ARM-BASED PERFORMANCE MONITORING
FOR THE ECLIPSE PLATFORM

Ashish Patel, Lead Eclipse Committer for ARM, IBM Corporation
Oliver E. Cole, President, OC Systems, Inc.

The Eclipse Test and Performance Tools Platform (TPTP) project released an open source and ARM 4.0 compliant implementation in June 2006, based on TPTP version 4.2. This paper describes the TPTP project, and explains how ARM has been implemented and integrated with TPTP. The tradeoffs involved in getting to this point are discussed, along with the challenges in moving forward. Specific examples show how to get started using ARM for your application with TPTP.

1. Introduction

This paper describes how the Eclipse Test and Performance Tools Platform (TPTP) project is extending its toolset to use The Open Group Application Response Measurement (ARM) instrumentation methodology for response-time tracking. Starting with TPTP version 4.2, performance analysts will be able to use Eclipse TPTP—or products based on TPTP technology—to identify and monitor individual transactions as they pass through the various components of composite applications.

The implementation of ARM described in this paper was released as a Technology Preview with TPTP 4.2 in June 2006. In the future, it will be fully integrated into TPTP.

1.1. About Eclipse

Eclipse1 is an open source community whose projects are focused on providing a vendor-neutral open development platform and application frameworks for building software. The Eclipse ecosystem is built on royalty-free technology that offers a universal platform for tools integration.

All technology and source code provided to and developed by Eclipse is made available royalty-free via the Eclipse Public License. As a result, tools vendors can and are integrating Eclipse technology into their products.

The Eclipse Platform is written in the Java language and comes with extensive plug-in construction toolkits and examples. It has already been deployed on a range of development workstations including Linux, HP-UX, AIX, Solaris, Mac OS X and Windows based systems.

The Eclipse Test and Performance Tools Platform2 (TPTP) is one of many Eclipse projects. TPTP provides an open platform that allows software developers to build unique test and performance tools, both open source and commercial, that can be easily integrated with the platform and with other tools.

TPTP addresses the entire test and performance life cycle, from early testing to production application monitoring, including test editing and execution, monitoring, tracing and profiling, and log analysis capabilities. The platform supports a broad spectrum of computing systems, including enterprise, high-performance, standalone and embedded systems.

The Tracing and Profiling Tools Project is a subproject of TPTP that provides a framework for building tools that collect and analyze application performance information. The project includes tools that collect trace and profile data for single-system Java applications through the JVMPI monitoring agent. It also includes a toolkit called Build-to-Manage (BtM), a generic toolkit for instrumentation using Application Response Measurement (ARM), Java Management Extensions (JMX), and Common Base Events (CBE).

Version 4.2 of Eclipse TPTP implements a version of the Application Response Measurement3 (ARM) specification, which can be used to instrument specific points in an application and collect trace and profile data for a distributed Java application while delivering more accurate profiling measurements.

1 See http://www.eclipse.org.

2 See http://www.eclipse.org/tptp/.

3 See http://www.opengroup.org/ARM/.
1.2. About ARM

The Application Response Measurement (ARM) standard is developed by The Open Group, which is a vendor- and technology-neutral consortium that strives to enable access to integrated information within and between enterprises based on open standards and global interoperability.

The ARM standard describes a common method for integrating enterprise applications as manageable entities. The ARM standard allows users to extend their enterprise management tools directly to applications, creating a comprehensive end-to-end management capability that includes measuring application availability, application performance, application usage, and end-to-end transaction response time.

Applications make calls to Application Programming Interfaces (APIs) defined by The Open Group when transactions begin and end, allowing these transactions to be measured and monitored. This information can be used to support service level agreements and to analyze response time across distributed systems.

ARM instrumentation has grown in popularity since it was first introduced in 1996 and is now built into software from leading vendors such as IBM, Hewlett-Packard, SAS and Siebel. Software that is not already ARM-enabled can be made to have calls to the ARM API embedded directly in the source code, or calls can be inserted into the machine code at run-time.

2. ARM Architecture in Eclipse TPTP

Using the ARM implementation in TPTP, a user can take any Java application and use TPTP to instrument ARM calls into application source code, collect the ARM data while the application is executing, and then view the collected data with graphical viewers.

Figure 1 shows a block diagram of the TPTP ARM Data Collection Infrastructure (TADCI), which consists of the following components:

2.1. TPTP ARM Library

The TPTP ARMv4 Library is an implementation of The Open Group ARM version 4 standard. Instrumented applications make calls to the Standard ARMv4 interface, which invokes the TPTP ARMv4 Library and reports events to the TPTP ARM Engine.

2.2. TPTP ARM Engine

The TPTP ARM Engine processes calls made to the Standard ARMv4 interface and produces event-level information that outputs an ARM event for each event.
during a transaction.

The TPTP ARM Engine communicates to a TPTP Agent, which utilizes the Agent Controller (AC) for communication. The output from the TPTP Agent is in an XML format, which is a human-readable, open document format that can be processed by the TPTP Event Loaders.

In order to impact the performance of the application as little as possible, the ARM Engine is designed to operate on a separate thread in the same process as the application’s Java Virtual Machine (JVM). Running the ARM Engine in a separate JVM process could have skewed or misled an analyst when viewing performance results since the JVM itself is executing the application under investigation. In addition, this design approach allows the application to manage ARM Engine’s lifecycle.

2.3. TPTP Workbench

The TPTP Workbench contains user interface tools to:

(1) Launch the application being profiled,
(2) Byte-code instrument applications to be profiled, and
(3) Deploy Probekit source files to systems where the application is executed.

The user interface allows the application to be instrumented with a user-defined filter. The filter is applied at the instrumentation level, such that only those points in the application that match the filter’s pattern are byte-code instrumented.

2.4. TPTP Agent Controller

The TPTP Agent and TPTP Agent Controller (AC) provide the communication mechanism between the ARM Engine and the TPTP Workbench.

Data is reported from the ARM Engine to the TPTP Agent, which transfers it to the AC. In turn, the AC forwards the data to the TPTP Workbench. Though this data flow sounds complicated, the TPTP Agent and AC are transparent to users of the TPTP ARM Engine.

2.5. Probekit Runtime

Instrumentation of the application is accomplished with the Probekit Runtime, which accepts compiled Probekit files and uses byte-code instrumentation techniques to instrument the application.

The runtime is supplied with a “pre-packaged” Probe Library, which contains a Generic ARM probe that will instrument the entry and exit site of a method. All methods that pass a user-defined filter will be instrumented using this probe.

The following items are bundled with the existing AC, which already includes the Probekit Runtime:

- J2EE Probekit Library (also known as the ARMv4 Instrumentation “pre-packaged” Probe Library)
- TPTP ARM Engine
- TPTP ARMv4 Library

2.6. Distributed Environments

When monitoring a single or distributed application, the user must install and configure the TPTP AC and TADCI on every machine where the application executes.

The TPTP ARM Engine provides the functionality for dynamically discovering systems involved in a business transaction. Once a system is discovered, the ARM Engine will automatically have the TPTP Workbench attach to the discovered system. Figure 2 illustrates the TADCI in a distributed environment.

In order to correlate transactions that occur on different processes and machines in a distributed environment, the ARM standard calls for the use of an ARM correlator. A correlator is generated for every root (or edge) transaction and each of its child transactions. A business transaction is determined by building a tree using these correlators. This process allows the ARM Engine to trace the path of a distributed transaction through the infrastructure.

In order to transport the ARM correlator across different application servers in a distributed environment, the instrumentation flows the correlator as part of a CORBA Portable Interceptor when a distributed call is executed. Portable Interceptors are objects that an ORB invokes in the path of an operation invocation to monitor or modify the behavior of the invocation transparently. For our purposes, a Portable Request Interceptor is created and used to marshal and de-marshal the correlator during distributed transactions. The user is required to install the request interceptor in his or her application server.

---

4 Dr. Dobb’s CORBA Metaprogramming Mechanisms: http://www.ddj.com/dept/cpp/184403860
3. **Steps to ARM Instrument an Application**

There are three basic steps involved in using ARM on a given application with Eclipse TPTP:

1. Use byte-code instrumentation to insert the ARM calls into the application,
2. Execute the instrumented application, and
3. View and analyze the results.

### 3.1. Instrumentation

The first step to instrument an application requires inserting calls to the ARM API at strategic points in the application code. The Build-to-Manage (BtM) Toolkit for Java Instrumentation provides a wizard that developers can use to add instrumentation to selected packages, classes and methods. The toolkit, originally developed by IBM Tivoli, has been released to open source with Eclipse TPTP version 4.2. The toolkit offers two ways to add the instrumentation: Probekit and AspectJ.

#### 3.1.1. Probekit

Probekit[^1] is a scriptable, byte-code instrumentation (BCI) framework that developers can use to write Java code fragments that are inserted into an application to provide information about the program as it runs. This method instruments the classes of the actual application, rather than the application's container.

Probekit works well with the Eclipse TPTP framework when the application under test is started in controlled mode. Controlled mode halts the execution of the application until a client attaches to the profiling agent. Starting an application server in this mode instantiates the applications in controlled mode automatically, eliminating the need to restart the application server after its applications have been instrumented.

This mode allows the AC to monitor the application server without the TPTP Agent commencing processing until the Eclipse Workbench is instructed to monitor some aspect of the application, and therefore, having the TPTP Agent loaded will not affect the application server's performance significantly.

Once instrumentation probes are deployed to the AC the probes will be applied to the classes that match the user-defined filter specification as they load in the JVM.

A user can create multiple probes and deploy any combination of these defined probes to the Probekit Runtime. This instrumentation mechanism uses the Class Load hook in JVMTI for dynamic instrumentation of applications.

Today, there is no mechanism to “undeploy” a probe. Therefore, once a set of probes is applied, the user

---


---

![Figure 2 - TPTP ARM Data Collection Infrastructure (TADCI) for a Distributed Environment](image-url)
must restart the application server because data from these probes is applied to the JVMPI agent and the Probekit Runtime. Users are required to utilize the "start" and "stop" buttons on the user interface to control these. However, research is ongoing, and future releases of TPTP may allow probed classes to be "undeployed." This behavior could be as simple as notifying the Probekit Runtime not to instrument new instances of the instrumented classes in the application when they load, and hence, one can force the JVM to redefine the class (in effect "undeploying" the probes).

3.1.1.1. Static Instrumentation

Probekit (as of TPTP 4.1) supports static instrumentation. During static instrumentation, Probekit rewrites the Java classes that are instrumented by adding invocation to the probe classes, which are added to the project. The instrumented application can run on any JVM and there is no need for a Probekit runtime. All that is required is to have the probe classes available and in the classpath of the program.

3.1.1.2. Dynamic Instrumentation

Probekit (as of TPTP 4.2) supports dynamic instrumentation. Dynamic instrumentation works similar to static instrumentation, but it performs the instrumentation in memory at class load time instead of rewriting the class file on disk. The inserted byte codes are the same as in the static instrumentation case.

There are some consequences of using dynamic instrumentation. The called methods are loaded by the boot class loader, and therefore, in the instrumentation fragments one cannot refer to classes that are loaded by higher-level class loaders.

3.1.2. AspectJ

AspectJ is an aspect-oriented programming (AOP) implementation for Java in Eclipse. AOP allows programmers to create special-purpose modules, called aspects. An aspect simplifies the problem of adding consistent behavior to various locations in application modules; the set of points at which an aspect is applied is called a point cut; advice is the code invoked at these points. Without AOP, code implementing a specific purpose (such as instrumentation of code) is scattered throughout application modules, making it difficult to add, remove, or modify.

The user then defines a point cut of method entries and exits. When the aspect is applied to the application at start, the resulting ARM calls profile the application.

Probekit and AspectJ instrumentation mechanisms work very similarly. The only difference is that during instrumentation Probekit uses a probe file while AspectJ uses an aspect file. The aspect file (aop.xml) defines several point cuts, or locations in the application code where instrumentation is to be injected. An example of a point cut is:

```xml
<pointcut
    expression="
    execution(* com.ibm.ui.Client.*(..)) ||
    execution(* com.ibm.core.Library.*(..))"
    name="monitoredOperation" />
```

3.1.3. Using the Build-to-Manage Toolkit

To use the Build-to-Manage Toolkit (BtM), programmers first install Eclipse TPTP. Support for ARM instrumentation is in TPTP Trace and Profiling Tools Project. BtM also supports Java Management Extensions (JMX) and Common Base Events (CBE), both of which are in TPTP Monitoring Tools Project.

BtM uses a preference page to determine whether to instrument with AspectJ or Probekit. At install, if the AspectJ Development Tools plug-in is present, the preference is set for AspectJ. Otherwise, the preference is set for Probekit.

The BtM wizard is invoked through the Run-Profile menu. The toolkit offers the choice of instrumenting with ARM, JMX or CBE. To use ARM, the programmer uses the ARM Analysis Type when profiling the application.
3.2 Run the Instrumented Application

To run the ARM instrumented application, programmers simply use the Profile button in Eclipse. Then they create a profile launch configuration, the same way they do in TPTP today. The following sample shows how to profile a "Hello World" application.

Figure 3 - Here is the sample “Hello world” application.

Figure 4 - Open the Profile Launch configuration and create a configuration for the sample Java application.
Figure 5 - Select the Instrument Collector as the data collector for profiling the application.

Figure 6 - Click 'Edit Options' on the Instrument Collector to configure the options.

In the Instrument Collector, filters can be added or removed to help the user determine what packages, classes, and/or methods should be included or excluded during instrumentation. You must add a filter to include the package(s) in which your application resides.
Figure 7 - Check off the ARM analysis type.

Figure 8 - Click 'edit options' on the analysis type and pick the TPTP ARM Engine (it is selected by default).

Then click Profile and run the application.
3.3 View and Analyze collected ARM data

The ARM data is stored in the TPTP trace data model, where it is available for viewing using any of the standard or custom TPTP views. The first two views shown here are from the “Hello World” application.

Figure 9 - The Execution Statistics view shows the console, the application code, and statistics about the execution time such as the number of methods called and the amount of time taken to execute every method.

Figure 10 - The UML2 Trace Interaction view shows the console, the application code, and the UML class interactions.
The next three figures show views from a more complex application based on an automated teller machine (ATM). Other views that are standard in TPTP include Profiling Monitor, Code Coverage Statistics, and Execution Flow.

Figure 11 – The Execution Statistics view shows statistics at the package, class, method and instance level.

Figure 12 – The UML2 Trace Interaction view shows the beginning of the ATM transaction.
Figure 13 – This UML2 Trace Interaction view shows the actual bottleneck in the application; in this case, the call to the checkCustomer method took 6 seconds.

Figure 14 – The Method Invocation Details view shows a representation of the entire course of a program’s execution.
4. Future Direction

Currently, future plans include adding enhanced types of instrumentation to the Probekit Library to support such technologies as Web services and native applications.

We are looking to provide a more elaborate probe library, specifically for J2EE applications. The extended library could include Servlet, Enterprise Java Beans (EJBs), Remote Method Invocation (RMI), Java Database Connector (JDBC), and Java Web Services.

Other enhancements will include support for instrumentation to work with the application servers currently supported by the Eclipse Web Tools Project (WTP), including IBM WebSphere, Apache Tomcat, JBox, JOnAS, and Microsoft IIS. In addition, we look to capture the SQL statements executed during a business transaction on various databases, such as IBM DB2, Cloudscape, and MySQL.

While Java and J2EE are leading technologies for application development, there are other technologies which are in widespread use, notably .NET (Visual Basic.NET or C#) and scripting languages (PHP, Perl, Ruby on Rails). Support for those technologies could be added.

A particularly important issue is that the ARM 4.0 Standard does not define the format of the correlator. This means that ARM implementations are not interoperable. As more ARM implementations arrive, we can expect this issue to start to gain prominence.

Eclipse TPTP is an open source effort, so if you are interested in contributing, please contact us.

5. Getting Started

This section describes step by step where to get the current version of the TPTP ARM capability, how to download and install it, and how to get ARM data from a demonstration application.

Download and Install

To download and install Eclipse TPTP and the Tech Preview with the ARM instrumentation, follow these steps:

2. Click the Download link.
3. Select 4.2 from the tabs at the top of the page.
4. Select a Stable Build of the version.
5. Download the following components:
   a. TPTP Runtime (SDK is optional),
   b. Agent Controller,
   c. Technology Preview Section: ARM Instrumentation (ARM Engine and ARM UI),
   d. All required components (including WTP, GEF, and JEM).
6. Install the workbench:
   a. Unzip the TPTP Runtime, ARM UI and all required components into the same folder.
   b. Run
      \bin\SetConfig.bat
      (or equivalent).
   c. Unzip the ARM Engine into the same folder as the agent controller.


Once TPTP and the Tech Preview are installed, the user is ready to begin using ARM. The software to run this demonstration application is available for download from the OC Systems website at http://www.ocsystems.com/eclipse/armtptp.html. A Flash version of the demo is also available at the same location.