Profiling and Testing Applications Using the Eclipse Test and Performance Tools Platform (TPTP)

Eugene Chan, IBM Tivoli Software
Valentina Popescu, IBM Tivoli Software
Paul Slauenwhite, IBM Rational Software
Agenda

- Overview of TPTP
  - TPTP Profiling tools
  - TPTP Testing tools
  - What’s new?
Overview - Eclipse TPTP

- Eclipse top level project
  - 2002: Eclipse tools subproject - Hyades
  - 2004: Eclipse top-level project - TPTP

Mission:
- To build a generic, extensible, standards-based platform for test and performance tracing tools.

Goals:
- Platform of choice for test, performance, and monitoring tools.
- Exemplary tooling.
- Enable value-added third-party tooling through extensibility and high-quality APIs.
Overview - TPTP project structure

- Comprised of four projects
  - Platform
  - Test
  - Trace
  - Monitoring

- Developed by a number of strategic contributors
  - Active contributors: Intel, IBM, OC Systems
  - Inactive contributors: Scapa Technologies, Computer Associates, Compuware, FOKUS

- For more information visit [http://www.eclipse.org/tptp/](http://www.eclipse.org/tptp/)
Overview - TPTP framework

- TPTP offers a common, extendable framework as well as reference implementations for the following functions:
  - Testing
  - Profiling
  - Static code analysis
  - Static and dynamic Bytecode Instrumentation (BCI)
  - Application monitoring and log analysis
Overview - TPTP Test tools

- Common framework for testing tools thereby integrating disparate test types and execution environments.
  - Common perspectives and views for interacting with target systems and resources.
  - Reference navigators, viewers, editors and wizards through extension points.
  - Standard EMF data model, query framework and assets repository.
  - Common data collection and execution framework on local and remote targets.

- Reference implementations for the TPTP Test framework:
  - Manual testing.
  - JUnit and JUnit Plug-in testing.
  - URL testing.
  - Automated GUI Recording/Playback.
Overview - TPTP Profile tools

- Common framework for profiling simple Java applications or complex Web applications running on multiple platforms, on different hosts.
  - User interface actions to interact with the profiled application.
    - Start, stop, attach, detach from the profiled application.
  - Views framework used to analyze the profiled application.
    - Views can be extended and customized using extension points.

- Reference implementation of the Profiling framework:
  - Java Profiling tools based on JVMPI and JVMTI.
  - Actions to interact with the profile application: run garbage collection, collect object references, filter and sort data
  - Views to analyze collected data: performance, memory analysis, thread views

- Probekit, a generic tool for probe creation and insertion.
Demo

- Eclipse TPTP:
  - 4.4.0.3 all-in-one
  - API Recorder Technology Preview

- Demo Java Application:
  - Simple store
Agenda

✓ Overview of TPTP
➢ TPTP Profiling tools
☐ TPTP Testing tools
☐ What’s new?
Profiling Tools

- Profiling Perspective
Profiling Tools

➢ Start a profiling session…
Profiling Tools

- Profiling Filter

![Diagram of Eclipse Profiling Tools with selected filter options and classes for profiling.]
Profiling Tools

- Profiling Option – Memory Analysis
Profiling Tools

- Open Profiling View
Profiling Tools

Profiling View - Object Allocations view
Profiling Tools

- Profiling View – Object Allocation View
Profiling Tools

- Profiling – Object Allocation
Profiling Tools

- Profiling Option – Execution Time Analysis
Profiling Tools

- Profiling View – Execution Statistic View
Profiling Tools

- Profiling View – Execution Statistics View
Profiling Tools

- Profiling View – Open Source Action
Profiling Tools

- Profiling – Execution Analysis
Profiling Tools

- Profiling Option – Thread Analysis
Profiling Tool

- Profiling View – Thread Analysis View
Profiling Tool

- Profiling View – Call Stack
Profiling Tool

- Profiling Views – Toolbar Actions
Profiling Tool

- Profiling On Sever
  - Simplify web application profiling.
    - A pre-configured shortcut to launch and attach to web application for profile.
      - Context menu
      - Servers view
Profiling Tool

- Help > Help Contents

Tutorial: Analyzing profiling data

Objectives
To analyze the data collected from profiling a Java process, using the Profiling and Logging views of the Profiling and Logging perspective in the following ways:
- Identify objects and methods that consume the most time
- Identify memory-intensive classes
- Gauge program concurrency
- Locate memory leaks
- Browse every execution of a method as a function of time
- Get a wider view of execution behavior as a function of time
- Identify active threads
- Identify when threads are active
- Identify frequently called methods
- Gauge garbage collection
- Identify different phases of program execution
- Study different method invocations
- Study the caller of a method

Time required
1 hour

Before you begin
Before you start this tutorial, you need to:
1. Install the Eclipse Platform, including the Test and Performance Tools Platform (TPTP).
2. Have the sample profiling file (extract.jar) ready for import. The file should be in the following directory:
   TPTP_install_dir\eclipse\plugins\org.eclipse.tptp.platform.collection.framework_4.0.0

Description
In this tutorial, you will use the different Profiling and Logging views to analyze the sample PerformanceExample class. The sample opens a frame containing two buttons.
You will complete the following tasks:
Agenda

✓ Overview of TPTP
✓ TPTP Profiling tools
➢ TPTP Testing tools
☐ What’s new?
Testing Tools – Basic Concepts

- Test Perspective
  - Set of navigators, viewers, editors and wizards for tests and test assets.

- Test Suite
  - Consists of test cases and behaviors.
  - Created manually with the test suite editor or automatically by a recorder.
  - May be associated with a test script or Java class.
  - Executable with TPTP test execution framework.

- Datapool
  - Provides input and expected output data to a test.
  - Consists of equivalence classes, variables and records.
  - Created with the datapool editor or imported from *.CSV files.
Testing Tools – Basic Concepts

- Test Execution
  - Framework for local and remote test deployment, execution, results collection.
  - Typically started from Launch Configuration.
  - Automatable Services Framework (ASF) for launching tests programmatically via scripts (e.g. shell and ANT) and external applications (e.g. Java).

- Test Log
  - Persisted execution results.
  - Test log viewer to summarize, view, navigate and filter execution events, and associated defects.

- Report
  - Aggregates and summarizes numerous test execution results over a period of time (report window).
  - BIRT integration and extensible report generators.
Testing Tools – Test Types

- **Manual**
  - Manual testing of applications by human testers based on textual test descriptions (plain text or HTML).
  - Manual Test View for executing a manual test on the target machine allowing the tester to step through the manual test and capture messages and results:
    - Verdicts
    - Reasons
    - Information
    - Attachments

- **JUnit and JUnit Plug-in**
  - Automated unit testing of applications and plug-ins based on JUnit test classes.
  - JUnit test classes generated and synchronized automatically from the test suite while preserving user modifications.
  - Wizard for creating TPTP JUnit and JUnit plug-in tests from existing JUnit and JUnit plug-in test classes without source modification.
Testing Tools – Test Types

- **URL**
  - Automated HTTP performance testing of Web applications from JUnit test classes.
  - HTTP proxy recorder to intercept and record HTTP requests/responses from user interactions with Web applications to create an URL test suite.

- **Automated GUI Recorder (AGR)**
  - Automated GUI recording and playback for testing Eclipse-based UIs.
  - Positional and object-based GUI recorder to intercept and record user interactions with UI widgets in the workbench to create an automated GUI test suite with datapools and verification points.
  - Object mine and adaptive widget resolution to help maintain tests.
Testing Tools – API Recorder

- Records the behavior of local and remote Java applications.

- Captures API invocations and their arguments/return values based on user-defined package, class, and method targets.

- Recordings are used for analysis, testing (e.g. datapools), and generating TPTP JUnit tests.
Testing Tools - Demo

- Target API
Testing Tools - Demo

- Start an API Recording...
Testing Tools - Demo

- API Recording Configuration – Test Suite Generator
Testing Tools - Demo

- API Recording Configuration – Target API
Testing Tools - Demo

JUnit Test Suite - Overview
Testing Tools - Demo

JUnit Test Suite – Test Methods
Testing Tools - Demo

JUnit Test Suite – Behavior (invocation)
Testing Tools - Demo

JUnit Test Suite – Behavior (loop)
Testing Tools - Demo

JUnit Test Suite – Test Class
Testing Tools - Demo

- Run the JUnit Test Suite...
Testing Tools - Demo

➢ Test Log - Overview
Testing Tools - Demo

- Test Log - Events
Testing Tools - Demo

- Create a Test Report...
Testing Tools - Demo

- BIRT Test Pass Report
Testing Tools - Demo

Create a Datapool…
Testing Tools - Demo

- Datapools - Overview
Testing Tools - Demo

- Datapools – Equivalence Class
Testing Tools - Demo

- Using the Datapool

```java
/**
 * demo_shoppers_Customer_updateName
 */
TFTP JUnit test case 'demo_shoppers_Customer_updateName' to test the 'java.lang.String demo.shoppers.Customer updateName'
@throws Exception
*/

public void demo_shoppers_Customer_updateName() throws Exception {
    Class targetClass = ClassLoader.getSystemClassLoader()
        .loadClass("demo.shoppers.Customer");
    Method targetMethod = targetClass.getMethod("updateName",
        new Class[]{java.lang.String.class});

    // Traverse through datapool:
    while (!datapoolIterator.dpDone()) {
        if(targetMethod.getModifiers() == Modifier.STATIC)
            assertEquals(datapoolIterator.dpCurrent().getCell("returnValue"),
                String.valueOf(targetMethod.invoke(null, new Object[]{
                    datapoolIterator.dpCurrent().getCell("parameter")},
                }));
    else
        assertEquals(datapoolIterator.dpCurrent().getCell("returnValue"),
            String.valueOf(targetMethod.invoke(targetClass.getConstructor(
                new Class[]{}, new Object[]{
                datapoolIterator.dpCurrent().getCell("parameter")},
            }));

    datapoolIterator.dpNext();
}
```
Testing Tools - Demo

Help > Help Contents
Agenda

✓ Overview of TPTP
✓ TPTP Profiling tools
✓ TPTP Testing tools
➢ What’s new?
What’s New in TPTP 4.5

Profiling Tools:
- Secure profiling (JVMTI) including authentication and encryption.
- Simplified stand-alone profiling (JVMTI).
- Thread contention analysis for locating monitor and data contention.
- Improved profiling filters (filter sets).
- Binary data format to increase profiler performance and scalability.
- Support for Java 6 and IPv6 networks.

Test Tools:
- Encrypted datapool variables for testing with confidential data.
- Parallel test execution for launching multiple tests concurrently.
- Reference maintenance when moving, copying, deleting, renaming, and importing/exporting test assets.
- URL recording on IPv6 networks.
References

- TPTP home page
- TPTP download page
- TPTP documentation
- TPTP tutorials and demos
- TPTP Eclipse corner article
THANK YOU