Embedding a Test Tool in Eclipse Java Development Environment

Maria Rosaria Morbidelli

IBM Software Group Rome Laboratory

00144 Rome, Italy
rosa.morbidelli@it.ibm.com

Abstract. For delivering high quality software products, it is important to test them in environments very close to the typical customer ones, where unexpected events may perturb the normal running of the application. Running tests in a realistic environment starting from the early development phases helps in reducing costs in terms of fixing. Moreover, in a distributed application, the importance of testing each single component increases when Agile methodologies are adopted because it can happen that a component is developed when its pre-requisites are not available.
Often the customer environments are very complex and replicating them in a laboratory is difficult if at all possible.
The aim of this paper is to describe the integration in the Eclipse IDE of a test tool useful for simulating the operating conditions of typical customer environments. This allows to seamlessly run complex tests directly from the Eclipse IDE without switching to a different environment.

Keywords: Java, Eclipse, plug-in, random, unit test, injection, hook, instrument, byte-code.

1 Introduction

In a commercial development organization, the process of identifying, correcting and verifying defects during the software development process is a very expensive activity; the later the problems are found the better costs increase.
Improving the effectiveness of software testing from the early phases of the development is an important aspect to increase the quality of a software application.
For a distributed application the efficiency of the test of a single component (Unit Test) is often affected by environment and configuration constraints: for user interfaces it could be not easy to setup a development environment that is connected to the back end; for core components that interacts with a database, it could not be easy simulating situations like concurrent accesses, huge numbers of data, loosing of database connection, and so on.

Moreover, adopting the Agile methodology, the development of all the Software Product application components may start at the same time, regardless the logical sequence the components should be developed.
In all those cases, it could be useful having the possibility to use tools for scaffolding data and simulating situations that could occurs on a typical customer environment.

JITAT (Just-in-time Injector Test Automation Tool) is a tool that can be used for such purposes. It has been developed in the IBM Software Group Rome Laboratory for internal test purposes and allows scaffolding data and simulating complex situations that usually may occur in a typical customer environment (such as: loss of network connectivity, unavailability of a needed service provider, JVM crashes) at runtime and without modifying the application code.

JITAT is a tool that can also run independently from Eclipse. Since in the IBM Tivoli Rome Laboratory the standard development platform is Eclipse, the solution to integrate JITAT in the IDE plays a key role in helping us in executing, starting from the Unit Test, a set of tests that normally would be performed only in the later phases of the test process.

This paper will describe the JITAT tool functionalities (Section 2), how JITAT has been integrated in the Eclipse IDE (Section 3), some JITAT plug-in internals (Section 4), some possible future enhancements of the plug-in (Section 5).

2 What's JITAT

JITAT is a tool can be used for testing Java applications.
It allows to:

- Emulate the presence of external components needed by the application that are not available.
- Test the recovery mechanisms and fault management capabilities of the application in case of unattended environmental situations like abruptly crashes, loss of network connectivity, shutdown of a server, temporary loss of database connectivity and so on.
- Test the presence of conditions that may create dangerous deadlocks.
- Obtain a deterministic behavior in presence of random events, by a record/reply mechanism that allows to record each generated event and to reproduce the same event at the same instant in a subsequent execution of the same test.

JITAT works by using a hooks mechanism: it instruments Java classes at runtime, doing the 'just-in-time' byte-code instrumentation of the application by injecting predefined or customized hooks in well defined locations of the application Java classes when they are loaded into the memory of the JVM.

In details, JITAT intercepts the entry, the exit of methods and the throwing of exceptions and inserts its own code in order to alter the behavior of the original method.

JITAT allows instrument the Java classes by two types of hooks:

- Synchronous hooks: which are executed in the context of a method invocation; they are of two types:
    - Replacing hook: that provides the capability of overwrite at runtime classes methods.
    - Augmenting hook: that allows to adding code at the beginning and at the end of a class method.
- Asynchronous hooks: that are executed asynchronously in a separate thread and allows simulating unattended situations like network interruption, database connection loosing, random crashes of the JVM processes, etc.

Each hook can be bound to a specific set of classes/methods by user configurable filters, which have to be specified in a configuration file.

JITAT provides a library of predefined hooks (synchronous and asynchronous power-off simulator, a fault injector, a delay/thread-switch injector) and allows the user to define also its own hooks.

The configuration of the JITAT is XML-based.

3 JITAT and Eclipse

JITAT has been integrated into the Eclipse Java Development Environment as a plug-in.

The JITAT Eclipse plug-in provides the developer a simple way to:

- Manage (create, edit, delete) one or several JITAT configurations (see **Fig. 1**).
- Create custom hooks by providing pre-defined templates for the hook classes and compile them (see **Fig. 2**).
- Bind the Java classes/methods of an application to a JITAT hook directly from the Eclipse Java perspective.
- Run a Java application instrumented by JITAT from the Eclipse platform (see **Fig. 3**).

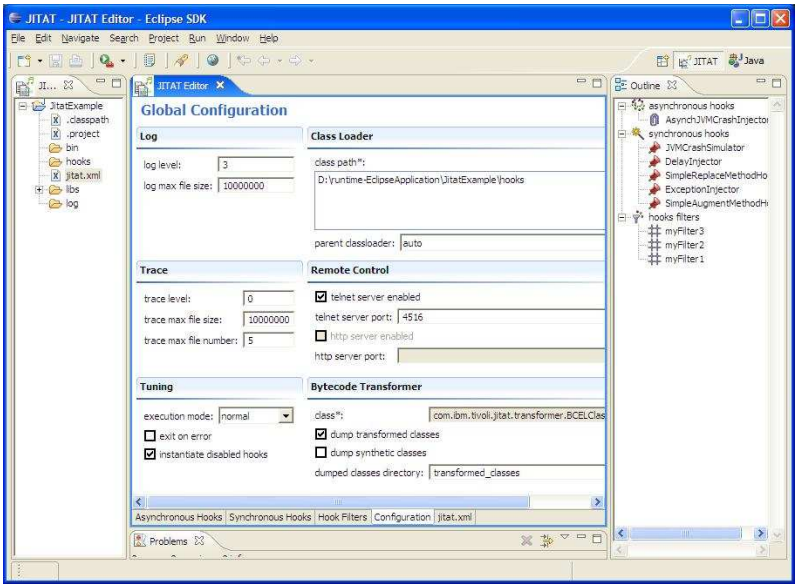

Fig. 1. This figure shows the editor of the XML JITAT configuration file: it allows to define general configuration parameters (*Configuration* tab), synchronous (*Synchronous Hooks* tab) and asynchronous hooks (*Asynchronous Hooks* tab) and the methods and Java classes where the defined hooks apply (*Hook Filters* tab).

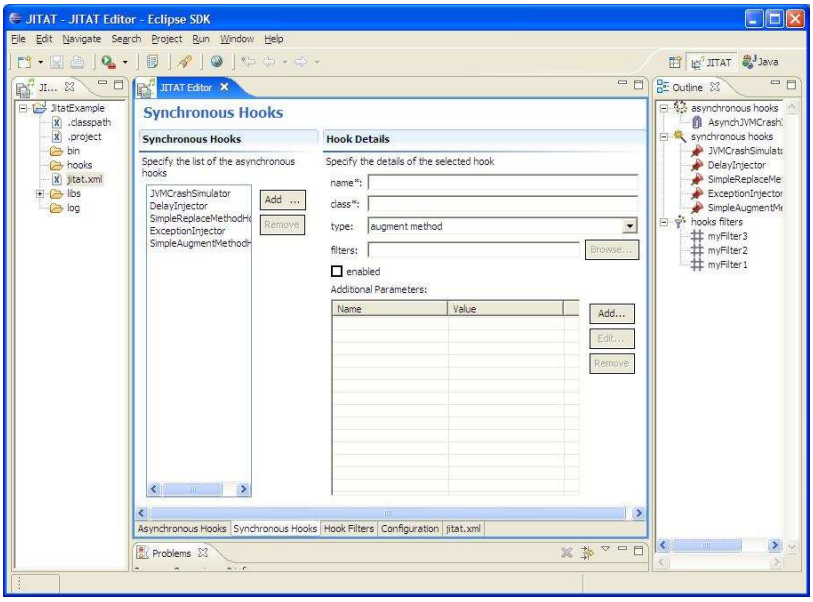

Fig. 2. This figure shows the hooks section of the JITAT configuration file: in this section it is possible to define the hooks and, eventually, their parameters.

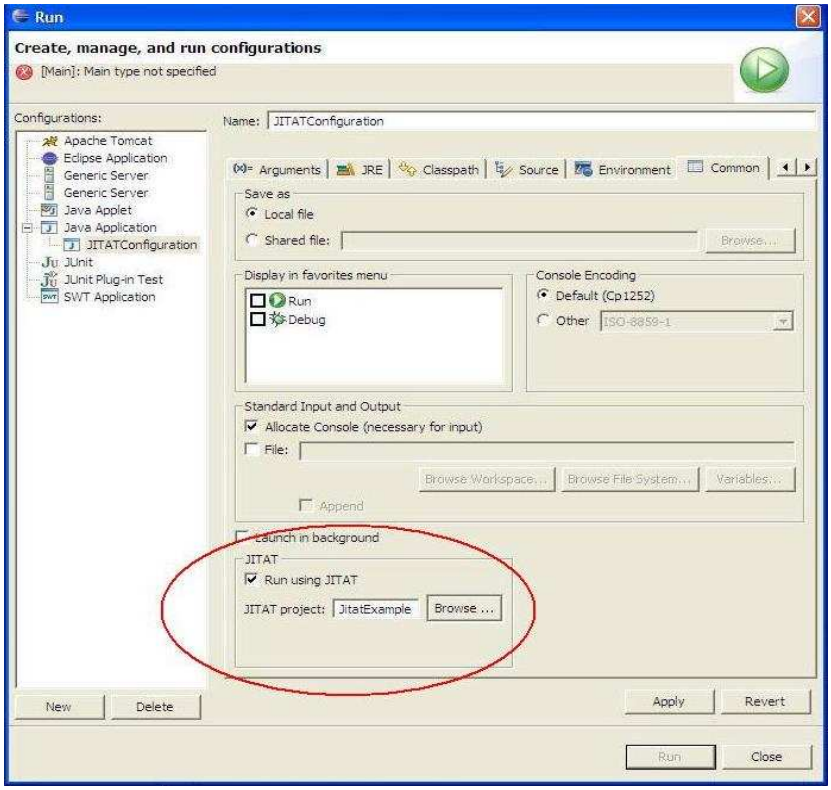

Fig. 3. This figure shows how to run a Java application instrumented by JITAT just setting a flag and choosing the JITAT configuration in the Eclipse Launch Configuration dialog.

Using the JITAT plug-in it is possible defining several configurations; this provides a friendly way for proceeding by steps in the Unit Test and for starting the test of components that usually needs other components available before be tested, such as the user interfaces when back end components are not been developed yet.

As a first step, the developer can define a JITAT configuration for scaffolding data that could come from the back end. This is possible by defining Synchronous Replacing hooks that provides the data to the specific classes methods.

In example, let's consider a User Interface that has to show data retrieved from a database; *getDataFromDB* is the method that establishes the database connection and run the query for getting the data from the database; using the JITAT plug-in, the user can easily define a replacing hook that simulates the presence of the database reading the data from a file; let's call this hook *SimulateDataRetrievingHook*. Binding the *SimulateDataRetrievingHook* with the *getDataFromDB* method in the JITAT configuration file, and running the User Interface with such defined JITAT

configuration, the user interface will run also without the data base since it will show the data in the file instead of the data in the database.

As a second step, the user can define a JITAT configuration for simulating errors and the occurrence of unattended situations, like connection failures, network interruption, and so on. This is possible by defining custom Synchronous/Asynchronous hooks or using some of the built-in hooks provided in the JITAT library.

As a third step, the developer can define a JITAT configuration for making scalability and performance tests by simulating bulk data coming from backend. This is possible by defining custom Synchronous/Asynchronous hooks.

This solution allows achieving high test coverage of a software application starting from the Unit Test development phase. Moreover, the capability to store several JITAT configurations and to switch in a simple way from one configuration to another one and the record/replay capability provided by the tool, allows a simple way to test problems' fixes to test problem fixes with a deterministic method, also when the test could require the recreation of random situations.

4 JITAT Plug-in Internals

JITAT plug-in interacts with Eclipse Java Development environment by defining the extension points listed in the **Table 1.**

Table 1. JITAT plug-in extension points.

Extension Point	Java Class	Description
org.eclipse.ui.perspective		Defines a new perspective specific to the JITAT plug-in
org.eclipse.ui.view	JITATNavigatorView	Defines the Navigator View of the JITAT perspective.
org.eclipse.ui.editor	JITATFormEditor	Defines the JITAT configuration file editor
org.eclipse.ui.newWizard	NewProjectWizard	Provides a wizard for creating specific projects where defining JITAT configurations and custom hooks
org.eclipse.ui.popupMenus	LinkToHookAction	Contributes with new menus and actions, like binding a method of an application Java class to a JITAT Hook (built-in or custom)

org.eclipse.debug.ui.launchConfigurationTabGroups	JITATLaunchConfigurationTabGroup	Contributed with the JITAT options in Java Launch Config. tab
org.eclipse.core.resources.natues	JITATProjectNature	Defines the specific nature for a JITAT Project

The class diagram below (see **Fig. 4**) describes how the JITAT plug-in interacts with the Java Launch Configuration in order to add the JITAT settings in the Common tab. This has been implemented by extending the Java Launch Configuration tab group by the *JITATCommonConfigurationTab* tab group, that extends the Java *CommonTab* by adding the JITAT settings.

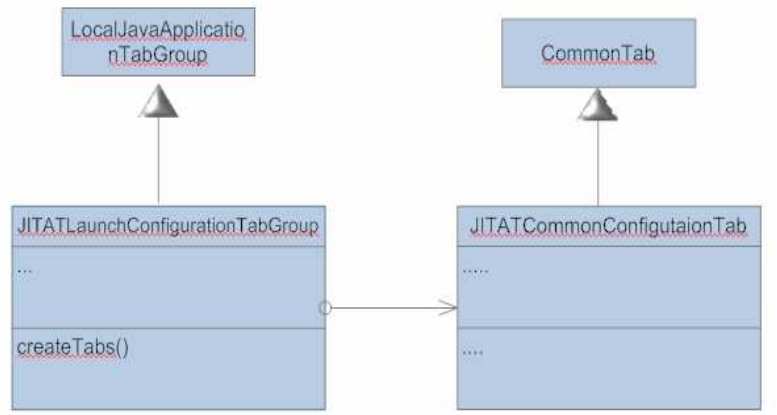

Fig. 4. This class diagram shows how the JITAT plug-in contributes in Launch Configuration Common Tab.

JITATLaunchConfigurationTabGroup is the class that implements the *org.eclipse.debug.ui.launchConfigurationTabGroups* extension point: in the *createTabs()* method, it creates the tabs shown in the Eclipse LaunchConfigurationDialog: in details, it creates a *JITATCommonConfigurationTab* that overwrites the CommonTab and provides the JITAT option.

5 Open points

There are some possible enhancements to JITAT plug-in that could improve the usability of the tool:

- Add a Console view where showing live the log of the JITAT.
- Add a view for showing charts representing the statistics provided by JITAT about some interesting execution information, like the number of instrumented methods/classes, the number of the total invocations of the different kinds of hook events, the number and type of the generated hook events for each method and so on.

6 Conclusions

This paper described how a tool for test has been embedded as a plug-in into Eclipse Java Development Environment.

This allowed increasing the power of the Unit Test providing an easy way for scaffolding data and for simulating realistic environment situations, without modifying the Java application code and without switching off from the development environment.

Adopting this solution we were able to execute a set of complex tests as a routine starting from the first development phases, with a sensible reduction of costs in terms of fixing.

A future possible extension of the use of JITAT plug-in should be a further integration with JUnit.

The 'Open Points' section has promising ideas about further improvements of the plug-in.

References

1. Mauro Arcese, Design for Testability put into Practice, Supplementary Proceedings of the 17th IEEE International Symposium on Software Reliability Engineering (ISSRE06), November 2006.
2. Marcel Christian Baur, Instrumenting Java Bytecode to Replay Execution Traces of Multithreaded Programs. (http://research.nii.ac.jp/~cartho/papers/mbaur-2003-instr.pdf)
3. Joseph Coha, Byte-code Instrumentation Revealed. (http://bdn.borland.com/borcon2004files/32628/32628_09221213_S.PPT)
4. Dennis Sosnoski, the "Java programming dynamics" series on developerWorks. (http://www-106.ibm.com/developerworks/java/library/j-dyn0429/)