



User's Guide

Contents

- 1 General Information..... 3
 - 1.1 System and Testing Artefacts 3
 - 1.2 Testing Methodology 6
- 2 Installation..... 8
 - 2.1 User Installation 8
 - 2.2 SVN Installation 11
- 3 Demo Project..... 13
 - 3.1 Creating a new TTS Project..... 13
 - 3.2 Static System model 14
 - 3.3 Test Model..... 16
 - 3.4 Checks on the model 16
 - 3.5 Test generation..... 18
 - 3.5.1 Adapter Generation..... 18
 - 3.5.2 Test Code Generation..... 18
 - 3.6 Test Execution and Evaluation 19
- 4 Configuration..... 21
 - 4.1 Database Configuration..... 21
- 5 Test Reports..... 23

1 General Information

Telling TestStories (TTS) is a system testing tool that has been developed within a research cooperation between the research group Quality Engineering at the University of Innsbruck and the SMU Softmethod GmbH.

In this chapter we give an overview of the main concepts and the methodology of TTS which is needed to understand the tool and to apply it in the right way.

1.1 System and Testing Artefacts

Figure 1 shows the artefacts of the TTS framework. Informal artefacts are depicted by clouds, formal models by graphs, code by transparent blocks and running systems by filled blocks.

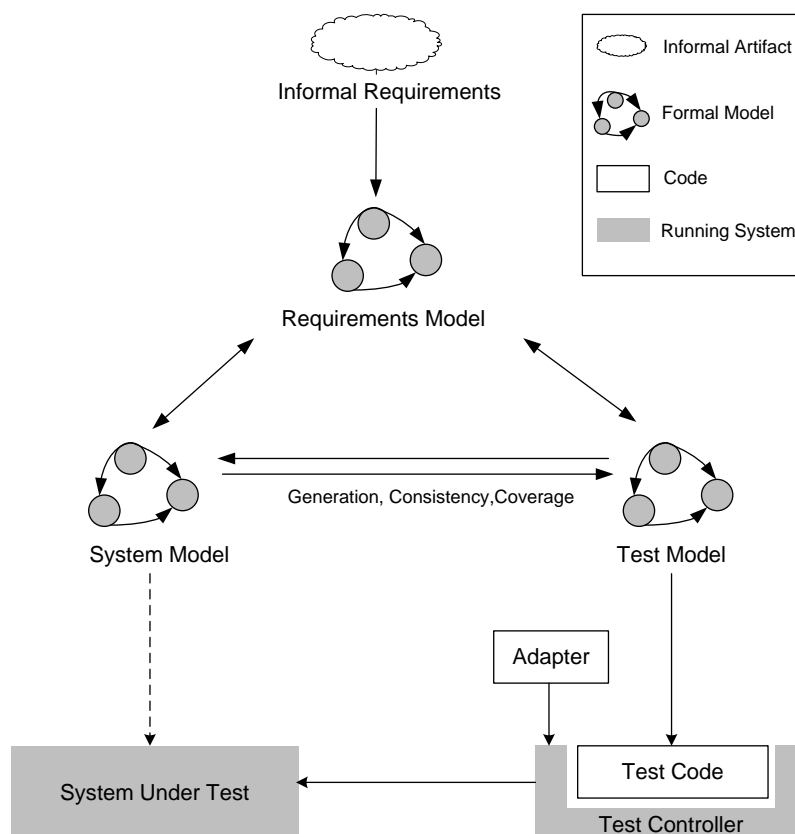


Figure 1 TTS System and Testing Artefacts

In the following paragraphs we explain the formal models and the code fragments of Figure 1 in more detail. The **Informal Requirements**, i.e. written or non-written capabilities and properties of the system, and the **System** providing and requiring services callable by the test controller, are not discussed in detail because they are not the main focus of our testing methodology.

Requirements Model. The requirements model describes the requirements for system development and testing in a formal way. It consists of actors, use cases, domain types, and requirements hierarchies denoted in use case diagrams, class diagrams, and requirements diagrams. The formal requirements are based on written or non-written informal requirements depicted as cloud.

System Model. The system model describes the system structure and system behaviour in a platform independent way. Its static structure is based on the notions of services, components and types. Each service operation call is assigned to use cases, actors correspond to components providing and requiring services, and domain types correspond to types. We assume that each service in the system model corresponds to an executable service in the running system to guarantee traceability. Therefore the use cases, the service operations and the executable services are traceable.

Test Model. The test model defines the test configuration, the test data and the test scenarios as so called test stories. *Test stories* are controlled sequences of service operation invocations exemplifying the interaction of actors. Test stories may be generic in the sense that they do not contain concrete objects but variables which refer to test objects provided in tables. Test stories can also contain setup resp. tear down procedures and contain assertions for test result evaluation. The notion of a test story is principally independent of its representation. We have used UML activity diagrams and sequence diagrams so far.

If the system model and the test model are created manually, it has to be guaranteed that they are consistent with each other and that the test model fulfils some coverage criteria with respect to the system model for consistency and coverage examples). Alternatively, if the system model is complete then behavioural parts of the test model can be generated, or otherwise if the test model is complete, behavioural fragments of the system model can be generated.

Each test story is linked to a use case and can be considered as part of the requirements. Our approach is suitable for test-driven modelling resp. development because the test stories can be defined before the behavioural artefacts of the system model or the system implementation are available. Test-driven development is possible because from test models and adapters it is possible to derive executable tests even before the implementation has been finished. Test-driven modelling

can be applied because the test model can be defined before the behavioural system models whose design can be supported by checking consistency and coverage between the system and the test model. In a system--driven development approach behavioural artefacts can be used to derive test models and test data.

Test Code. The test code is generated by a model--to--text transformation from the test model. It generates test code that can be executed by a test controller.

Adapters. The adapters are needed to access service operations provided and required by components of the system under test. For a service implemented as web service, an adapter can be generated from its WSDL description. Adapters for each service guarantee traceability.

System Under Test. The system under test (SUT) is a service oriented system that may contain additional interfaces for testing purposes.

1.2 Testing Methodology

Figure 2 shows the workflow of our testing methodology. The methodology of our framework supports test-driven development of systems on the model level.

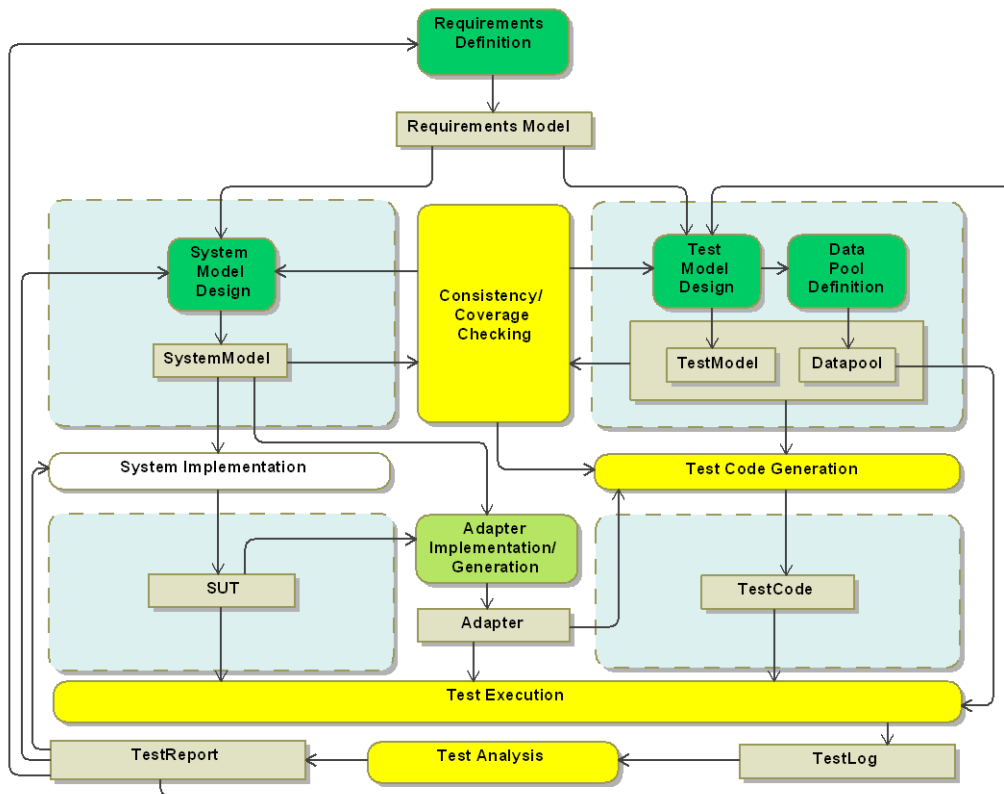


Figure 2 TTS methodology

The first step is the definition of requirements. Based on the requirements, the system model and the test model are designed in parallel. The test design includes the data pool definition, i.e. the definition of test data, and the test sequence definition, i.e. the sequence of test stories together with states and data to be tested. The system model and the test model, including the test stories, the data and the test sequences, can be checked for consistency and coverage. This allows for an iterative improvement of their quality and supports model-driven system and test development. Our methodology does not consider the system development itself but is based on traceable services offered by the system under test. As soon as adapters which may be - depending on the technology - generated automatically or implemented manually are available for the system services, the process of test code generation can take place. The generated test code is then automatically compiled and

executed by a test execution engine which logs all occurring events into a test log. The test evaluation is done offline by a test analysis tool which generates test reports and annotations to those elements of the system and test model influencing the test result.

2 Installation

In this chapter we consider two types of installation. On the one hand we describe the installation procedure for users who want to test a project with TTS (user installation) and on the other hand for developers who want to implement some functionality (SVN installation).

2.1 User Installation

TTS is based on the Eclipse Modeling Tools available at <http://www.eclipse.org/downloads/>. So install it first.

Alternatively you can also install modelling components on an Eclipse IDE for Java Developers but this is much more time consuming and error-prone.

Based on your Eclipse installation, apply the following installation instructions:

Install the following software bundles via the Eclipse Update Manager (**Help – Install New Software...**):

1. Install **Apache Jakarta log4j** via the Galileo Update Site (see Figure 3). You have to uncheck “Group items by category” to see the item “Apache Jakarta log4j”.

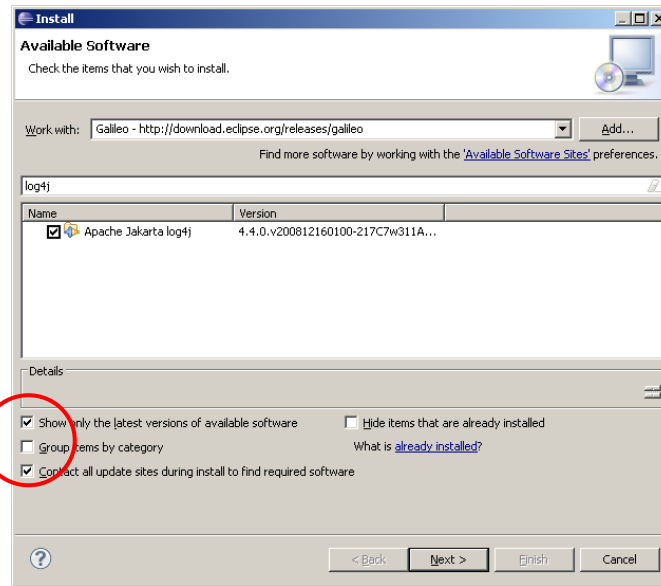


Figure 3 log4j Installation

2. Install **OpenArchitectureWare** via its Eclipse Update Site available at <http://www.openarchitectureware.org/updatesite/milestone/site.xml> (see Figure 4).

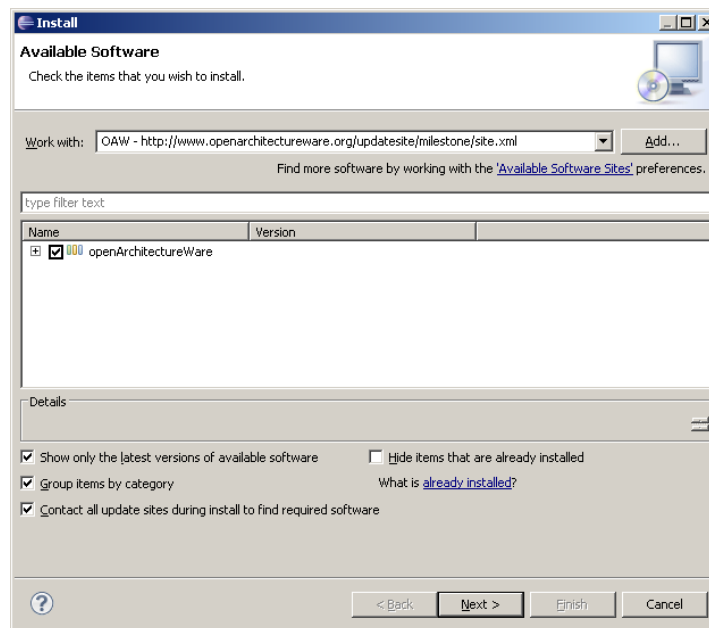


Figure 4 OAW Installation

3. Install BIRT for reporting via the Galileo update site. BIRT is represented by the category **Business Intelligence, Reporting** (see Figure 5).

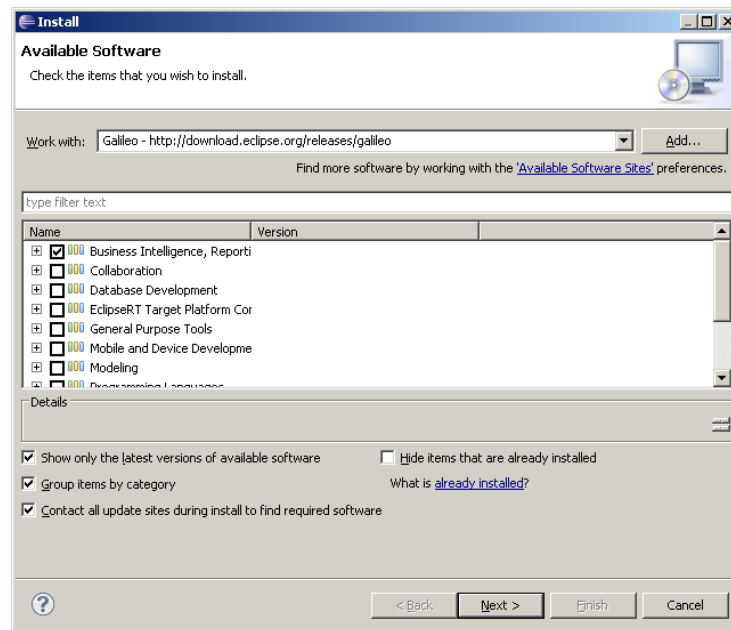


Figure 5 BIRT installation

4. TTS uses Derby or MySQL for logging. If you want to use MySQL which is recommended copy the JDBC driver for MySQL available at <http://dev.mysql.com/downloads/connector/j/5.1.html> into the folder **plugins/org.eclipse.birt.report.data.oda.jdbc_X.X.X.vXXXXXXX/drivers** of your Eclipse installation (see Figure 6).

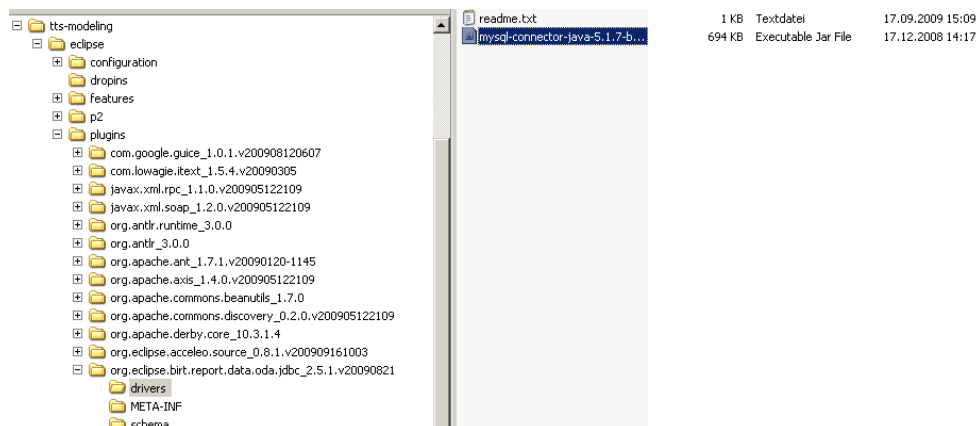


Figure 6 Driver installation

5. Install the TTS plugins via its update site available at <http://www.teststories.info/update/site.xml> or directly by extracting the zip-File containing the TTS plugins available at

<http://teststories.info/update/downloads/TTS.zip> to your eclipse directory (not to the plugins subdirectory) of your Eclipse installation. It may be the case that the TTS update site is temporarily unavailable. In this case apply the second installation procedure.

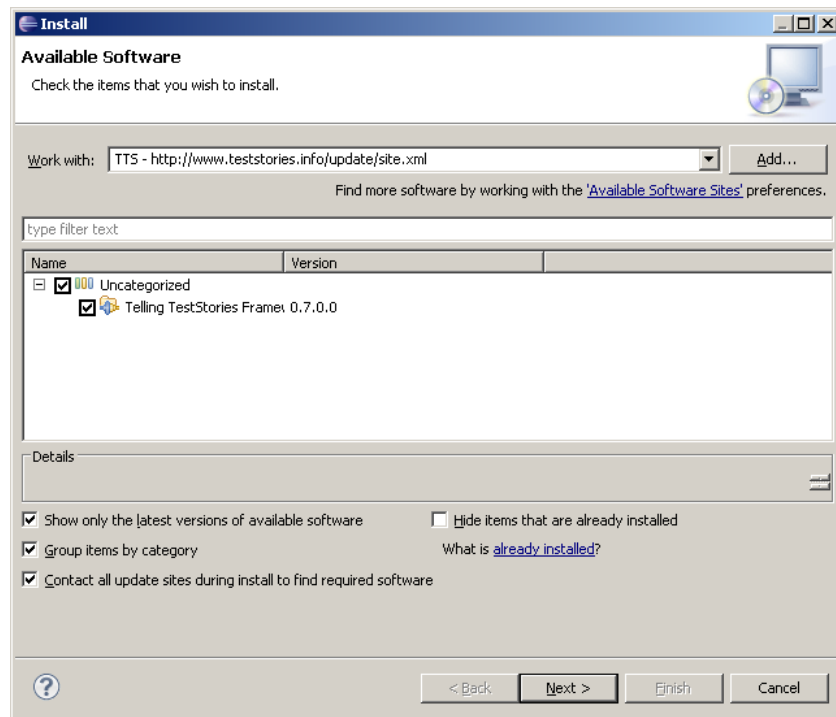


Figure 7 TTS installation

You can test your installation with existing example projects available at <http://teststories.info/download/examples/>.

2.2 SVN Installation

The installation of TTS via SVN is based on the same prerequisites as the installation of TTS via the Eclipse update manager. Therefore first install Eclipse Modelling and then apply the installation steps 1 to 4 of the user installation. After that install an SVN plugin for Eclipse, e.g. Subclipse available at <http://subclipse.tigris.org/>.

Then you can check out the TTS plugin projects from <https://qe-informatik.uibk.ac.at/subversion/tts/src/active/main> and TTS demo projects from <https://qe->

informatik.uibk.ac.at/subversion/tts/src/active/demo and modify them in the same way as other Eclipse development projects.

3 Demo Project

In this chapter we develop test project for the Ping web service. This example is also available online as demo video at <http://teststories.info/documentation/videos/>.

3.1 Creating a new TTS Project

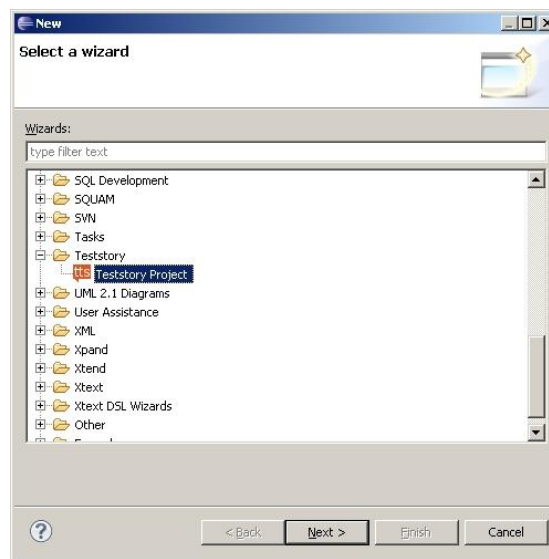


Figure 8 Project Wizard

Figure 8 shows the wizard for the creation of a new TTS project. After the wizard has been executed an empty TTS project is created.

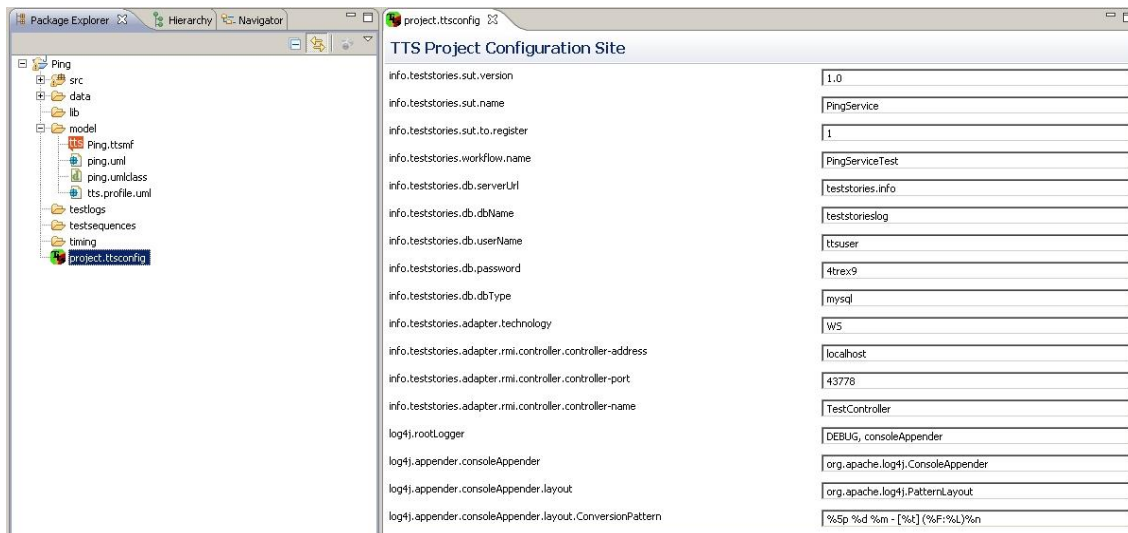


Figure 9 TTS Project and Configuration file

A new TTS project as depicted in Fig. 2 contains the following directories and elements:

- **src** contains the Java source of the adapters and test scripts
- **data** contains the test data in xml files
- **lib** contains libraries for the adapter technologies
- **model** contains the system and test model as EMF UML 2 file
- **testlogs** stores log files of test runs
- **testsequences** contains sequences of test stories to execute
- **timing** contains timing information
- **project.ttsconfig** holds the configuration data for a test project, e.g. which adapter type is used.

This file is depicted in Figure 9

3.2 Static System model

The static system model consists of the classes and instances of them needed as input and return types for services, the components and their provided and required services.

Figure 10 shows the data editors for the class Delay with the attribute timeout. The data editors consists of one editor for data input (Ping.ttsmf) and one for data view (Delay.xml).

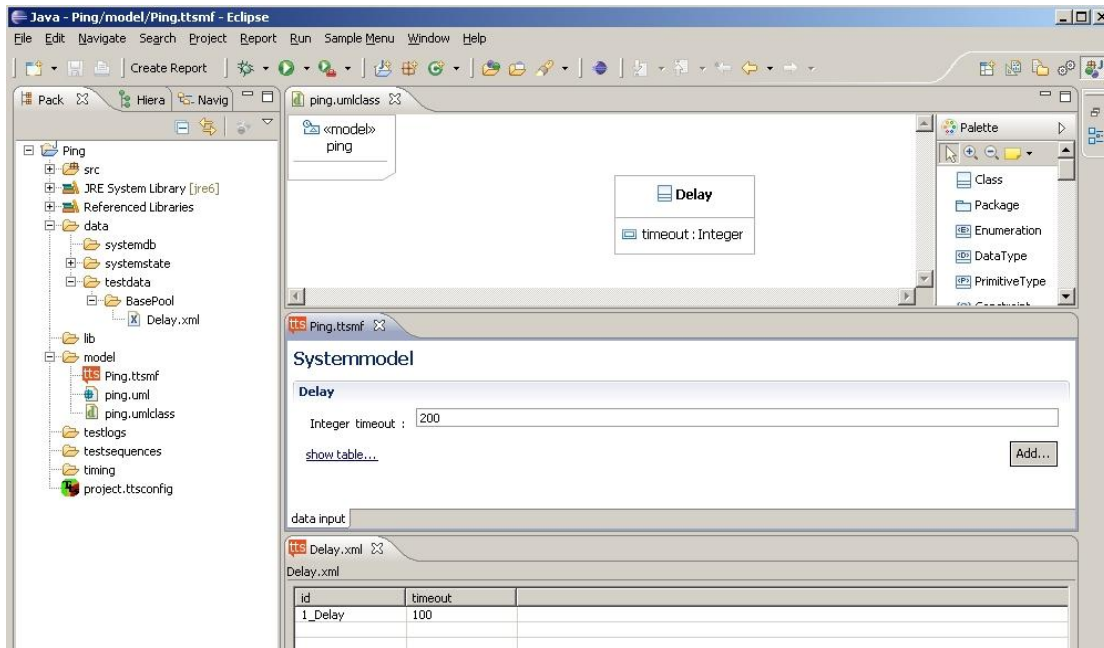


Figure 10 Datainput editor

Figure 11 shows the static parts of the system model that must be modelled before the test model which models some kind of behaviour can be defined. The static system model contains a class (Delay), which is already depicted in Fig. 3 for the data input, a service modelled as interface and a component providing that service.

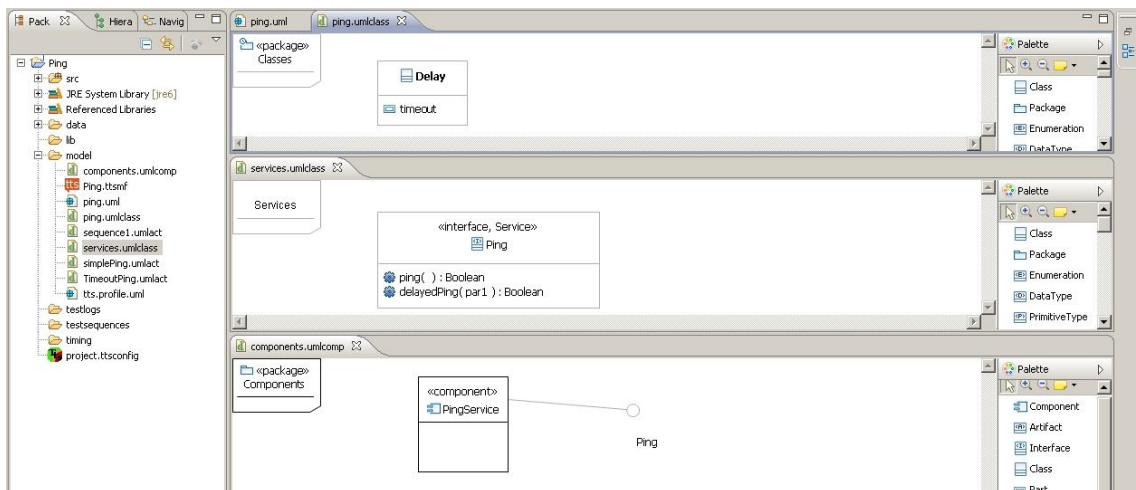


Figure 11 System model

3.3 Test Model

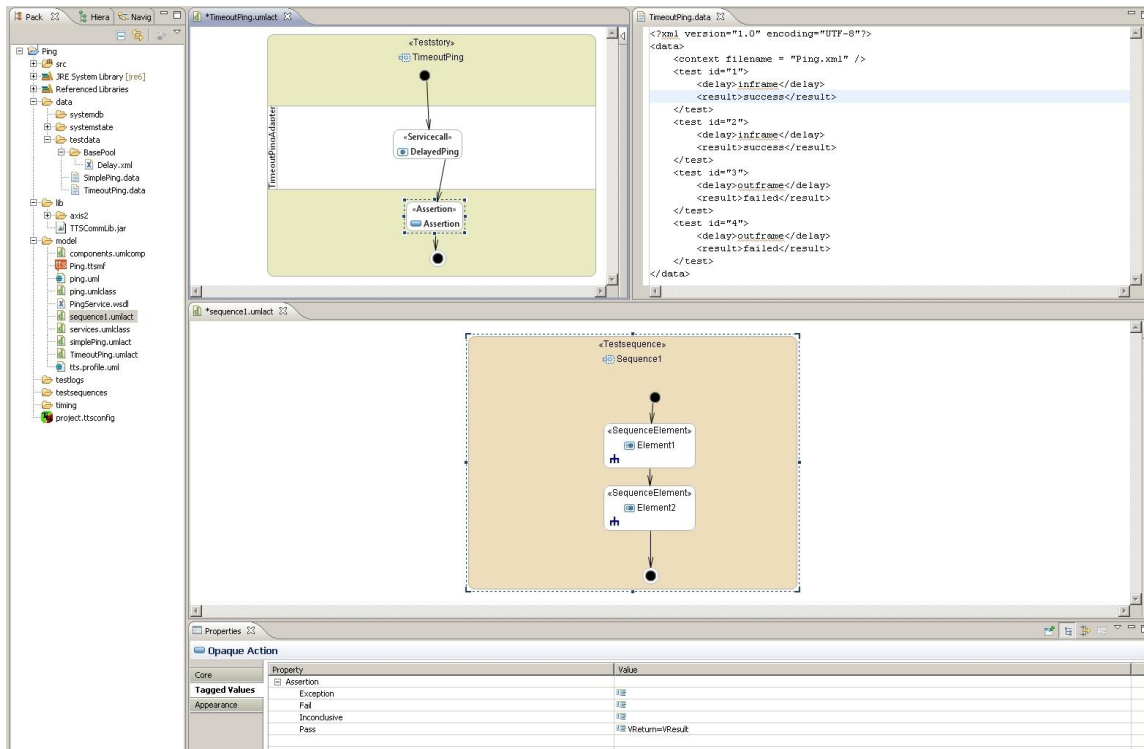


Figure 12 Test Model

The test model contains test stories modeled as UML activity, e.g. TimeoutPing.umlact and corresponding test data, e.g. TimeoutPing.data. The test sequence, e.g. Sequence1.umlact, defines the execution order of the test stories and the arbitration for each sequence element. The test story contains service calls and an assertion, which is defined in the property editor at the bottom.

3.4 Checks on the model

The system and test model are stored together in one EMF UML2 file depicted in Fig.6. We have integrated the SQUAM framework (<http://squam.info>) to execute OCL checks on the system model, the test model and between them. We have defined correctness, consistency and coverage checks. In the right editor window of Figure 1 one can see a correctness check (hasAssertion, which checks whether a test story has at least one assertion), a consistency check (hasTeststoryUniqueName, which checks whether a test story has a unique name) and a coverage check (allOperationsCoverage, which checks whether all operations off all services are checked in at least one test story).

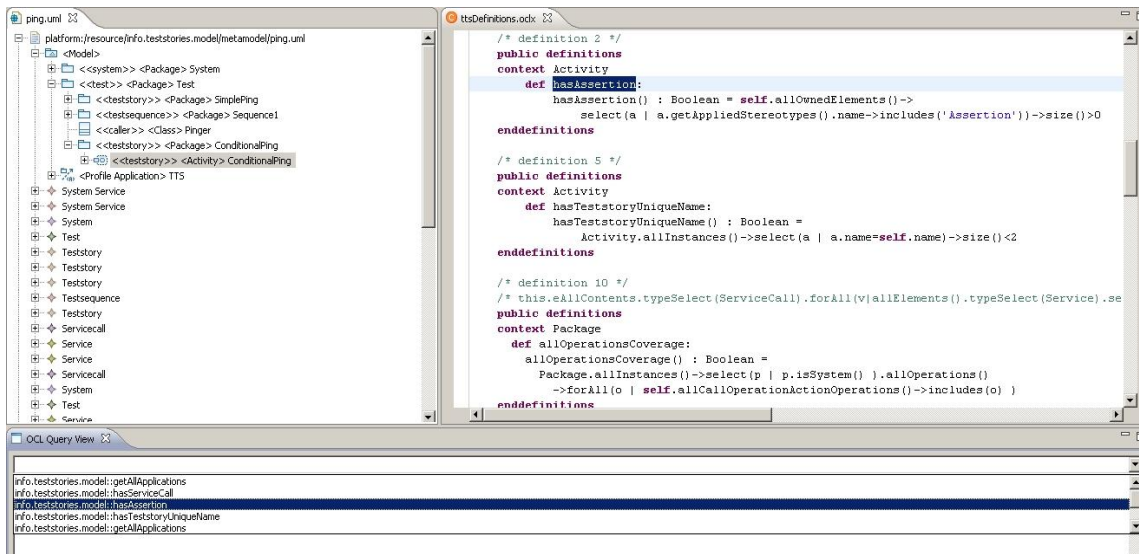


Figure 13 Correctness, consistency and coverage checks for the system and test model

After the checking the correctness, consistency and coverage of the system and test model, the test generation can be started.

3.5 Test generation

The test generation consists of two parts. On the one hand adapters to access the executable system services are needed, and on the other hand the test code for the test sequence and the test stories has to be generated.

3.5.1 Adapter Generation

Technically adapters are classes that implement a specific Java interface. For web services these adapters can be generated from an existing WSDL file defining the service interfaces (see Figure 14 for the generation context menu entry and the resulting adapter).

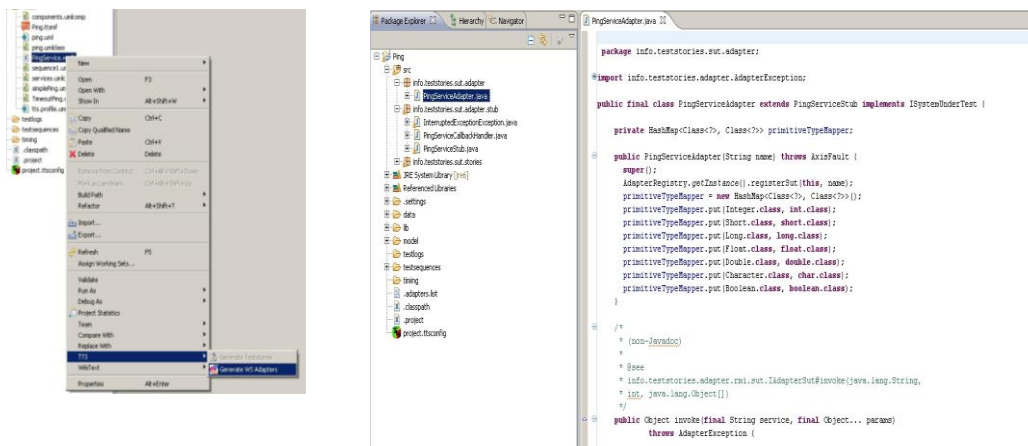


Figure 14 Web Service Adapter Generation

We have also implemented adapters for the Java RMI technology manually.

3.5.2 Test Code Generation

The test code generator creates from the EMF UML2 file executable java files for every test story and sequence files from the test sequences (see Figure 15) which can then be executed.

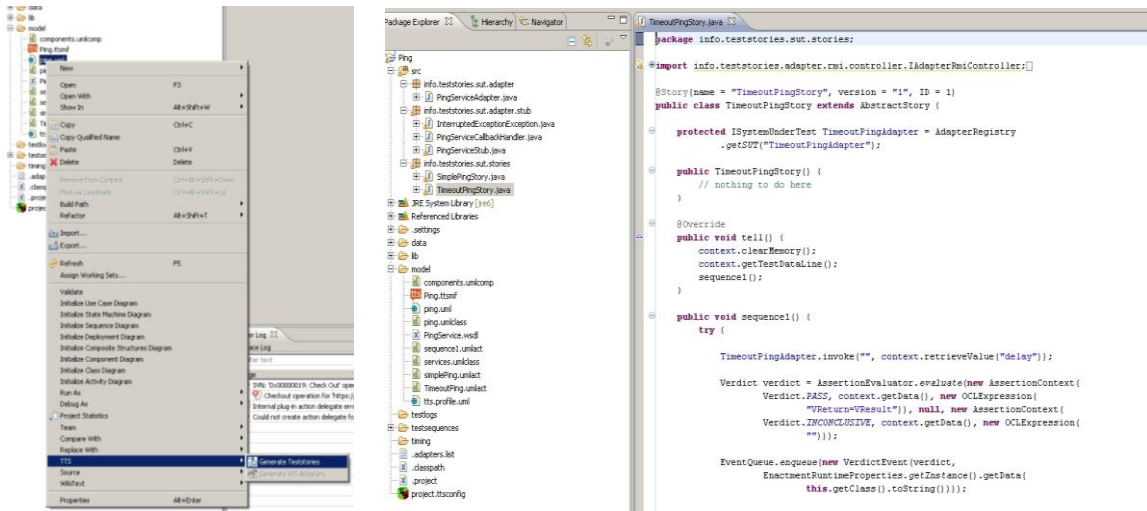


Figure 15 Test Code Generation

3.6 Test Execution and Evaluation

The test execution can be started as run configuration from a test sequence (left part of Figure 16). After the test execution, the test result can be evaluated in special views. In the right part of Figure 16 a JUnit-like view listing every test case for every test story is depicted.

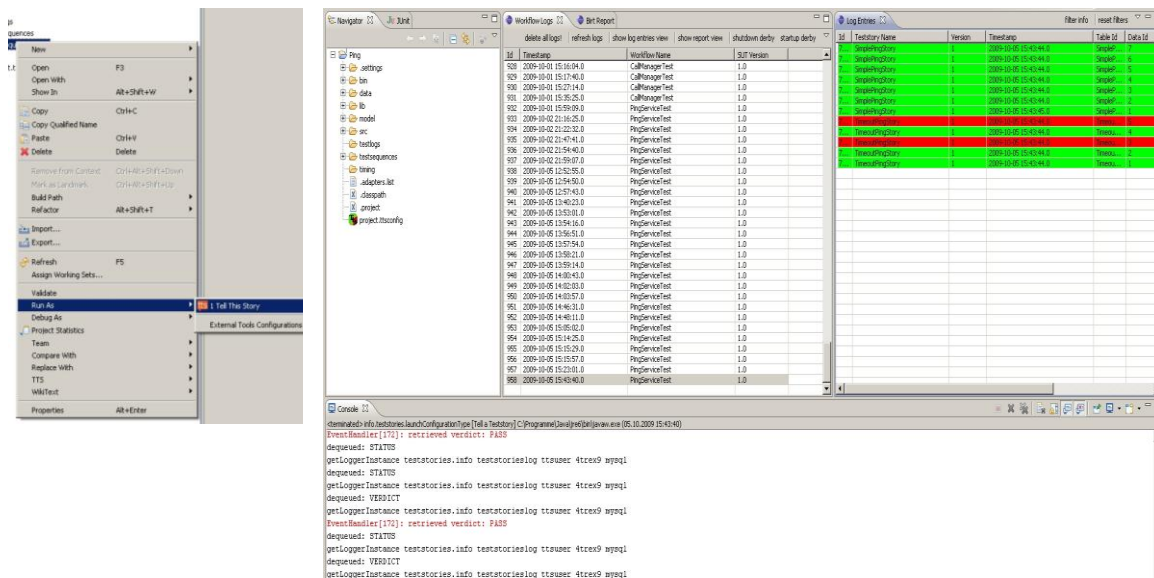


Figure 16 Test Execution

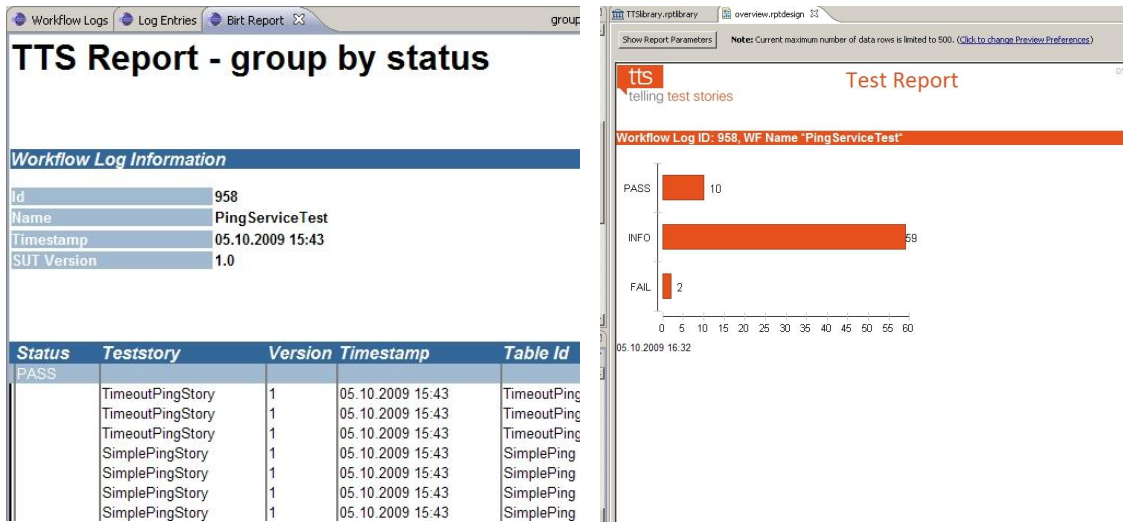


Figure 17 Test Report

We have also implemented textual and graphical reports for test evaluation (see left part of Figure 17). Tailored reports can be defined via a report library (see right part of Figure 17).

4 Configuration

4.1 Database Configuration

The TTS database has to be configured in Window/Preferences/TTS (see Figure 1) and in project.ttsconfig (see Figure 19).

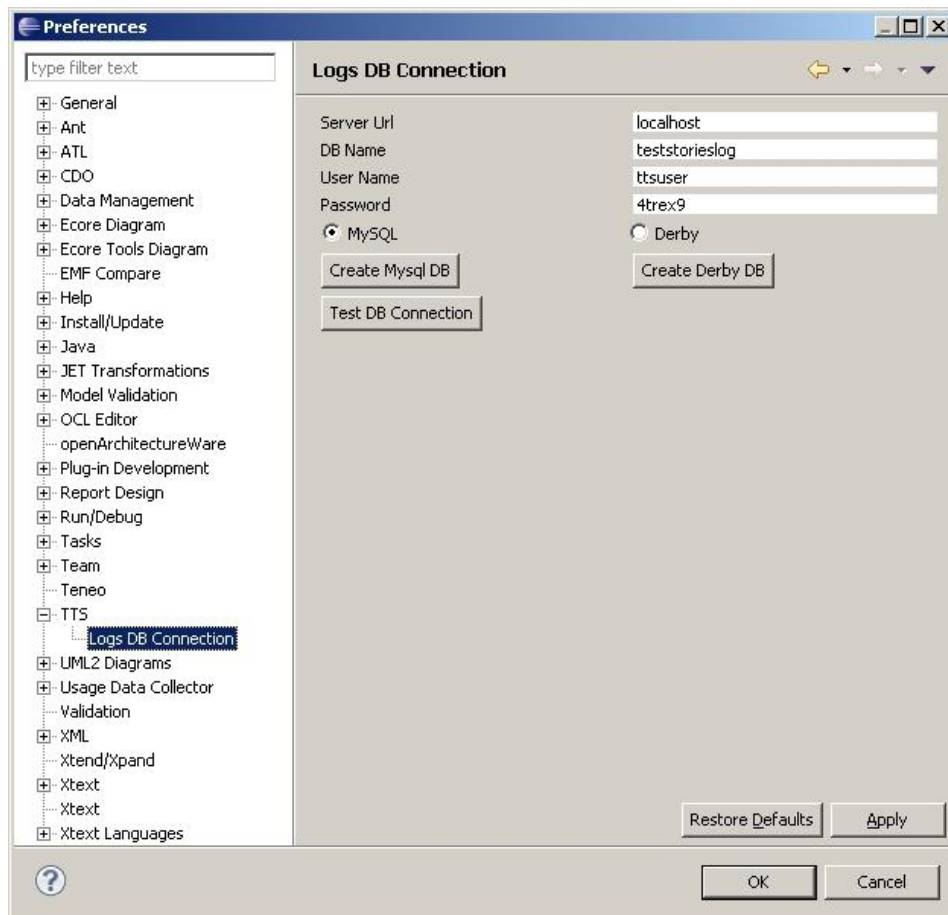


Figure 18 Preferences

project.ttsconfig

TTS Project Configuration Site

info.teststories.sut.version	1.0
info.teststories.sut.name	PingService
info.teststories.sut.to.register	1
info.teststories.workflow.name	PingServiceTest
info.teststories.db.serverUrl	teststories.info
info.teststories.db.dbName	teststorieslog
info.teststories.db.userName	ttsuser
info.teststories.db.password	4trex9
info.teststories.db.dbType	mysql
info.teststories.adapter.technology	WS
info.teststories.adapter.rmi.controller.controller-address	localhost
info.teststories.adapter.rmi.controller.controller-port	43778
info.teststories.adapter.rmi.controller.controller-name	TestController
log4j.rootLogger	DEBUG, consoleAppender
log4j.appender.consoleAppender	org.apache.log4j.ConsoleAppender
log4j.appender.consoleAppender.layout	org.apache.log4j.PatternLayout
log4j.appender.consoleAppender.layout.ConversionPattern	%5p %d %m - [%t] (%F:%L)%n

Figure 19 Configuration file

Alternatively, one can create a TTS database manually by importing the database dump `tts.sql` available online and executing the following commands in MySQL:

```
create database teststorieslog;
create user 'ttsuser'@'localhost' identified by '4trex9';
grant all on teststorieslog.* to ttsuser@localhost identified by "4trex9";
mysql -p -u root teststorieslog < tts.sql
```

5 Test Reports

TTS provides BIRT Reports for evaluating test runs. Some predefined reports can be found in the repository (). However, BIRT enables to define BIRT Libraries to create and deploy easy to use and tightly controlled report writing capabilities. In the case of TTS this means, that the TTSLibrary defines standard elements of reports that are often used in the context of TTS, e.g. the header of a report, a table including the log lines of a test run, etc. There are mainly two big advantages in using a BIRT library:

- Predefined elements (Report Items) make it easier for users to create new reports
- Changes in the library will automatically be propagated to all reports using this library (e.g. a change of the logo in the library's header element will change also the logo in all reports that use the library's header element)

In this section we shortly explain the different steps needed to create a report with the help of the TTSLibrary.

1. Create new (blank) BIRT report.
2. Open the Eclipse Resource Explorer and go to the TTSLibrary.rptlibrary depicted in the following Figure.

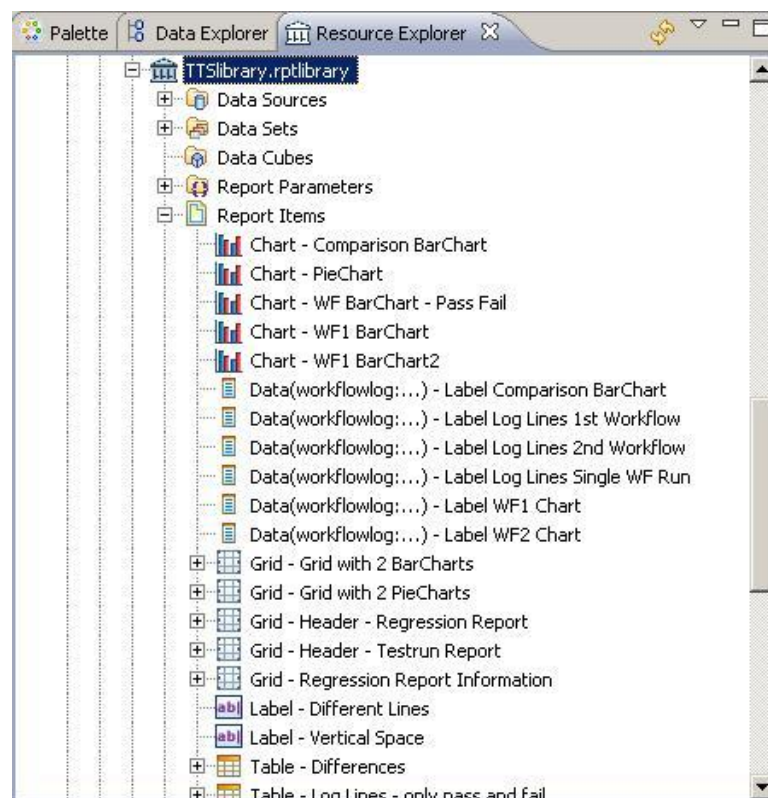


Figure 20 Eclipse Resource Explorer

- Copy the following elements from the TTSLibrary (depicted in the Resource Explorer tab) to your new report (depicted in the Outline tab), by keeping the correct directory structure (e.g. copy the ttsSource from the Data Sources directory of the TTSLibrary to the Data Sources directory of your new report): Data Sources (ttsSource), all needed Data Sets and Report Parameters.
- Now you can add with simple drag and drop Report Items from the library to your new report (see Figure 21)

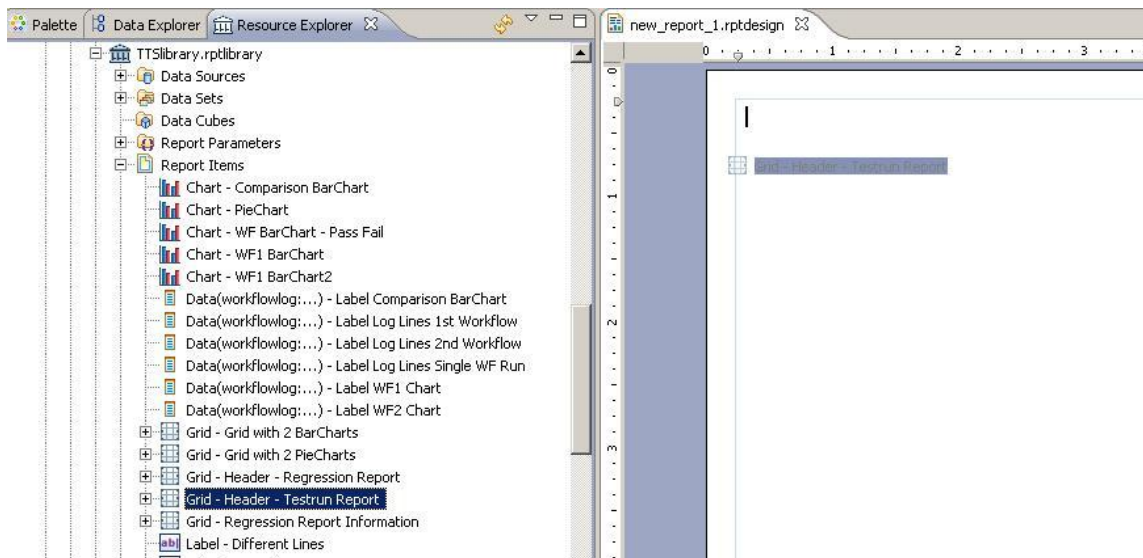


Figure 21 Add Elements with drag and drop

- After adding all elements you can look at your report by changing to the Preview tab or by generating a .pdf, .doc, .html, etc.

Some important Report Items of the TTSLibrary are explained in the following:

- Grid – Header – Regression/Test Report:** item represents the header of each TTS report
- Data(workflowlog:...) - Label *:** denote labels for different diagrams or tables
- Table – Log Lines 1st/2nd Workflow:** a table that shows all log lines of a test run
- Table – Log Lines – only pass and fail:** a table that only shows log lines with the status “pass” or status “fail” (log lines with status “inconclusive”, “timeout”, etc. are ignored)
- Table – Differences:** shows the log lines with different status elements of two compared workflow runs
- Chart – PieChart:** shows the distribution of log lines of a workflow grouped by status of the log events drawn in a pie chart

- **Chart - * BarChart:** shows the distribution of log lines of a workflow grouped by status of the log events drawn in a bar chart
- **Chart – Comparison BarChart:** shows the distribution of log lines of two workflows grouped by status of log events per workflow drawn in one single bar chart

5.1 Adding Reports to the Reporting Perspective

To add a custom tailored report design to the Reporting Perspective of TTS, modifications to the class `ttsreportmanager.view.BirtReportView`, implemented in the package `info.teststories.reportmanager` plug-in, have to be applied. After opening the class, the following four TODO tasks indicate the extension points in the source code:

1. `//TODO DEFINE CUSTOM ACTIONS`
2. `//TODO ADD CUSTOM ACTION TO PULLDOWN`
3. `//TODO ADD CUSTOM ACTION TO TOOLBAR`
4. `//TODO IMPLEMENT CUSTOM ACTIONS`

At the first one, nothing more than the actions are defined (see the source following the TODO tag for a living example).

The second and third add the actions to the views' toolbar and pull-down menu, respectively. For sure, there are other possibilities where to add the button, yet this is not a guide about eclipse UI programming, therefore the interested reader is advised to take a look at the eclipse corner articles. At the last tag, simply the code between the two comments `/** REPORT ACTION IMPLEMENTATION BEGINNING */` and `/**REPORT ACTION IMPLEMENTATION BEGINNING */` has to be copied.

Finally, the variable name has to be replaced by the actual action, the name of the report has to be adopted in the call of the `FileLocator.find(...)` operation and, last but not least, the messages printed by the message dialogs and the actions' text remain to be changed.

After successfully rebuilding and deploying the plug-in, the report is available in the view.