

# Designing, Modelling and Implementing a Toolkit for Aspect-oriented Tracing (TAST)

Tina Low  
Siemens Corporate Technology Software and Engineering Department  
Otto - Hahn - Ring 6  
81730 Munich, Germany  
Email: Tina.Low@mchp.siemens.de

## Abstract

Monitoring and diagnostic tools are essential for successful development and operation of complex software systems. The Monitoring and Performance Engineering team at the Siemens Corporate Technology department CT SE 1 developed a tool called TMT (Test and Monitoring Tool) that visualizes so-called traces, a sequence of designated application events. In order to generate trace information, probes that intercept and log application events have to be introduced in the System Under Test. This instrumentation can be done manually, but this "cut and paste" approach is work intensive and error prone. As tracing is a typical "crosscutting concern", we used AspectJ to develop the instrumentation aid TAST for Java applications.

TAST encapsulates all tracing-related issues (incl. the generation and transmission of unique identifiers that are necessary to correlate corresponding trace events). It offers many advantages over manual instrumentation with respect to maintainance, documentation, consistency, completeness and reusability.

In our demonstration at the AOSD 2002 we show how an application is instrumented with AspectJ. The audience will be able to watch the running application producing traces that are visualized by TMT and get an insight in the visualization and analysis functionality of TMT.

For more information on TMT see:  
<http://java.sun.com/products/jfc/tsc/sightings/S04.html>

## Status: Position paper

The development of TAST is one of the „prefield“ activities within the Siemens Corporate Technology Department.

## Introduction

The complexity of modern software systems introduces challenging requirements for testing, deployment and maintenance. Monitoring and diagnostic tools are essential for successful deployment of these systems. The Monitoring and Performance Engineering team at the Siemens Corporate Technology department SE 1 developed a tool called TMT (Test and Monitoring Tool) that visualizes so-called traces, a sequence of designated application events for use by developers, quality testers, system integrators and application managers see [TMT 00].

In order to generate trace information, probes that intercept and log application events have to be introduced in the System Under Test. This step is called instrumentation. It can be done manually or – even better – with tool support. This tool support is realized by TAST.

In this paper we present some goals of the instrumentation aid TAST for Java applications, the instrumentation aspects and inherent design issues.

We are proud to present the visualization of the processed traces using the TMT monitoring tool. The trace format used in TAST is equivalent to the TMT trace Format, see [TMT 00].

## Goals

We developed the instrumentation aid TAST for Java applications, which is based on Aspect Oriented Programming and AspectJ (see [AspectJ]). The trace points in the System Under Test are formulated in „pointcuts“ and the instrumentation calls themselves are encapsulated in „advices“. We want to be able to trace 1-process applications as well as distributed applications.

One of the main requirements of TAST is to automate the instrumentation process and to support the identification of application elements (devices, processes, objects) with unique Ids in a distributed java application. The separation of tracing concerns from the System Under Test during the development phase and during the Test phase is one of the advantages of AspectJ.

Each method call in a java system is represented by an trapezium view in the visualisation tool TMT. One of the most important TAST requirements shall be monitoring RPC calls (remote method invocation) over JVM borders.

## ***Reasons for using AspectJ for this instrumentation efforts***

The AspectJ approach offers several advantages:

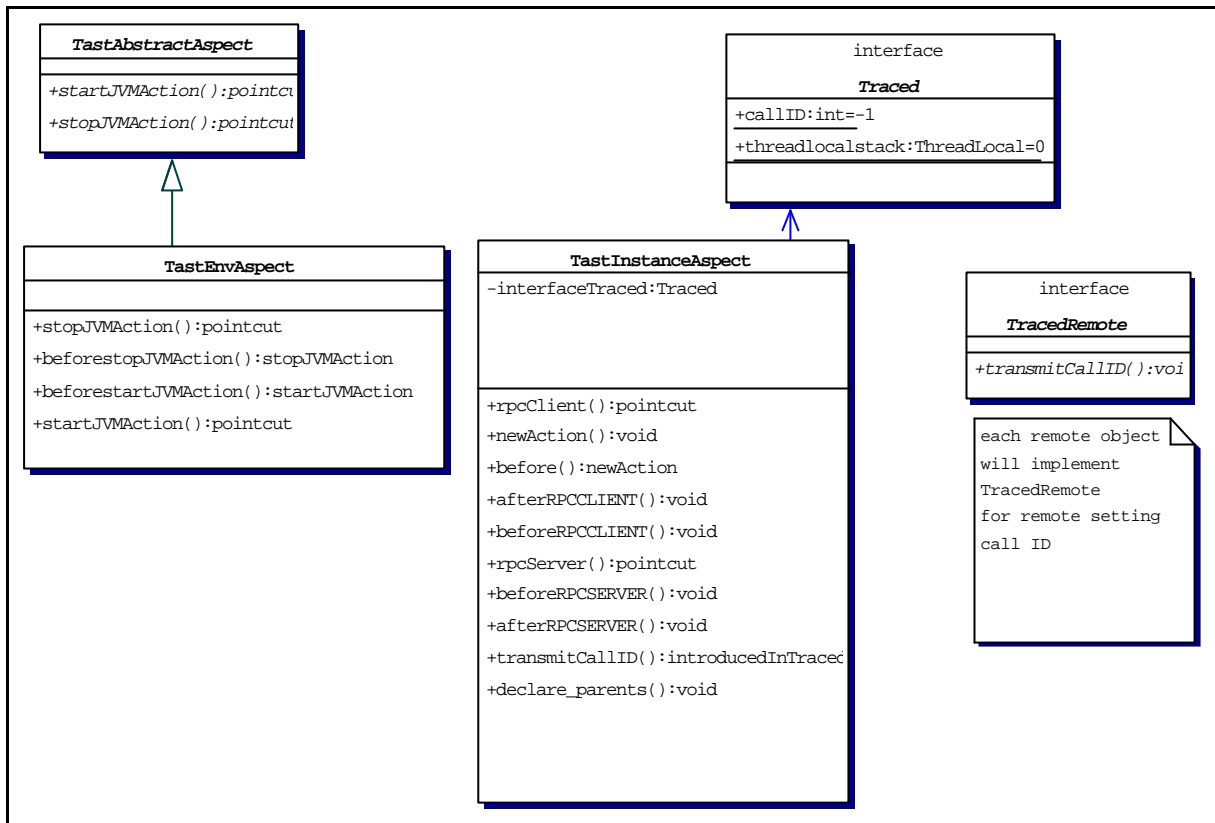
- The instrumentation is implemented in Aspects, i.e. it is separated from the application source code. Therefore the instrumentation can be easily changed and documented without affecting the original source code.
- The instrumentation Aspects solve typical tracing problems, particularly the generation and transmission of unique identifiers that are necessary to correlate corresponding client/server side trace events.
- The instrumentation Aspects are reusable for other applications.
- The instrumentation Aspects help the user to define specific trace points, i.e. what he/she wants to instrument (packages, classes, methods). Different application events (object creation, thread activity, method calls...) may be instrumented.
- As opposed to manual instrumentation which is highly error-prone, hard to control and may be incomplete, the instrumentation Aspects guarantee that exactly those application events which fulfill the criteria specified by the user are captured.
- The AspectJ compiler automatically inserts instrumentation calls at many locations of the source code thus rendering the work of manual instrumentation unnecessary.
- Instrumentation may be entrusted to one developer. This makes it much easier to fulfill instrumentation guidelines consistently. The instrumentation concept and API need not be the concern of every developer on the team.

## ***Design Issues***

The identification of concerns and sorting out which will be encapsulated and which will not is an important part of designing that leads to the decision of when and how to separate what concern. Most likely there will be a tradeoff between simplicity of individual concerns (which improves as you get better separation) and complexity of large numbers of concerns (numbers increase as you get more separation).

We developed TAST using the JBuilder4.0 Foundation Edition. TogetherJ was used in order to have a presentation of the TAST aspects realized while programming. Currently TogetherJ is able to design a java system or reverse engineer the software system's java classes. This java system is the one we want to instrument using the TAST aspects' advices.

At this stage TogetherJ can not model the aspects we develop. Instead we made a simple but adequate view of the aspects using the class diagram of the modelling tool. Figure 1 shows an UML model of the aspects.



**Figure 1. Simulated UML View of TAST aspects**

Tast separates the Tracing concerns in 2 aspects.

The TastEnvAspect recognizes the devices and processes of the java application when the main routine is executed.

The TastInstanceAspect realizes the object creation, method calls and method executions. Currently the classes and methods to be traced are placed in the aspect code in this aspect. These pointcuts will be interwoven with to the application in the production process.

As mentioned above we need unique identifiers for each object, process and method call. The Ids are served by an UUIDGenerator class.

In order to send an unique identifier (called „call\_ID“) to the corresponding method execution (over JVM borders) we used the introduction possibility to insert the Id in the corresponding server object, which is executing the method.

Figure 2 shows a TMTs List View extract. There you can see the correct callID transmitted from client object to server object.

Philo[20]	RPC Client Begin: invocation of "getLeftFork", rpcId=-1818173420
PhiloServerImpl[4]	RPC Server Begin: invocation of "getLeftFork" called by "Unknown Task[-1]", rpcId=-1818173420
PhiloServerImpl[4]	RPC Server End: method end "getLeftFork" called by "Unknown Task[-1]", Returns: getLeftFork, rpcId=-1818173420
Philo[20]	RPC Client End: method end "getLeftFork", answer from "Unknown Service_Provider[-1]", Returns: getLeftFork, rpcId=-1818173420

**Figure 2. TMTs list view – a remote call**

## ***The Production Process***

An application, as we have discussed, is instrumented with Aspects.

We explain the main parts in the build process of a Java application which is using RMI for remote method calls. This „Dining Philosophers“ application is a widely known distributed RMI application. It consists of 5 clients and 1 server, where the clients compete to take 2 forks for eating. The application is an distributed example of java applications, where deadlocks can occur. This „Dining Philosophers“ application is described in [Magee 99].

### **Procedure:**

- 1) Compile your application with standard Java 1.3 to ensure that classes, which aren't to be woven have their bytecode processed.
- 2) Weave the application together with the TAST aspects (in order to introduce the tracing points as well as the call ID and its methods).

In the sample philosophers' application the „default.lst“ would look like this:

```
Philo.java  
PhiloServer.java  
PhiloServerImpl.java  
PhiloServerIntf.java  
TracedRemote.java  
TastEnvAspect.java  
TastAbstractAspect.java  
TastInstanceAspect.java
```

- 3) This would be sufficient for a local application. The classes compiled by the AspectJ-Compiler would be the basis of the running application, but for the RMI application we have still one little thing to do:  
The Remote implementation code (PhiloServerImpl.java) now has an additional interface more (TracedRemote). This can be proven by the preprocessed code. This class which implements the remote methods must now have their stubs built again, then the application can run finally producing trace output.

While running the application produces traces that are visualized by TMT.

A remote method invocation of the known „Dining Philosophers“ application is shown in the TMT Sequence Chart View in Figure 3.

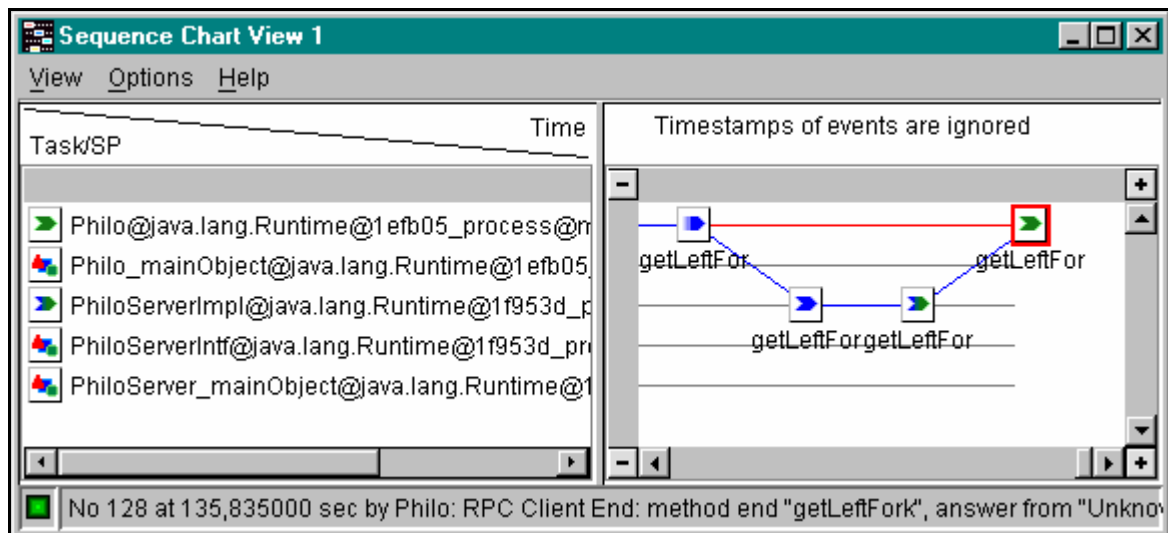


Figure 3. TMT's SequenceChart View for a remote call

## Summary

The application described herein can also be seen functioning in the demonstration „Toolkit for Aspect-oriented Tracing (TAST)“ which is held on the AOSD by Mrs. Canditt [Low 02].

## Current Status & Ongoing Work

TAST is currently under development with AspectJ1.0 and JBuilder4.0. Further aspects extending the core functionality of TAST are planned. For future adjustments in TAST and Siemens projects using AspectJ in the Development or Test process we would be glad to have support in UML Tools like TogetherJ or Rational Rose for communicating and adopting our results within the tracing projects. This would allow the design of TAST to be applicable on designs of other complex software systems.

## References

[AspectJ] <http://aspectj.org>

[Low 02] Low, Canditt, TAST - Toolkit for Aspect-oriented Tracing Demonstration at the AOSD 2002  
<http://trese.cs.utwente.nl/aosd2002/index.php?content=tast>

[Magee 99] Jeff Magee, Jeff Kramer, Concurrency: State Models & Java Programs, ISBN: 0-471-98710-7 April 1999

[TMT 00] Berg K. et al, Monitoring with TMT - Insight into Distributed Systems. In: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Application, CSREA Press, Las Vegas, 2000,  
<http://w4.siemens.de/ct/en/publications/cross-sectional/index.html>  
 TMT Homepage: <http://se1.mchp.siemens.de/se1/tmt/index.htm>

## Authors

Tina Low, Siemens AG, CT SE 1  
 Sabine Canditt, Siemens AG, CT SE 1