securityandtrust.lu

Use Case Modeling and Test Cases Generator (UMTG)
User Manual v1.0

Software Verification and Validation Laboratory
Interdisciplinary Centre for Security, Reliability and Trust
University of Luxembourg

June 9, 2015

UNIVERSITÉ DU LUXEMBOURG

Contents

1 Introduction

UMTG (Use Case Modeller and Test cases generator) is a tool for the automatic generation of system test cases from use case specification written according to RUCM. UMTG is a toolset that supports automated test generation and implements the methodology presented at the *2015 International Symposium on Software Testing and Analysis (ISSTA'15)* [**?**] (a version of the ISSTA'15 paper is enclosed in the appendix). UMTG comprises two plugins, one for Eclipse [**?**] and one for Telelogic/IBM Doors [**?**].

This document overviews the methodology behind UMTG, details the main functionality of UMTG and provides the installation instructions for UMTG. This document proceeds as follow. Section 2 provides an overview of the UMTG methodology, Section 3 details the activities required to install and setup UMTG, Section 4 describes the features implemented in the plugin for DOORS, Section 5 describes the features implemented in the plugin for Eclipse, Section **??** provides a tutorial based on BodySenseIII.

2 UMTG

We designed a modeling methodology based on three main components: use case specifications written in RUCM format to express requirements [**?**]; domain models to capture the domain and support the identification of test inputs; OCL constraints defined on the domain model to refine use case constraints and post-conditions.

Figure 1 overviews the methodology. The methodology comprises activities performed by the software engineers and activities that are automated by a toolset. The software engineer elicits requirements with RUCM (Step 1). RUCM is a use case specification format that provides restriction rules and specific keywords constraining the use of natural language in use cases. The domain model is manually created as a UML class diagram (Step 2). Automated tools are used to check if the domain model includes all the entities mentioned in the use cases (Step 3). NLP is used to extract domain entities from the use cases. Missing entities are shown to the software engineer who refines the domain model (Step 4). Steps 3 and 4 are iterative: the domain model is refined until it is complete.

Once the domain model is completed, textual descriptions of pre, post and guard conditions in the use cases are automatically extracted (Step 5) to be reformulated as OCL constraints by engineers (Step 6). An automated toolset further processes the use cases with the OCL constraints to generate a Use Case Test Model for each use case (Step 7). A Use Case Test Model is a directed graph that explicitly captures the implicit behavioural information in the corresponding use case.

The methodology relies on constraint solving for OCL constraints that are attached to the nodes of the test models. The goal is to generate test inputs associated with use case scenarios (Step 8). We use the term use case scenario for a sequence of use case steps that starts with a use case precondition and ends with a postcondition of either a basic or alternative flow. Test inputs cover all paths in the testing model and therefore all possible use case scenarios.

The software engineer provides a mapping table that maps high-level operation descriptions and test inputs on the concrete driver functions and inputs that should be executed by the test case (Step 9). Executable test cases are automatically generated through the mapping table (Step 10). The mapping table can be improved in an iterative fashion after observing the test cases generated by the toolset. If the test infrastructure and hardware drivers change in the course of the system lifespan, then only the mapping table needs to change.

Figure 2 shows the mapping between the steps of the UMTG methodology and the activities performed by software engineers by using the UMTG plug-ins. Steps in italic indicates activities performed by software engineers by using editors provided either by Eclipse, Doors, or Rhapsody. The names of the other steps correspond with the names of the UMTG menu items that provide a given functionality (each functionality is described in the coming sections). Each step in Figure 2 has a number that recalls the corresponding step of the methodology in Figure 1.

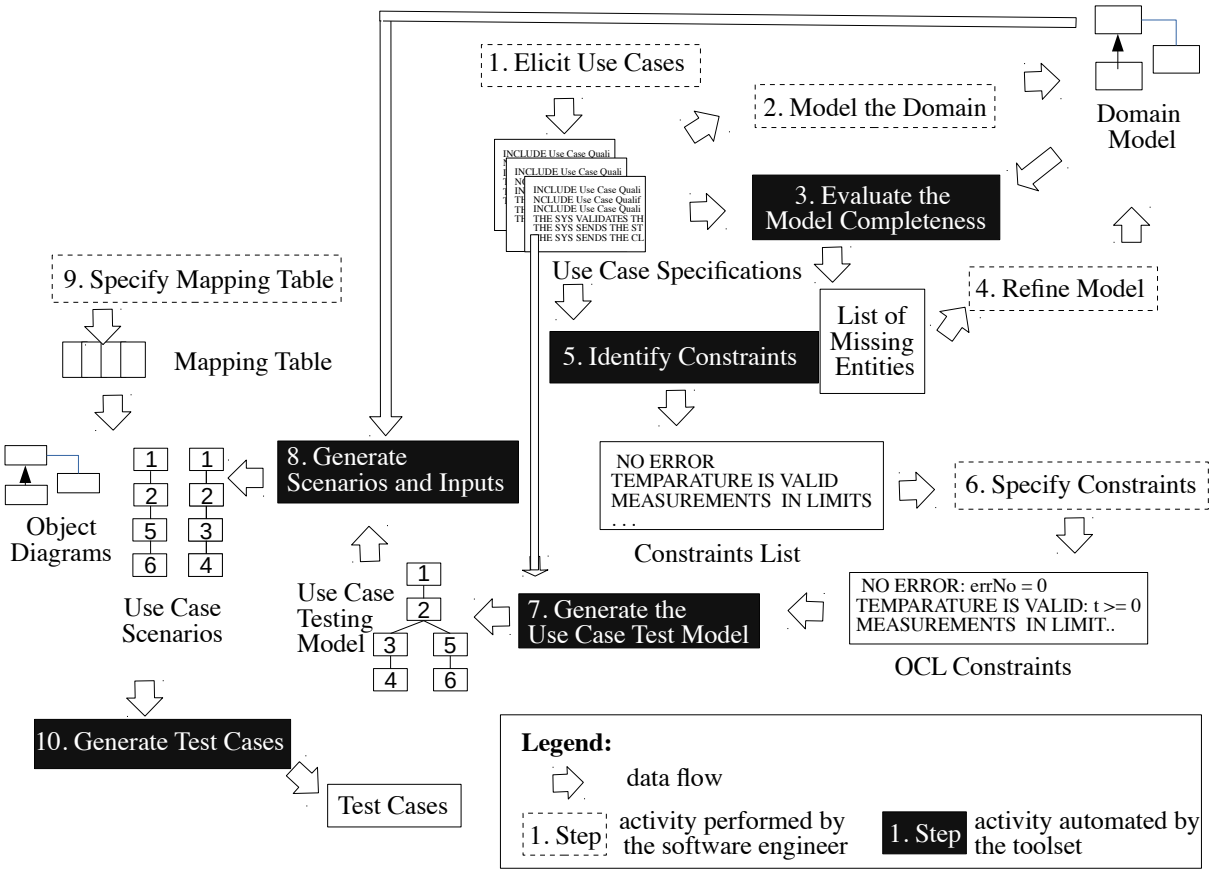


Figure 1: A Methodology for the Automatic Generation of Test Cases

Figure 2: Mapping between UMTG methodology and the features implemented by UMTG plug-ins

3 UMTG Setup

3.1 UMTG distribution

UMTG is distributed in a folder that contains different sub-folder with all the artefacts required for both running UMTG and replicate the Tutorial enclosed in this document. Figure 3 shows the UMTG Distribution folder and its content.

To simply try UMTG for a workshop or tutorial we suggest to copy the folder *"UMTG_distribution"* into your local drive, for example on your desktop as *"Desktop\UMTG_distribution"*.

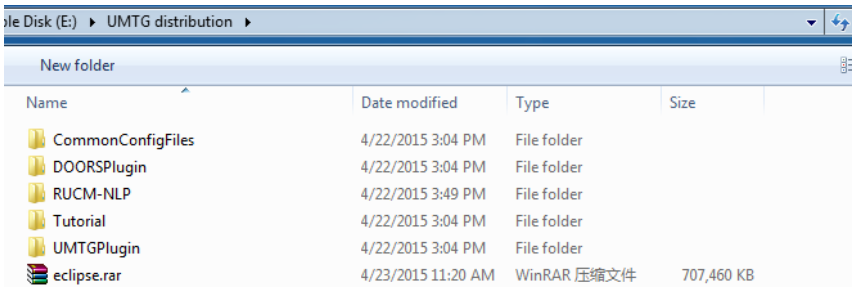


Figure 3: UMTG distribution folder

3.2 Prerequisite: Java 7

Please verify the installed java version using the "java –version" command (see Figure 4). If Java 7 is not listed in the result, please click Java SE Runtime Environment 7 to install java 7.

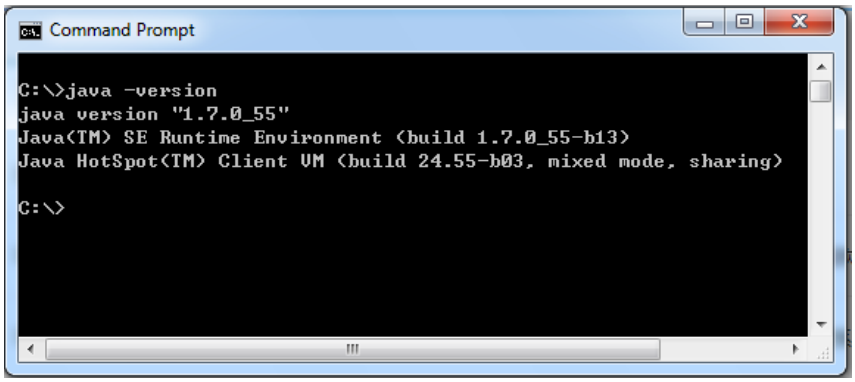


Figure 4: verify java version

3.3 Prerequisite: XMI4Rhapsody plugin

XMI4Rhapsody plugin usually installed under *%User Folder% \IBM\Rational\Rhapsody\x.x.x\Sodius\XM*
Please re-install Rhapsody by using the bath file (uninstallation is not required) in case of no installation.

NOTICE: If Rhapsody installed using Administrator account, please find the XMI4Rhapsody plugin under the user folder of Administrator account.

3.4 Copy and Edit the Common Config Files

Before installing plug-ins, all the dependent libraries and configuration files, which under the *"CommonConfigFiles Folder"*, need to be firstly placed to the User folder (for example: c:\Users \John\).

Copy the *"SVVTestGenerator"* folder under *"CommonConfigFiles"* into the User folder (see Figure 5).

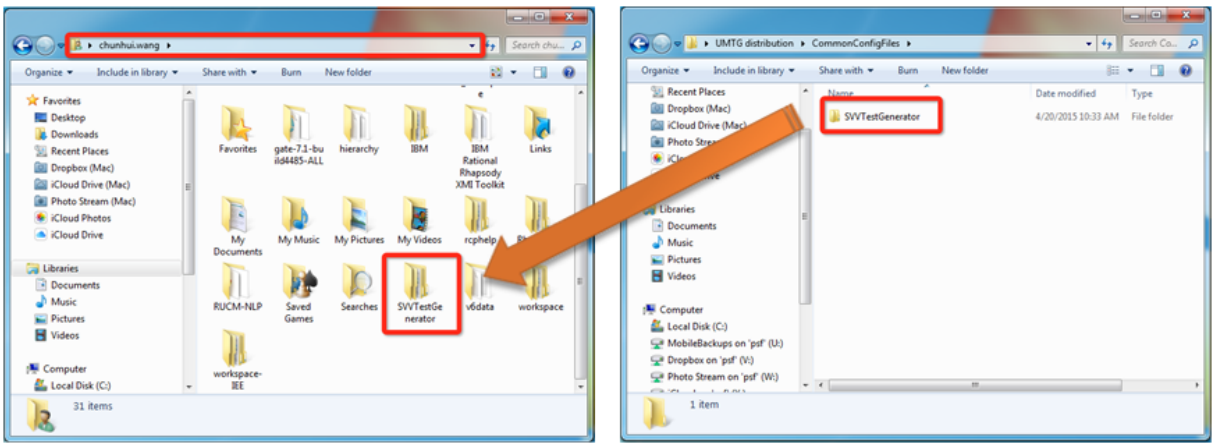


Figure 5: Copying the common config files

Secondly, there are two configuration files (*"doors.config.properties"* and *"umtg.config.properties"*) need to be edited.

Open the file *"doors.config.properties"* under the *"DoorsPlugin"* folder and edit the file by following the instructions provided in the comment lines (see Figure 6).

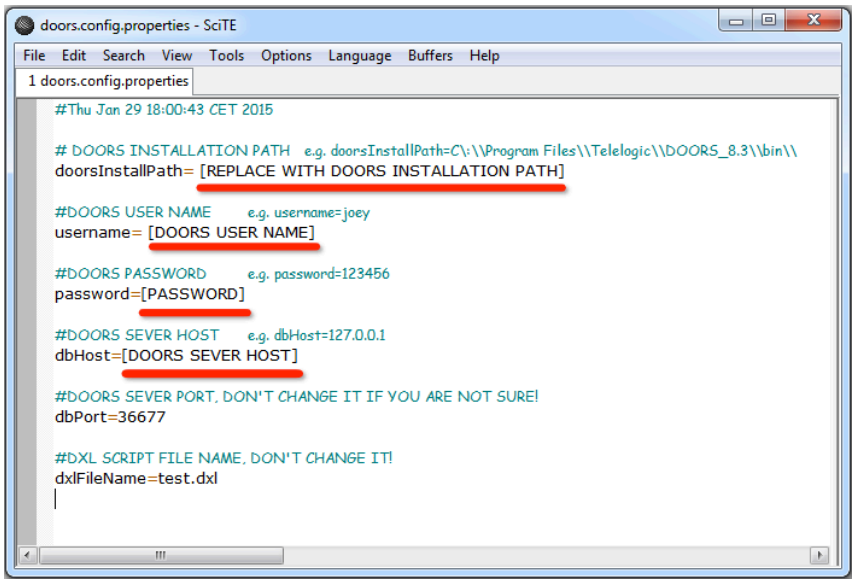


Figure 6: Edit DOORS properties file

Open the file *"umtg.config.properties"* under the *"UMTGPlugin"* folder and edit the file by following the instructions provided in the comment lines (see Figure 7).

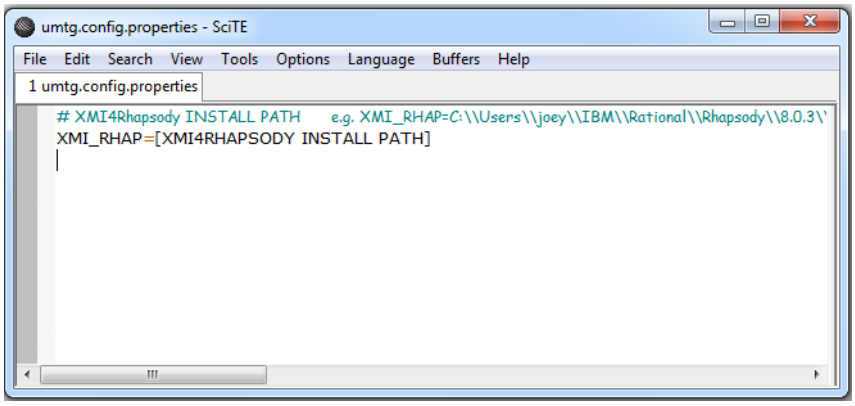


Figure 7: Edit UMTG properties file

3.5 Installing the Doors plug-in

This section describes how to install the DOORS plug-in. The installation proceeds by copying the plug-in into a folder that we name *"Addins Folder"*.

The folder *"Addins Folder"* is the folder that contains all the addins for DOORS written by using dxl scripts. The *"Addins Folder"* is located under "***%DOORSInstallFolder%***\lib\dxl\addins".

The keyword ***%DOORSInstallFolder%*** indicates the installation location of DOORS (for example: C:\Program Files\Telelogic\DOORS_8.3\).

Following paragraphs list the steps to perform to copy and setup the UMTG plug-in for DOORS.

1) Unzip (a third party Unzip tool like 7zip) the file *gate-7.1xxxx.zip* into your User folder.

2) Copy the folder *"NLP"* under *"/DOORSPlugin"* into ***Addins Folder*** (see Figure 8).

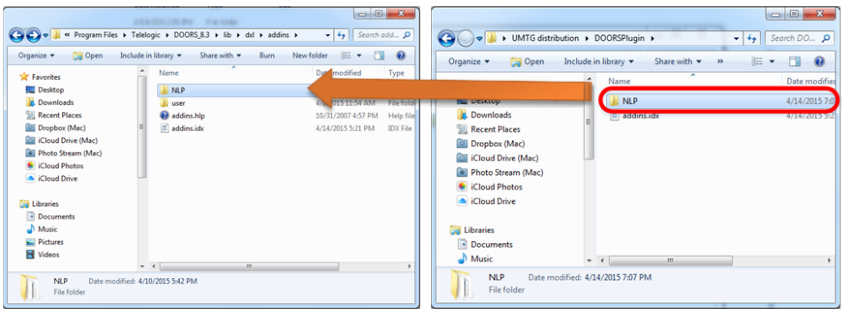


Figure 8: Copying folder *"NLP"*

3) Edit the *addins.idx* file under *"Addins Folder"* with an Administrator privilege (or simply copy the file *addins.idx* to your desktop, edit it, and then copy it back to *"Addins Folder"*).

   Add "`NLP G _ UMTG`" as a new line to the end and save.

4) Open a Formal Module in DOORS, If the installation succeeded, the UMTG menu will appear in the top menu bar. Figure 9 shows the UMTG menu.

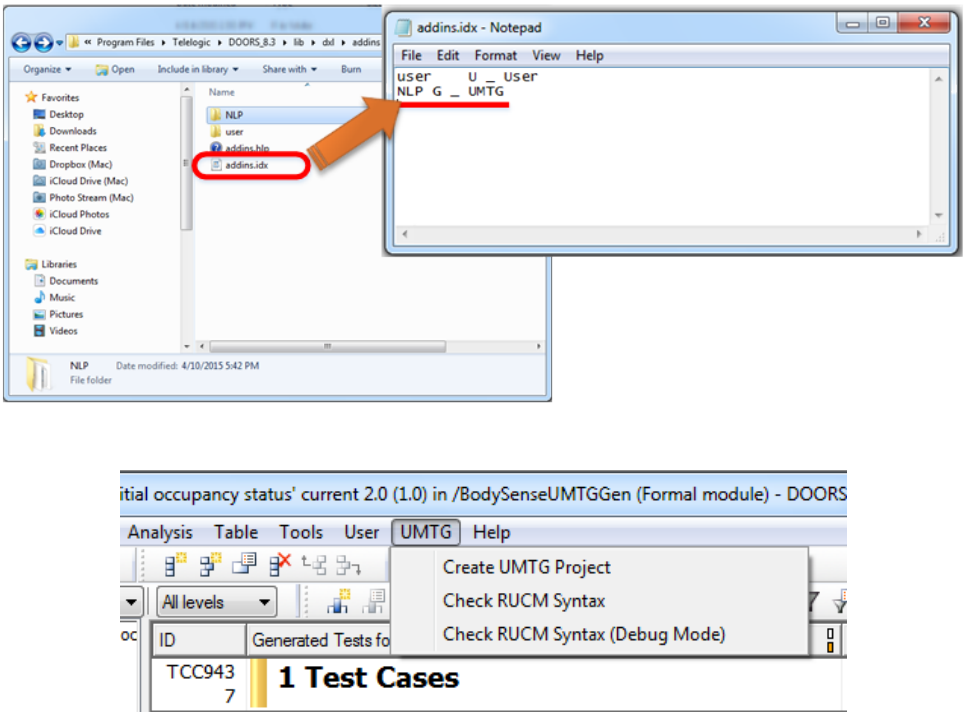


Figure 9: UMTG menu for DOORS

3.6 Installing Eclipse with plug-ins

This sections describes how to install the UMTG plug-in for Eclipse. The installation of UMTG can be performed either by copying a dedicated Eclipse version into the system (suggested choice) or by installing the plug-in into an existing Eclipse version. Following Sections detail the activities required for both the cases.

3.6.1 Install a dedicated Eclipse version

1) Unzip the Eclipse distribution into a local folder (for example under *"Desktop \UMTG_distribution \eclipse"*.

2) Start Eclipse by clicking on eclipse.exe (for example under *"Desktop\UMTG distribution\eclipse"*).

3) Install the Rhapsody Eclipse plug-in (see Figure 10)

   Figure 10 provides screenshots of the menu items to click in order to perform this activity. To start the installation open Eclipse and go to *Help -> Install New Software...*, then perform the following activities:

   ① Add a new repository by clicking the button "add".

   ② Type a meaningful name for the new repository, e.g. "Rhapsody"

   ③ Click the button "Local..." and locate to the "Eclipse" folder under the Rhapsody Install Path.

   For example: C:\Program Files\IBM\Rational\Rhapsody\8.0.3\Eclipse

   ④ Select the first item "IBM(R) Rational(R) Rhapsody(R) Platform Integration" in the new added repository and make sure other items are unselected.

Then click *Next* and *Finished* to start the installation (Eclipse needs to be restarted to make the changes to take effect).

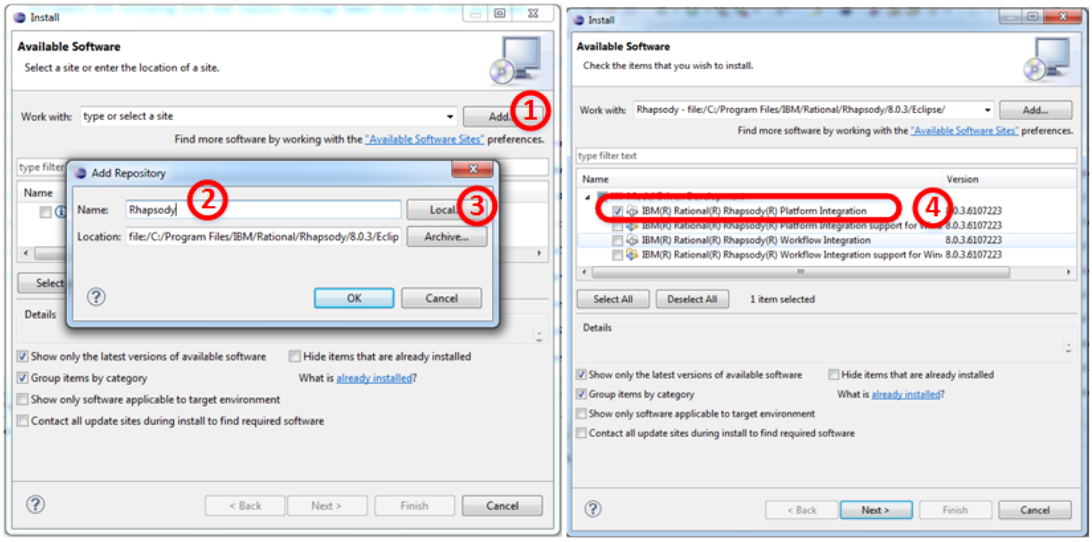


Figure 10: Install the Rhapsody Eclipse plug-in

4) Verify the installation (see Figure 11)

① After restarting Eclipse go to *Window -> Open Perspective -> Other...*

② If the installation succeeded a new perspective named "Rhapsody Modeling" will appear in the popup window.

③ Select "Rhapsody Modeling" and click ok to open that perspective.

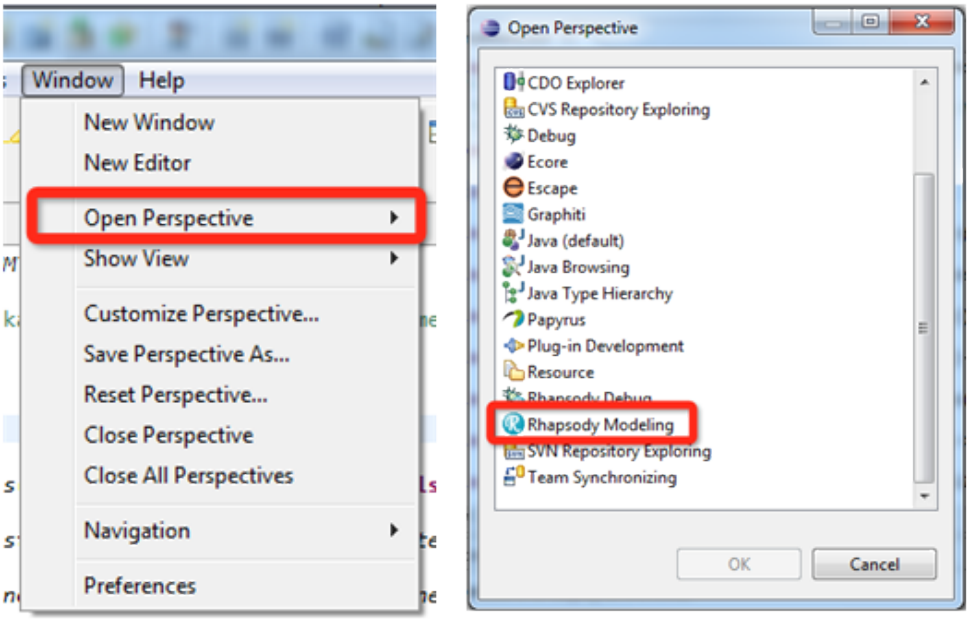


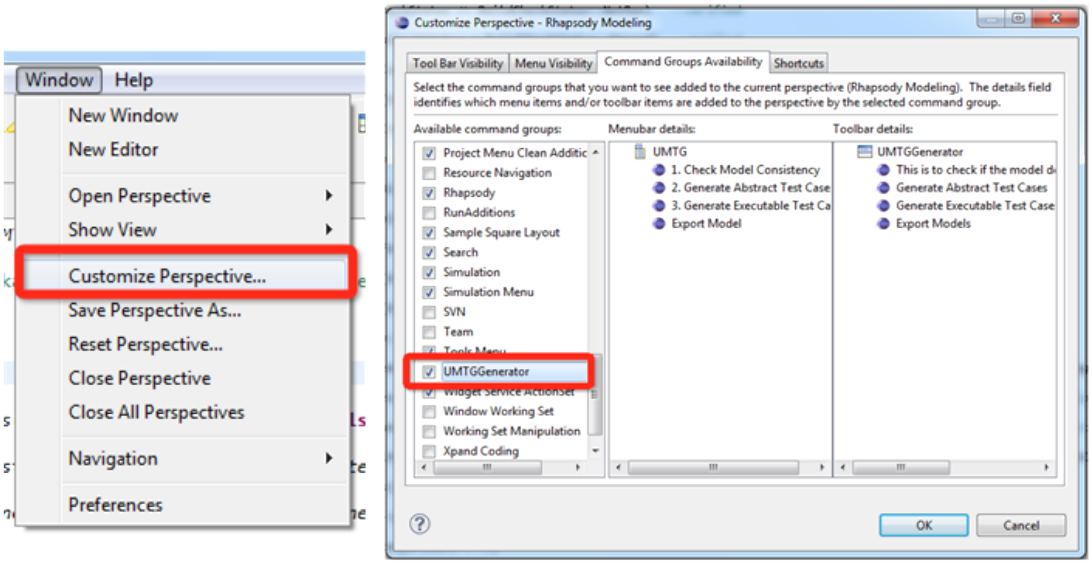
Figure 11: Verify the UMTG installation on Eclipse

5) Enable the UMTG menu

① Under "Rhapsody Modeling" perspective, go to Window -> Customize Perspective...

② Switch to "Command Groups Availability" Tab in the popup window.

③ Select "UMTGGenerator" and Click OK.



3.6.2 Install the UMTG plug-in into an existing Eclipse distribution

This Section describes how to install the UMTG plug-in into an existing Eclipse distribution. Follow these instructions only if you did not already installed a dedicated Eclipse distribution.

1) Install the UMTG dependencies and plug-ins (see Figure 12)
   ① Open eclipse, go to Help -> Install New Software...
   ② From the drop list select "Indigo - http://download.eclipse.org/releases/indigo"
   ③ After the repository loaded, select the Modeling package to install.
   ④ Restart Eclipse to make the changes to take effect.

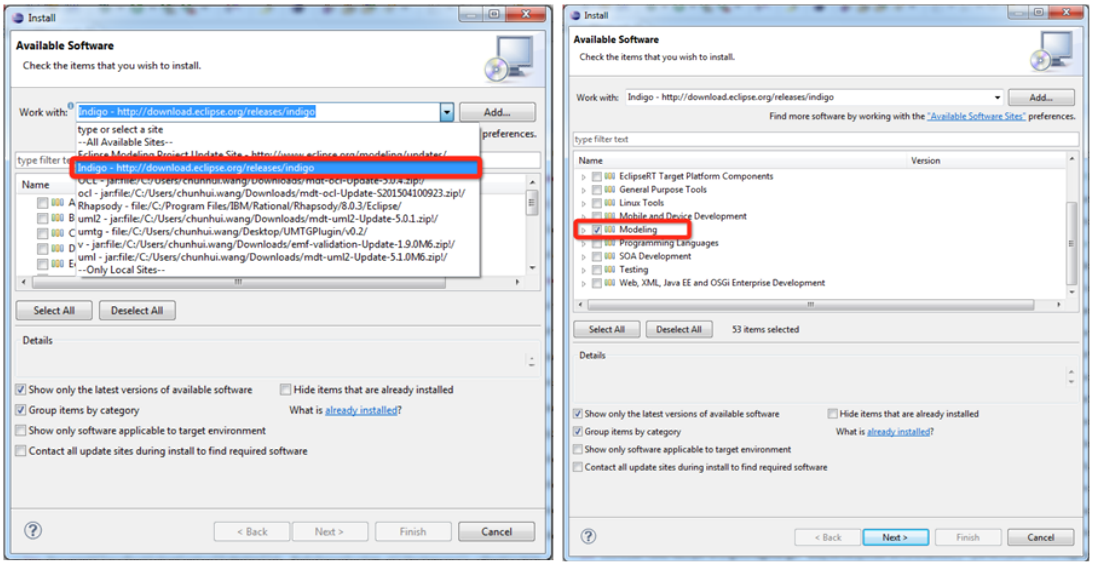


Figure 12: Install UMTG dependencies and plug-ins

2) Install the UMTG plug-in (see Figure 13)

This is done by opening Eclipse and going to *Help -> Install New Software...*, and then:

① Add a new repository by click add button.

② Type in a meaningful name for the new repository, e.g. "UMTG"

③ Click the button *"Local..."* and locate to the "UMTGPlugin" folder under the UMTG distribution folder. Click the button *OK* to close the window.

④ Select the item "snt.svv.umtg.testgenerator" in the new added repository.

Then click the buttons *"Next"* and *"Finished"* to start the installation (Eclipse needs to be restarted to make the changes to take effect).

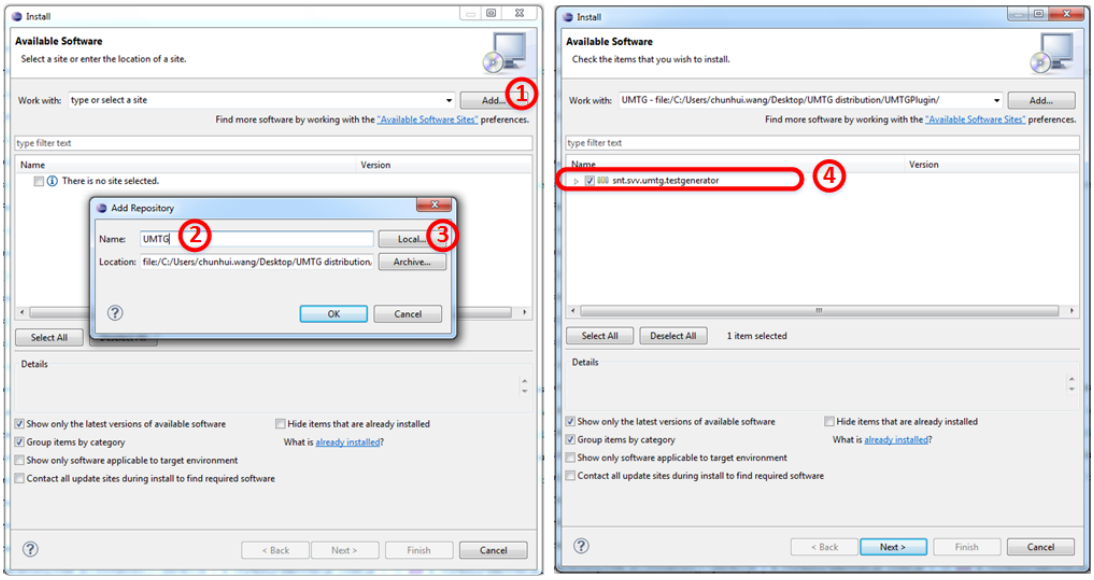


Figure 13: Installing the UMTG plug-in into an existing Eclipse

3) Enable the UMTG menu

Please refer to the 4th step in Section 3.6.1.

4 DOORS Plug-in

The UMTG DOORS plug-in activates a new menu in the Doors interface called *UMTG*. The menu *UMTG* provides three functions, *Check RUCM Syntax*, *Check RUCM Syntax (Debug Mode)*, *Create UMTG Project* that are described in the following.

4.1 Check RUCM Syntax

The function *Check RUCM Syntax* enables the software engineers to check if the use case specification enlisted in the current Doors module follow the RUCM template. In case of errors in the syntax of the use cases the function will list the sentences which violate the RUCM template. These sentences are marked as being *unknown* sentences.

Figure 14 shows function *Check RUCM Syntax*. Label (1) points to the menu item that activates the function. Label (2) indicates the command window that starts the external tool used for checking (this window will be automatically closed along with the closing of the page with the list of errors). Label (3) shows the window with the list of errors.

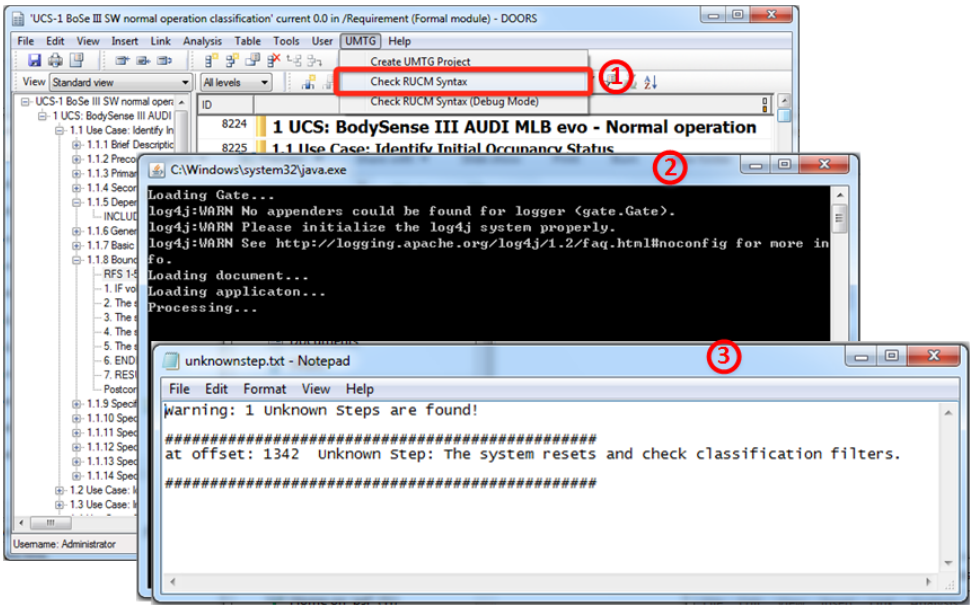


Figure 14: Function *Check RUCM Syntax*

4.2 Check RUCM Syntax (Debug Mode)

The function *Check RUCM Syntax* simply lists all the sentences that do not match the RUCM format, and indicates that they are not well-written, but do not give any additional information about the errors made by the software analyst.

To retrieve additional information about the errors made when writing RUCM test cases, the software engineer must use the function *Check RUCM Syntax (Debug Mode)*. This function starts the user interface of GATE, a open source Natural Language Processing (NLP) library, to let the software engineer collect more information about the error made when writing the use cases. GATE is the underlying library used by UMTG to process use cases and extract information.

Figure 15 shows the output of the function *Check RUCM Syntax (Debug Mode)*.

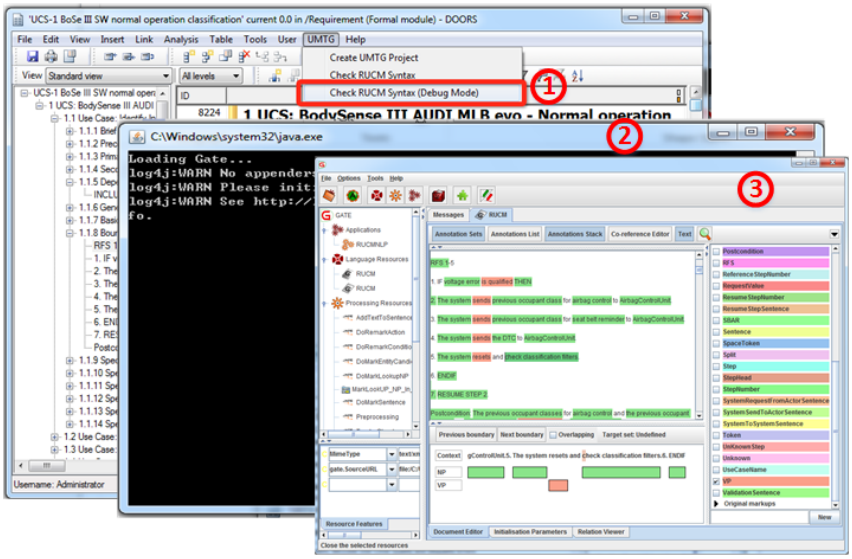


Figure 15: Function *Check RUCM Syntax (Debug Mode)*

4.3 Create UMTG Project

Function *Create UMTG Project* is used once the RUCM syntax checks passed. Function *Create UMTG Project* lets the user to create a UMTG Project, which is a kind of Eclipse project that is used to exchange information between DOORS and Eclipse/Rhapsody. The UMTG project created by DOORS will be imported into Eclipse to complete the test case generation from the Eclipse/Rhapsody user interface.

Figure 16 shows the output of function *Create UMTG Project*.

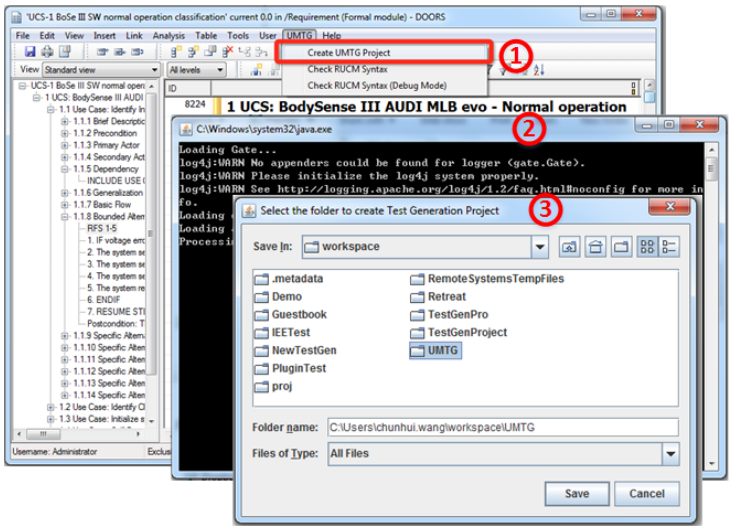


Figure 16: Function *Create UMTG Project*

5 Eclipse plug-in

The UMTG Eclipse plug-in activates a menu that provides four features: *Export Domain Model*, *Check Model Consistency*, *Generate Abstract Test Cases*,*Generate Executable Test Cases*.

Following sections describe the four main functionality provided by the UMTG Eclipse plug-in, plus two prerequisites for the proper execution of the plug-in, i.e. Import a UMTG project, and Open Rhapsody project.

5.1 Prerequisite: Import UMTG project

Importing a UMTG project created by using the UMTG Doors plug-in is required in order to generate test cases by using the UMTG Eclipse plug-in. Project import is done by means of the import functionality provided by Eclipse.

5.2 Prerequisite: Open Rhapsody project

A Rhapsody project needs to be opened before using the UMTG plug-in for Eclipse.

5.3 Export Model

The function *Export Model* exports the domain model created by Rhapsody into the XMI format. To this end the plug-in natively relies upon the export feature of the *Rhapsody XMI Toolkit*. Figure 17 shows the wizard that is displayed by Eclipse when the user selects this functionality. Label (1) in Figure 17 shows the menu item that activates the function *Export Model*. Label (2) shows the *Exporter* wizard from the *Rhapsody XMI Toolkit* that opens when this function is selected. Once the *Rhapsody XMI Toolkit Exporter* wizard is displayed, the user is expected to select the folder with the domain model to export, and then press *next* to select the folder where to save the exported file, i.e. the folder named *model* in the UMTG project.

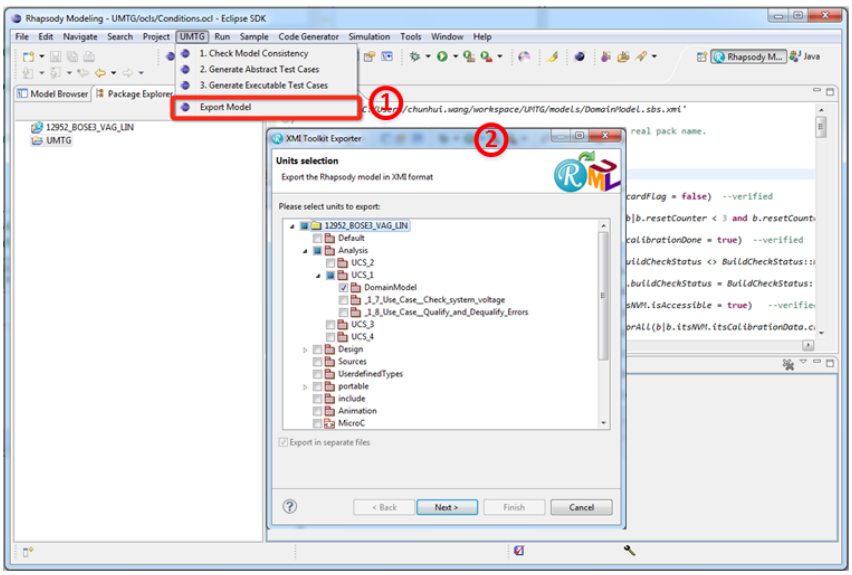


Figure 17: Function *Export Model*

5.4 Check Model Consistency

Once the domain model has been exported, users can use the function *Check Model Consistency* to check the consistency between the domain model and the use case specification. UMTG will generate a report in HTML format, this report will be then automatically opened with the default browser.

In case of inconsistencies, i.e. entity names present in the use case specifications but not in the domain model, software engineers can repair the domain model by adding domain entities using the Rhapsody plug-in, execute again function *Export Model*, and verify again the model consistency. Software engineers are not forced to build models that are fully consistent with the use case specifications, model consistency is useful because it helps software engineers in writing appropriate OCL constraints.

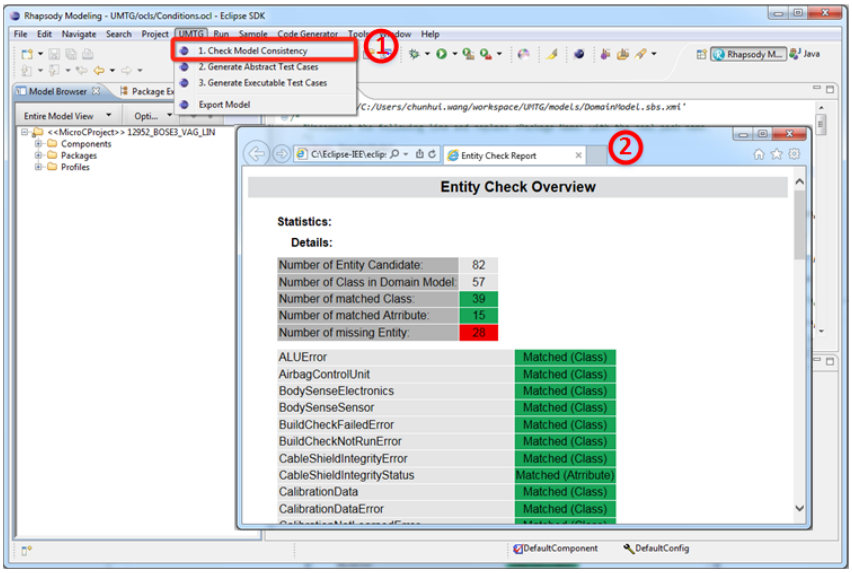


Figure 18: Function *Check Model Consistency* and its output.

5.5 Generate Abstract Test Cases

The function *Generate Abstract Test Cases* is used by software engineers after they have prepared a domain model consistent with the use case specifications, and after they have reformulated all the constrains into OCL.

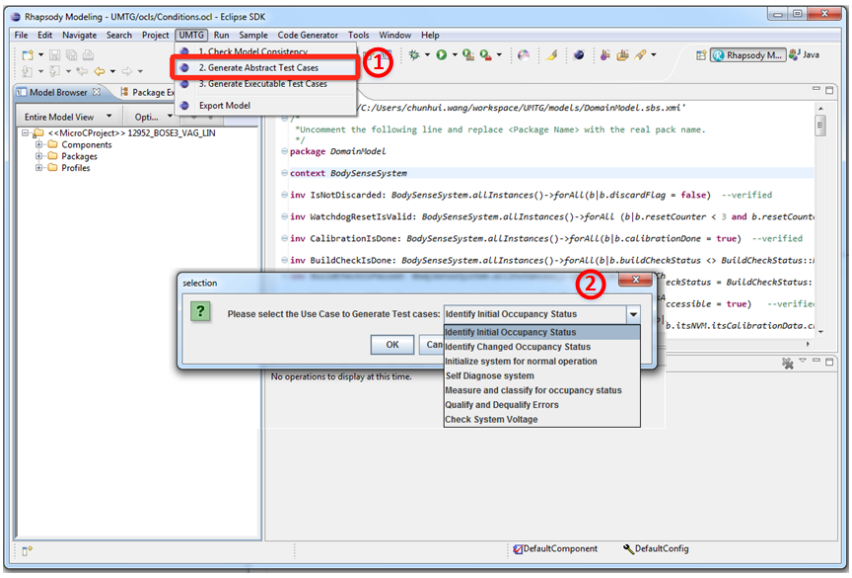


Figure 19: Function *Generate Abstract Test Cases* and widget

5.6 Generate Executable Test Case

Function *Generate Executable Test Case* is used to generate executable test cases. Before using this function software engineers must have filled in a mapping table under *ATS/mapping.csv*. The mapping table can be created incrementally, i.e. by first executing function *Generate Executable Test Case*, observing the results, and then improving the mapping table.

The generated test cases are directly loaded by UMTG into DOORS, to this end software engineer must fill the entry *"testcasePath"* in the UMTG configuration file named *UMTG.config.properties*. The entry *"testcasePath"* indicates the path where to store test cases into the DOORS database.