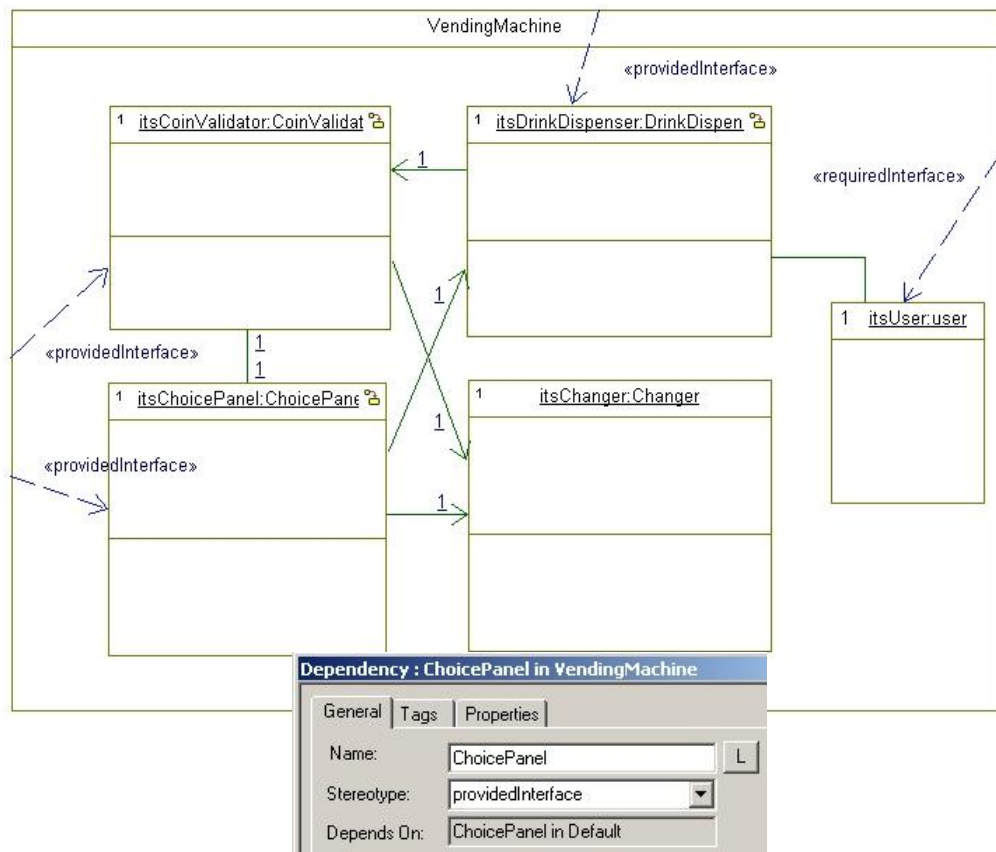
You can specify input and output interfaces for an SUT within a UML model. These are the provided interfaces (inputs to the SUT) and required interfaces (outputs of the SUT). ATG uses these interface specifications for test case generation.

### Provided Interfaces and Required Interfaces

A typical Rhapsody model contains a larger set of classes. Usually, only very few of those classes are classes that realize communication with the environment. Those classes serve to implement the interface between the model and its environment. These classes provide operations and events to the outside world, or they use the services of other required, external classes.

The `VendingMachine` contains six classes. Classes `ChoicePanel`, `CoinValidator`, and `DrinkDispenser` provide services to the environment—they provide an interface. The additional class `User` has been added to the model in order to describe the fact that external users/actors will receive a drink from the vending machine. Class `User` is not part of the SUT. If the drink is ready to be delivered, the `DrinkDispenser` calls an operation on class `User`. `User` provides this operation to the `DrinkDispenser`. The `User` class is considered to be the required interface in this sample model.

The following Figure shows how to specify interfaces in the model.



**Figure 24: Provided and Required Interfaces**

The object model diagram (OMD) shows the available classes. In addition, it shows that four stereotyped dependencies are used. Three of those are stereotyped as `<<ProvidedInterface>>`, one as `<<RequiredInterface>>`. These classes will be presented in the ATG view; however, classes `VendingMachine` and `Changer` will not be shown. ATG uses the public message of classes `ChoicePanel`, `CoinValidator`, and `DrinkDispenser` to generate test cases for the model. It traces messages to class `User` to trace reactions of the model.

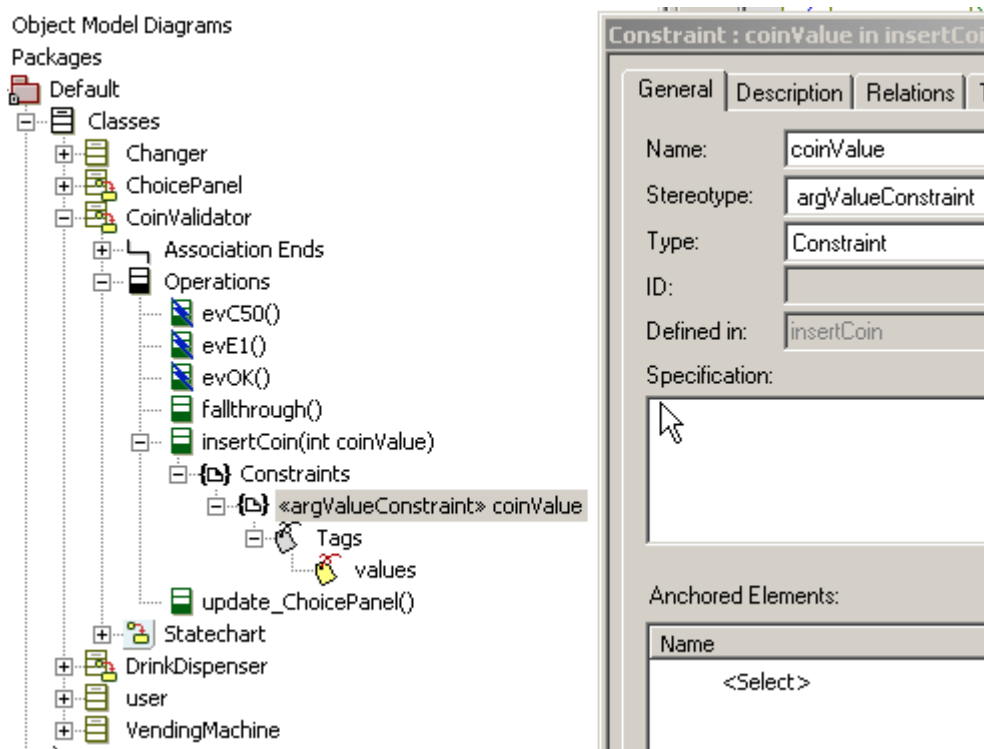
A stereotype `<<ProvidedInterface>>` can be added to `Dependency` and `generalization` arrows, while a stereotype `<<RequiredInterface>>` can be added to `Association`, `Association-End`, and `Dependency` arrows.

## Operations and Events – Argument Constraints

Operations and events of provided classes might have parameters (arguments) of predefined or user-defined types (for example integer, float, or Boolean). ATG uses type information to generate test cases (input messages together with parameter values). You can control the range of values used by ATG in order to make sure that ATG only generates test cases with valid and intended argument values. For example, you might

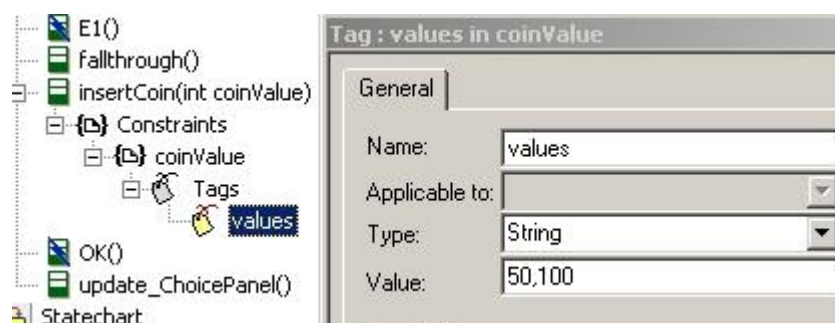
know that for a certain message `m(int p)`, parameter `p` can carry values only between 0 and 100. Those aspects can be specified in the model using *constraints*.

Consider the following Figure. Class `CoinValidator` provides an operation `insertCoin(int coinValue)`. For the argument `coinValue` assume only two values: 50 cents and 100 cents. Specify a constraint to operation `insertCoin()` with the same name as the argument (`coinValue`) and stereotype the constraint as `argValueConstraint`. This is used to enumerate the allowed values in the supported tag values of this stereotype.



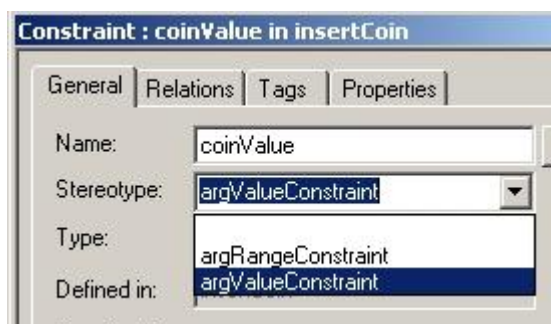
**Figure 25: Argument Constraints and Stereotypes**

The following Figure shows how to add relevant enumerated values to the tag in the dialog box. The Type of the tag is set to *String*, but users can enter integer, real, boolean, or string values in the field such that values match the type of the constraint argument. In this case the type of `coinValue` is integer and we specify to valid values 50 and 100.



**Figure 26: Enumerated Values Using Stereotype argValueConstraint**

In addition to value enumeration with constraints, ATG supports range definitions. This enables you to specify the tags `low_value` and `high_value` to specify the boundaries of an argument. The stereotype supporting it in the model is named `argRangeConstraint`, as shown in the following Figure.



**Figure 27: argValueConstraint and argRangeConstraint**

## User-Defined Constraints on Types

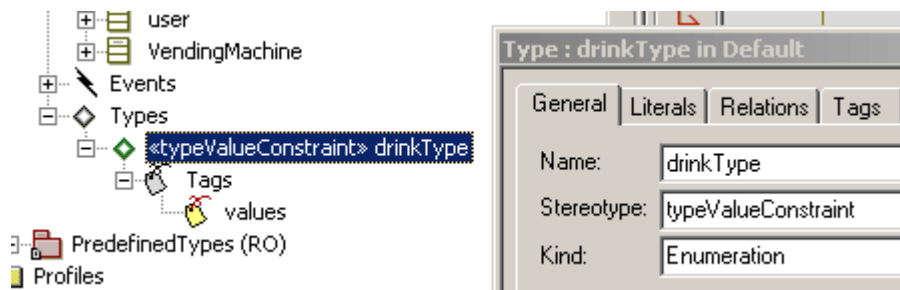
The previous section showed how to specify constraints for message arguments. Similarly, you can add constraints to types that are used to declare and define message arguments.

ATG provides stereotypes for most of the predefined Rhapsody types.



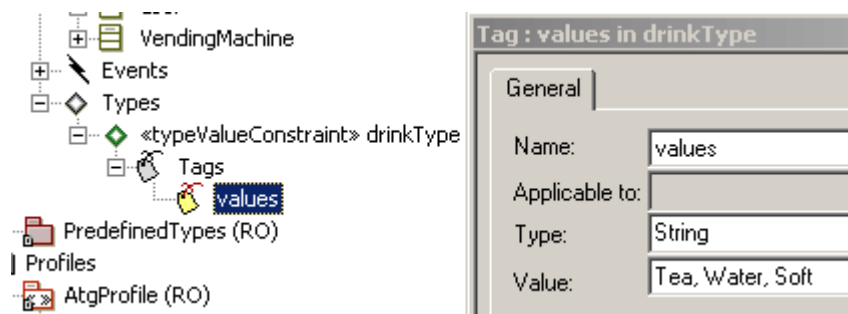
**Figure 28: Available ATG Stereotype for Types**

The following Figure shows a user-defined type, `drinkType`.



**Figure 29: User-Defined Type `drinkType` and its Stereotype**

The selected stereotype is `typeValueConstraint`. This stereotype offers a tag so you can specify a list of possible values for the type. ATG uses these values for every message argument declared using this type.



**Figure 30: Enumerated Values Using Stereotype `typeValueConstraint`**

## Working with Libraries

Software developers often use libraries when developing new software. These libraries are often only available as header files and object code, but not as C++ source code. This is not a problem for code compilation and linking, since a standard linker links object code. ATG cannot use object code for test case generation. ATG requires source code for the analysis. Still, source code is usually not available for external libraries.

ATG provides a feature that generates stub code fully automatically for user functions that are implemented in libraries. The generated stub code is used for test case generation.

Suppose, a user uses an external library *change\_money.lib* in TheVendingMachine sample model of this user guide. Some parts of the header file of that library is shown in Figure 50 below.

```
// Header file for Library change_money.lib

#ifndef CHANGE_MONEY_H
#define CHANGE_MONEY_H

enum changeMoneyType {
    fifty = 1,
    hundred = 2
};

// p==1 returns 10; p==2 returns 100
int get_amount(int p);

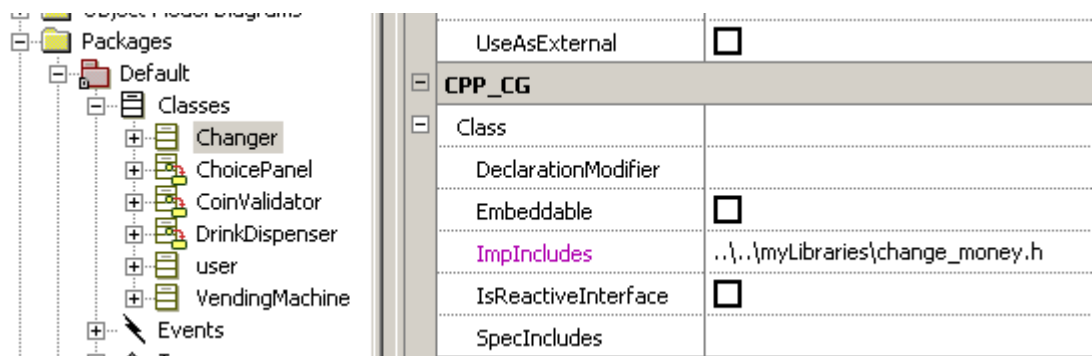
#endif
```

**Figure 31: Header File of Library Change\_Money.lib**

This library provides a function *int get\_amount(int p)* that returns an integer value depending on an integer argument *p*. The behaviour of the functions is not visible to ATG, because it is implemented in the library.

This function *int get\_amount(int p)* is used for the implementation of functions *giveback\_50()* and *giveback\_100()* of class *Changer*.

Figure 32 below shows that class *Changer* includes *change\_money.h*.



**Figure 32: Change\_money.h is Included by Class Changer**

Figure 33 below shows that library function *get\_amount(1)* is called in user function *giveback\_50()* of class *Changer*.

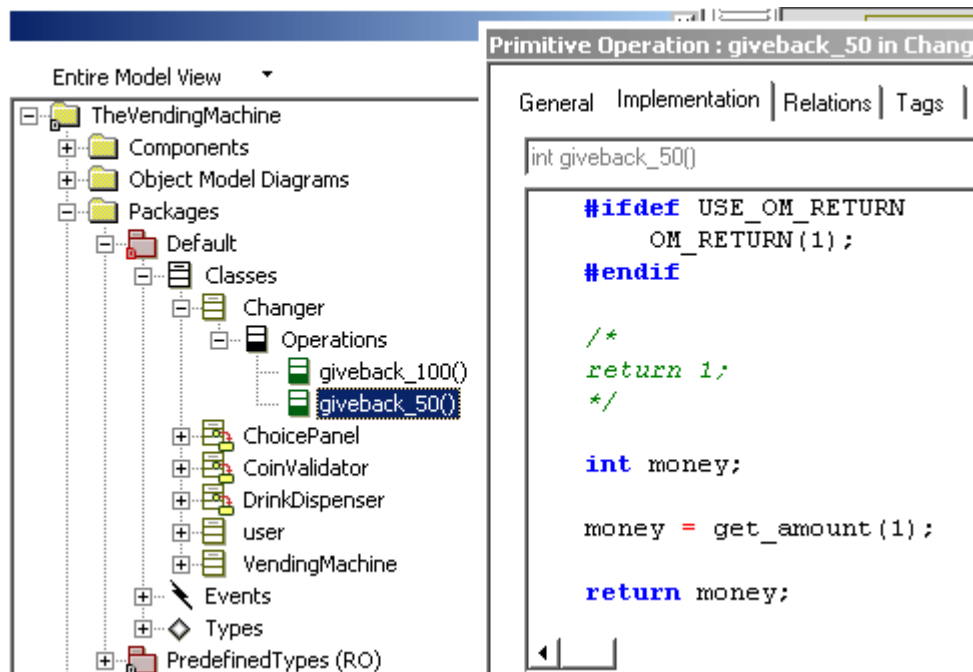


Figure 33: `get_amount(1)` is called in `giveback_50()` of class `Changer`

Open the Implementation tab and change the implementation such that the code looks like in Figure 33. Do the same in user function `giveback_100()`.

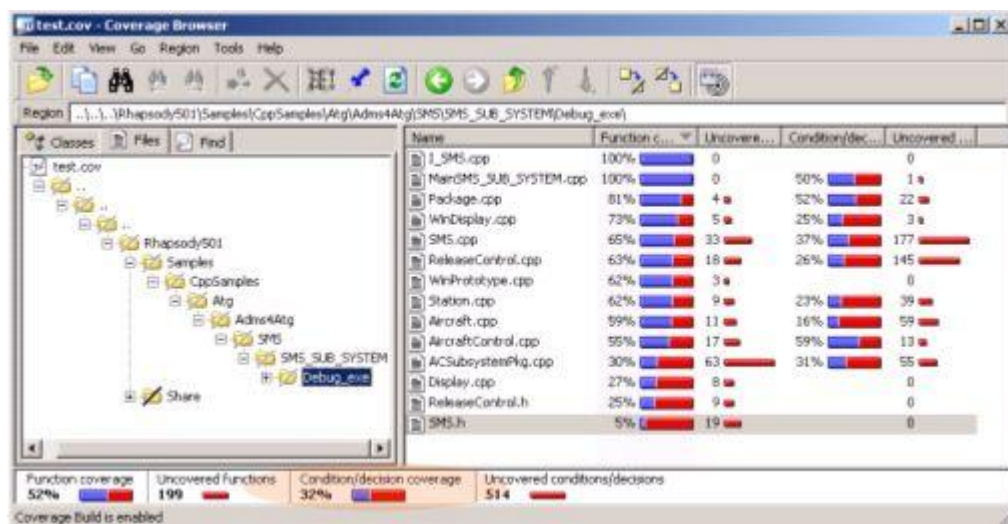
The user can launch test case generation as usual via **Tools > Generate Test Cases**. ATG will find the header file and will analyze the signature of the function. Since for the selected scope there is no implementation of this function, ATG automatically “stubs” this function, i.e., ATG generates an implementation for this function automatically.

# Coverage Measurement with Third-Party Tools

Actual code coverage measurement is an important activity in order to assess the quality of the test cases. Third-party tools support these activities when test case execution is performed on real production code.

Several third-party code coverage measurement tools exist on the market that you can use to figure out the total code coverage achieved by executing all the ATG generated test cases.

You can place these tools into the code generation and compilation process. These tools do have all information about the C++ code and its objects. In addition, since TestConductor drives the executable with the test cases generated by ATG, code coverage (such as MC/DC coverage) can be measured. Some examples of code coverage measurement tools are the tools from Bullseye, and IBM® Rational® Test RealTime.



**Figure 34: Code Coverage Measurement Tool**

**During TestConductor Execution of Test Cases**

# Appendix

---

## Restrictions

The restrictions and limitation of ATG are described in an additional document [rhap\\_atg\\_limitations.pdf](#)

## Frequently Asked Questions

Currently known FAQs are described in the addition document [rhap\\_atg\\_FAQs.pdf](#)