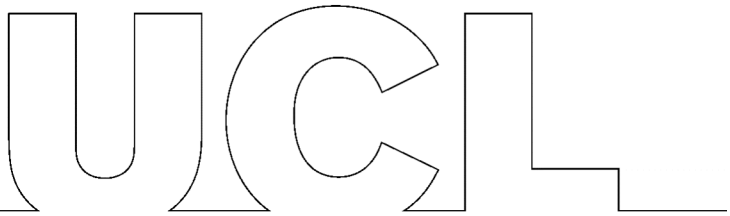
UCL DEPARTMENT OF COMPUTER SCIENCE

Wolfgang Emmerich, Dr rer-nat, CEng, MIEE
Professor of Distributed Computing
Director of Research



GS04: Tools and Environments

Lab Session 9: Mocking Objects with JMock

The aim of this lab session is to improve your unit testing knowledge further by being able to write unit tests of complex classes that depend on other classes. In order to test such classes in isolation you need to replace these other classes. We will use JMock for this purpose. We will unit test classes of the supervisor/worker architecture used in Chapter 11 of Magee&Kramer. Concurrency – From State Models to Java Programs (2nd edition). Wiley 2007. You may review an animation of the task that the architecture performs at http://www.doc.ic.ac.uk/~jnm/book/book_applets/SupervisorWorker.html. As you can see, the particular application uses a supervisor/worker architecture for numeric integration of different functions.

Setup

Check out the relevant interface and class source code for the project from folder JMockLab/trunk in my SVN repository. There are a number of interfaces involved in this application. The two principal components are ISupervisor and IWorker. ITupleSpace is an interface for a tuple space component by means of which supervisors and workers communicate asynchronously. IFunction is the interface of the function that we want to integrate numerically. Implementations of supervisor are expected to call display() of IDisplay once the calculation is complete and show how large the integral was. We only have the implementation of ISupervisor in class Supervisor and class Result (which is essentially part of the interface between ISupervisor and ITupleSpace).

Testing with JUnit 4 and JMock

Using this project you can now explore the use of JMock with JUnit 4. In JUnit 4 test cases can be defined using annotations and you need no longer follow the naming conventions of JUnit 3. Check out the respective documentations at http://junit.sourceforge.net/doc and http://www.jmock.org

You want to test that the Supervisor creates the correct number of tasks for the workers, depending on how many slices for the numeric integration you instruct the supervisor to create. You therefore decide to create different tests for several different numbers of slices.

Moreover you want to check that the results that the workers return are correctly processed and added up by the supervisor before calling the display method.

You want to test these criteria without having or using the implementations of IDisplay or ITupleSpace and instead you use JMock to derive mock objects from these interface definitions.

University College London Gower Street London WC1E 6BT
Tel: +44 (0)20 7679 4413 (Direct Dial) Fax: +44 (0)20 7387 1397
www.cs.ucl.ac.uk/staff/W.Emmerich
w.emmerich@cs.ucl.ac.uk